AN ANALYSIS OF A MODEL-BASED EVOLUTIONARY ALGORITHM:
LEARNABLE EVOLUTION MODEL

by

Mark Coletti
A Dissertation
Submitted to the
Graduate Faculty
of
George Mason University
In Partial fulfillment of
The Requirements for the Degree
of
Doctor of Philosophy
Computer Science

Committee:

_____	Dr. Kenneth De Jong, Dissertation Director
_____	Dr. Sean Luke, Committee Member
_____	Dr. Carlotta Domeniconi, Committee Member
_____	Dr. Thomasz Arciszewski, Committee Member
_____	Dr. Sanjeev Setia, Department Chair
_____	Dr. Kenneth S. Ball, Dean, The Volgenau School of Engineering

Date: _____ Spring Semester 2014
George Mason University
Fairfax, VA

An Analysis of a Model-based Evolutionary Algorithm: Learnable Evolution Model

A dissertation submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy at George Mason University

By

Mark Coletti
Master of Science
George Mason University, 2007
Bachelor of Science
Southern Polytechnic State University, 1989

Director: Dr. Kenneth De Jong, Professor
Department of Computer Science

Spring Semester 2014
George Mason University
Fairfax, VA

Dedication

In memory of Emma

Acknowledgments

I thank the following people for providing invaluable help along the way: My advisor, Dr. Kenneth De Jong, for his valued guidance. For my committee members, Dr. Sean Luke, Dr. Carlotta Domeniconi, and Dr. Thomasz Arcescweski for shepherding me through this challenging process. Dr. Guido Cervone for his valued conversations regarding LEM. Members of the Krasnow Institute for Advanced Study's Adaptive Systems Laboratory, Dr. Jeffrey K. Bassett, Eric "Siggy" Scott, Dr. Uday Kamath, and Dr. Jayshree Sarma for their sound advice. The Center for Social Complexity for my graduate research assistantship; and, in particular, Dr. Claudio Cioffi-Revilla, for allowing me to use Center equipment to write this document. My dissertation writers group, which includes Dr. Susan Farley, John MacDowell, Changwei "Coco" Liu, and others for much needed support. My late night writing crew that includes Anne Hardy, Melisse Ilhan, Charles Cressey, Andy Thrasher, and Dr. April Mattix. And, finally, my parents, brother, and sister-in-law for their emotional support.

I also thank the Krasnow Institute, Starbucks, Saxbys, Caribou Coffee, and Panera for providing safe and comfortable writing havens, sometimes in the wee dark hours of the morning.

Table of Contents

List of Tables

List of Figures

Abstract

AN ANALYSIS OF A MODEL-BASED EVOLUTIONARY ALGORITHM: LEARNABLE EVOLUTION MODEL

Mark Coletti, PhD

George Mason University, 2014

Dissertation Director: Dr. Kenneth De Jong

An evolutionary algorithm (EA) is a biologically inspired metaheuristic that uses mutation, crossover, reproduction, and selection operators to evolve solutions for a given problem. Learnable Evolution Model (LEM) is an EA that has an evolutionary algorithm component that works in tandem with a machine learner to collaboratively create populations of individuals. The machine learner infers rules from best and least fit individuals, and then this knowledge is exploited to improve the quality of offspring.

Unfortunately, most of the extant work on LEM has been *ad hoc*, and so there does not exist a deep understanding of how LEM works. And this lack of understanding, in turn, means that there is no set of best practices for implementing LEM. For example, most LEM implementations use rules that describe value ranges corresponding to areas of higher fitness in which offspring should be created. However, we do not know the efficacy of different approaches for sampling those intervals. Also, we do not have sufficient guidance for assembling training sets of positive and negative examples from populations from which the ML component can learn.

This research addresses those open issues by exploring three different rule interval sampling approaches as well as three different training set configurations on a number of test problems that are representative of the types of problems that practitioners may encounter. Using the machine learner to create offspring induces a unique emergent selection pressure separate from the selection pressure that manifests from parent and survivor selection; an outcome of this research is a partially ordered set of the impact that these rule interval sampling approaches and training set configurations have on this selection pressure that practitioners can use for implementation guidance. That is, a practitioner can modulate selection pressure by traversing a set of design configurations within a Hasse graph defined by partially ordered selection pressure.

Chapter 1: Introduction

In this chapter I first briefly describe evolutionary algorithms and Model-based Evolutionary Algorithms (MBEAs). I then discuss my motivation for focusing on a specific MBEA, Learnable Evolution Model (LEM), and the approach I used to addressing its open issues. Finally, I relate the contributions I made towards our understanding of this algorithm.

1.1 Evolutionary Algorithms

Evolutionary algorithms (EAs) are a biologically inspired approach to problem solving, and their very nature makes it straightforward to construct an EA to solve a given problem. That is, posed solutions to real-world solutions can be recast as individuals in an artificial ecosystem where their respective quality serves as a measure of fitness. Then parent selection, survival-of-the-fittest, and reproduction processes similar to their biological analogs can, over time, "breed" better solutions. EAs have been successfully applied in this way to a variety of problems that include real-value function optimization, job-shop scheduling, planning, design, and so on [De Jong, 2006].

Generally, an evolutionary algorithm consists of a set of individuals that each represents a proposed solution to a problem of interest. Associated with each individual is a fitness value that indicates the quality of that solution. Each individual is comprised of a set of genes that reflect some aspect of the problem to be solved. E.g., one set of genes may represent design parameters for an engine simulation, for a different problem the genes may represent a tour for a Traveling Salesman problem, and a still different set of genes may by comprised of an S-expression corresponding to a simple equation to solve a given problem. New individuals are created by selecting individuals to be parents and then using some form of reproduction to create offspring from those selected parents. Reproduction typically

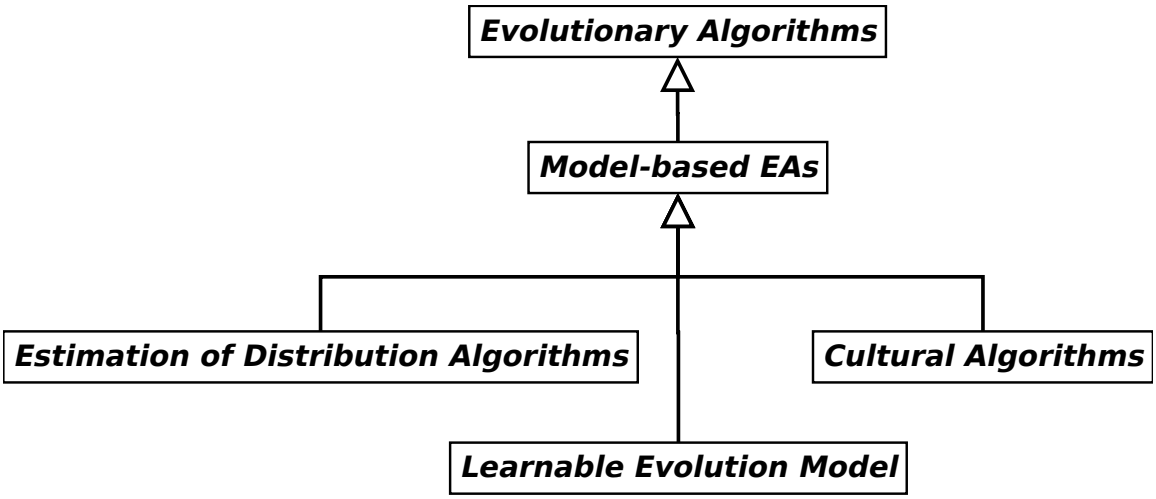


Figure 1.1: This diagram depicts the hierarchy of Model-based Evolutionary Algorithms (MBEAs). MBEAs are a type of evolutionary algorithm that derives a model from a population, and then uses that model to create offspring. Cultural Algorithms (CAs) keep knowledge of the best individuals in a "belief space" from which it influences operators. Estimation of Distribution Algorithms (EDAs) use statistical-based models, and Learnable Evolution Model (LEM) uses knowledge-based models.

takes the form of crossover of the parents' genes and applying a mutation, or perturbation, operator to them. A fitness function evaluates these new individuals to determine their respective fitnesses. Then a subset of that population is selected to be in the next generation. The process repeats until some termination condition is met.

1.2 Model-based Evolutionary Algorithms

A Model-based Evolutionary Algorithm (MBEA) is an evolutionary algorithm (EA) where a model of individuals is derived from a population, and then that model is used in some way to create offspring; Fig. 1.1 shows three types of MBEAs: Cultural Algorithms, Estimation of Distribution Algorithms, and Learnable Evolution Model.

Cultural Algorithms (CAs) that use "belief spaces" that contain information from the fittest individuals to influence EA operators [Reynolds, 1994]. For example, a belief space can contain information associated with a current set of associated "exemplar individuals; real-valued mutation operators can then skew their distributions based on this information

Table 1.1: This enumerates the estimated number of Model-based Evolutionary Algorithm publications [Google Scholar, 2014], and shows that Learnable Evolution Model (LEM) has the least number of publications.

Model-based Evolutionary Algorithms	# publications
Cultural Algorithms	2,300
Estimation of Distribution Algorithms[a]	5,110
Learnable Evolution Model	287

[a] Note that the number of publications for EDAs is underestimated given that many EDA publications refer to the specific subtype of EDA; e.g., a CMA-ES publication may not mention that CMA-ES' are EDAs at all, and yet that paper would otherwise count towards the total number of EDA publications.

[Chung, 1997].

Estimation of Distribution Algorithms (EDAs) are MBEAs that eschew traditional evolutionary algorithm operators, such as mutation and crossover, and instead create individuals from probability distributions computed from an existing population. In a sense, EDA offspring are not directly made from selected parents as with other EAs, but are instead created via a layer of indirection manifested in the form of statistical models.

In its simplest form, *Learnable Evolution Model* (LEM) uses a machine learner to infer rules derived from gene values of the least and most fit individuals and then exploits that knowledge when creating children [Michalski, 2000]. The machine learner then infers rules that describe allele ranges in some genes that belong to fitter individuals. Gene values in newly created individuals are stochastically generated within those ranges. LEM also has a number of options for self-adaptive behavior, which are described in the next chapter.

1.3 Motivation

Tbl. 1.1 lists the estimated number of publications for each of the previously described model-based evolutionary algorithms. This table shows that of all the MBEAs, Learnable Evolution Model by far has the least number of extant publications, which by that criteria alone indicates that of the set of listed MBEAs LEM is the least understood. However, the relative paucity of existing LEM research is not, by itself, sufficient motivation to further explore LEM — there needs to be more compelling reasons to do so.

As practitioners we desire a well stocked toolbox of algorithms from which to choose an ideal fit to apply to a problem of interest; the greater the number of choices the likelier we are able to find a good fit for a given problem. After all, we have to be mindful of the No Free Lunch Theorem that dictates that there does not exist any single "one size fits all" algorithm [Macready and Wolpert, 1995]. With better understanding we may find a suitable place for LEM in our toolbox for algorithms and, moreover, appropriately tune it for selected problems.

We already know that LEM has been usefully applied to a number of optimization problems. However, one type of problem where LEM may be particularly suited are problems with fitness functions that involve lengthy computations; e.g., a large simulation may have to be run for each evaluation. This type of problem means smaller parent populations and a limited birth budget as dictated by the expensive fitness evaluation. Consequently there are fewer individuals exploring the search space for less time, which makes it less likely to find viable solutions. LEM may be a good match for those types of problems because the machine learning component may yield more leverage with the smaller populations and birth budgets. Indeed, LEM has already been applied in this way to pinpoint the source for a chemical release that involved running atmospheric simulations to evaluate each individual [Cervone and Franzese, 2011].

There has been some work focused on getting a deeper understanding of LEM. First, the impact of using binary tournament selection on LEM's performance was studied [Coletti, 2009]. Also, it was observed that using rules learned from a ML to create offspring was a novel source of selection pressure separate from selection pressure induced from parent and survival selection, and that this unique selection pressure responded to training set sizes and the time rules are allowed to linger [Coletti, 2012].

However, this research barely scratches the surface of our ignorance regarding LEM. So the focus of this work was to build on this prior research to further our understanding of LEM to thereby provide implementation guidance to practitioners. I next describe my approach to expanding our understanding of LEM.

1.4 Methodology

As we will see in the next chapter, LEM is very complex — the original specification described mechanisms for dynamic, self-adaptive behavior and also covered a lot of optional, additional functionality [Michalski, 2000]. LEM has changed over the years, and with the passage of time LEM's complexity has only increased [Michalski et al., 2007, Wojtusiak, 2008]. Unfortunately this inherent complexity makes analysis particularly challenging, so the first step I took work was to create a stripped down version that would make analysis tractable. I refer to this stripped down LEM as Simplified Learnable Evolution Model (SLEM), which is described in more detail in §3.2 on page 24.

Again, in prior work I observed that the machine learner induced a separate and unique form of selection pressure [Coletti, 2012], and I decided for this research to use that work as a baseline. That is, I wanted to isolate this ML-induced selection pressure to learn the influence of different form of rule interval sampling strategies and training set configurations would have on it.

There are two legacy sources of selection pressure in evolutionary algorithms: parent selection and survival selection. That is, design choices regarding how parents are selected for offspring creation and for determining what individuals can linger from generation to generation can effect a selection bias based on fitness. One can intuit that always picking the best individuals is going to be greedier than stochastically selecting individuals based in proportion to their fitness since the latter selection scheme has a chance for selecting inferior individuals that would never be selected in the former. To that end, I reduced selection pressure sourced from parent selection by deterministically selecting each parent to create a single offspring; since fitness had no role in selecting parents, this eliminated that source of selection pressure. Similarly I used non-overlapping generations; i.e., I discarded all the parents with each generation, and all the children became the parents for the next generation. Again, since fitness had no role in the decision on what individuals survived to the next generation, that source of selection pressure was muted. Therefore, any observed convergence behavior is due to the influence of the machine learner and not due to any

preference by fitness for parents or surviving individuals.

Why focus on rule interval sampling strategies and training set configurations? I focused on those facets of LEM design because those are some of the first design decisions that an implementor must face when assembling these kinds of systems, and there does not currently exist any research providing useful guidance. This work fills that void.

I chose to exercise SLEM on a number of test problems that were representative of some of the problems a practitioner may encounter. These test problems are described in more detail in §3.3 on page 27.

Next I describe what contributions to the field of evolutionary computation my research yielded.

1.5 Contributions

I learned that one of the first problems that a practitioner may encounter when implementing this type of MBEA is that if they choose to use a decision tree-based machine learner, such as C4.5 [Quinlan, 1993], that many rules will have only one bound — i.e., just a lower or upper bound, and not both. This is a problem since we need both bounds in which to sample new values for offspring.

In this work I explore three different means of assigning values to missing bounds and their respective influence on selection pressure. That is, assigning missing bound values from the values used to initialize the population, values of the most extreme gene value, or values from the most extreme values of only the fittest parents. I found that assigning missing bound values from those from which the population was initialized was too disruptive; and I found that assigning missing bounds from only the fittest parents exhibited higher selection pressure than if extrema gene values from the entire set of parents was used.

Once we have ensured a complete rule interval bounds, we are then left to sample within those bounds when creating offspring. I investigated three different mechanisms for rule interval sampling. The first used a uniform distribution, which is the distribution found in legacy LEM implementations. The second considers using a Gaussian distribution over

the intervals using two and then three standard deviations. And, lastly, an approach similar to EDAs in that intervals were divided into bins, frequencies were computed based on the number of genes that corresponded to each bin, and a probability distribution was then, in turn, computed from those frequencies. However, since it is possible for some bins to get zero or a relatively small number of corresponding gene frequencies, a uniform distribution was blended with the final probability distribution. The amount of uniform distribution to be so blended could be modulated by the user to control the amount of selection pressure.

I found that the uniform distribution yielded the least amount of selection pressure of the three approaches. The next least selection pressure was exhibited by the histogram-based approach; moreover, the more of a uniform distribution that was blended with the histogram frequency-based distribution, the less the selection pressure. The Gaussian distribution exhibited the most selection pressure with three standard deviations being used to sample an interval showing the highest selection pressure.

I also explored the influence three training set configurations had on the ML induced selection pressure. These three configurations, inspired by prior work [Wallin and Ryan, 2009], included one where the top- and bottom-most third parents as sorted by fitness were used as positive and negative training sets for the machine learner; another configuration whereby the parent population was evenly split; and, finally, a configuration where the population was split, but 15% of the parent population extrema were dropped for consideration by the machine learner.

The last training set configuration, where the very best and the very worst were elided from the training sets, was originally considered by Wallin and Ryan as an experimental control. That is, they wished to learn whether separating the positive and negative examples by a buffer of mediocre individuals for LEM-like hybrids was necessary, so they considered the traditional top and bottom third configuration and one with the population evenly divided between the best and worst parents. However, the latter configuration would have more individuals than the former, so to rule out effects of training set size, they made a third training set configuration that was also split as with the original second configuration,

but with enough extrema of the best and worst removed to make the size the same as the original training set configuration, the one that split the parent population into thirds of the best, mediocre, and worst parents.

Wallin and Ryan found that this "dropped-extrema" approach converged to better solutions than the other training set configurations. They speculated that this was because this configuration introduced a novel mechanism for escaping local minima. That is, the machine learner would not "see" the very best because they were in the top 15%; however, when the population dipped into a local minima, a new set of parents with higher fitnesses would be added to the population thus pushing some of the previously saved elite into the machine learner's view. Since it was likely that these newly visible parents were not in the local optima, then these newly visible parents could potentially navigate the population out of the local optima either by influencing the ML to generate new rules to explore other areas or by otherwise directly contributing genetic material to offspring via crossover or cloning.

I found that of the three aforementioned training set configurations that the "dropped-extrema" approach exhibited the least selection pressure, followed by the approach where the training sets were evenly split and, finally, the approach that divided the best and least with a set of mediocre individuals had, in general, the most selection pressure. Moreover, since this approach did not find better solutions than the others supports Wallin and Ryan's conjecture. That is, since I discard parents every generation, it is impossible for elite parents to "hide" in the upper 15% to be used to escape from local optima.

However, these three different training set configuration were not the only aspect regarding assembling training sets that I explored. The original LEM specification described two means for creating training sets of positive and negative examples to be given to the ML. Both means have in common that parents are first sorted by fitness. However, the first approach then selects the training sets by fitness rank; e.g., the top- and bottom-most 20 individuals are chosen for training sets. The second approach is similar except that the training sets are selected by fitness percentage; e.g., of the full range of parent fitness values, the individuals that are in the top 20% are chosen for one set, and those in the bottom 20%

are selected for the other. These two approaches are discussed in more detail in §2.1.1 on page 14. So, in the latter approach it is possible for training set sizes to be very different, whereas in the former they are always the same. Indeed I found that that there were more individuals in the positive training sets than the negative due to the underlying problem topology. This is also discussed in more detail in §5.3.2 on page 135.

The effects of design decisions regarding rule interval sampling and training set configurations on selection pressure by themselves is not useful to practitioners since they will have to commit to decisions regarding all of those implementation choices. Given that the relative selection pressures for rule interval and training set configuration approaches defined partially ordered sets, I blended these partial orders into a single partially ordered set. This single poset gave an overall map of design choices for practitioners in which to navigate through design choices that affect selection pressure, and is the focus of Ch. 6 starting on page 153.

One other contribution I made was discerning that the initial parent population defines an implicit bounds for exploration. That is, subsequent generations will not stray beyond the region of the initial parent population, which may be a problem if, by happenstance, the global optimum is outside this region. I did explore using a Gaussian distribution to sample rule intervals as a means of escaping these implicit bounds. I found that wide Gaussian distributions, i.e., those that had two standard deviations within rule intervals, could escape these implicit bounds, but those with narrower distributions, i.e., those that used three standard deviations, not only would not escape these implicit bounds, but would define an implicit bounds that was more conservative than if a uniform distribution was used to sample rule intervals.

Some algorithms have a bias such that they gain a performance boost with problems that have axially aligned optima; this bias can be foiled by rotating the problem [Whitley et al., 1995, Bäck et al., 1997]. The rule interval representation used in most LEM implementations has a rectilinear bias that may give it better traction on problems with axially aligned optima. To explore this I chose two test problems that could be randomly

rotated, the Rastrigin and Griewangk functions. I found there were mixed results. First, randomly rotating the Rastrigin function did seem to somewhat impair the performance of SLEM; however, in the case of the Griewangk function, rotation either made no performance difference, or rotation made it easier to converge towards the global optima.

Finally, I found that the machine learner was unable to gain any traction for a problem that had large basins of attraction with a global optimum that was very challenging to find. Because of the size of the local optima there was not enough difference between the positive and negative training sets for the machine learner to learn anything useful. Since the ML was the sole source of novelty for SLEM, that meant that SLEM was unable to make any progress from the very start. Therefore, we should be cautious about similar problems with large basins of attraction such that the ML component is unable to gain any traction.

I invented two EC-related visualization tools to aid in analysis. First I used heat maps to portray by-generation frequency data, which I used to depict the frequency of rules learned per generation by the ML as well as the number of genes cited by rules. Second, I visually compare two runs using 95% confidence intervals for Wilcoxon rank sum tests by generation; this shows details of convergence effects that are otherwise missed by more traditional methods that just examine end-of-run or convergence time statistics.

1.6 Summary

In this chapter I provided some overall context for this work. I briefly defined evolutionary algorithms and further described a framework for model-based evolutionary algorithms. This research focused on a particular type of model-based evolutionary algorithm, Learnable Evolution Model, and that my motivation for doing so was to expand our understanding of this algorithm so that practitioners could be better informed. I described the approach I used for this research and enumerated my contributions.

The next chapter covers the background to provide better context and motivation for this work. Following that I cover the methodology. Then I share research regarding the impact of different approaches to sampling rule intervals on the emergent selection pressure;

following that I do the same, but for different training set configurations. Next, I cover the combined effects on selection pressure of both the rule interval sampling strategies and training set configurations. And, finally, I summarize the contributions of this work as well as describe future research directions.

Chapter 2: Background

This chapter reviews the Model-based Evolutionary Algorithms enumerated in the previous chapter with a particular focus on Learnable Evolution Model (LEM).

2.1 Learnable Evolution Model

Learnable Evolution Model is an evolutionary algorithm augmented by a machine learner. A machine learner observes most and least fit individuals to form a model comprised of rules; these rules are then used to create offspring [Michalski, 2000]. Offspring creation benefits from the model's influence by biasing new gene values in favor of areas of higher fitness.

Algorithm 1 Learnable Evolution Model

1: $P \leftarrow$initialize() ▷ stochastically create initial population
2: ComputeFitnesses(P)
3: $b \leftarrow \varnothing$ ▷ b is the best so far
4: **repeat**
5: $i_m \leftarrow 0$
6: **repeat** ▷ machine learning mode
7: $b \leftarrow \text{best}(P, b)$ ▷ Update b with best from P, or keep b if there are none better
8: $P^{\oplus} \leftarrow \text{high}(P)$
9: $P^{\ominus} \leftarrow \text{low}(P)$
10: $M \leftarrow \text{learn}(P^{\oplus}, P^{\ominus})$ ▷ learn a model from best and worst
11: $C \leftarrow \text{instantiate}(P, M)$ ▷ create offspring using model
12: ComputeFitnesses(C)
13: $P \subset P \cup C$ ▷ cull population for next generation
14: $i_m \leftarrow i_m + 1$
15: **until** |fitness(b) - fitness(best(P, b))| $< \tau_l \vee i_m = m_p$ ▷ repeat until the population is not improving
16: ▷ or exhausted number of ML mode iterations
17: $i_e \leftarrow 0$
18: **repeat** ▷ Darwinian evolution mode
19: $b \leftarrow \text{best}(P, b)$
20: $C \leftarrow \text{breed}(P)$ ▷ use mutation and optionally crossover to create offspring
21: ComputeFitnesses(C)
22: $P \subset P \cup C$
23: $i_e \leftarrow i_e + 1$
24: **until** |fitness(b) - fitness(best(P, b))| $< \tau_d \vee i_e = m_e$ ▷ repeat until population is not improving
25: ▷ or exhausted number of EA mode iterations
26: **until** halt()
27: **return** b

Alg. 1 depicts the original LEM specification. An initial random parent population is created and then LEM enters *machine learning mode* — as denoted by lines 6 through 16. The best and worst subsets of the parent population as selected by high() and low(), respectively, are presented as training sets to the machine learner, which then infers a model describing the problem space's areas of higher fitness. (How high() and low() can be implemented is discussed in more detail in §2.1.2.) A new population of offspring is created using this model, and then a subset of parents and offspring are selected as survivors for the next generation; the subset that survives uses a form of truncation selection in that the least fit individuals are replaced by fitter offspring. This process repeats until the difference in fitnesses between the best individual so far, b, and the current generation's best is below some threshold, τ_d, which is known as the *learn-threshold* in the literature [Michalski, 2000]; the process can also end if the number of iterations budgeted for machine learning mode, m_p, or what is called *learn-probe*, is exceeded. Once either of these conditions is true then execution falls into *Darwinian evolution mode*, which is denoted by lines 18 through 24.

Darwinian evolution mode is essentially a plain EA that uses mutation and/or crossover operators to create offspring. Next, a subset of the parents and offspring survive for the next generation, though in evolution mode a legacy EA survival mechanism such as binary tournament, fitness proportional, or tournament selection can be used. Similarly if the change in the best fitness so far is below τ_d, otherwise known as *dar-threshold*, then execution falls out of the Darwinian evolution mode where the overall algorithm either halts, or spends more time in machine learning mode; execution will also leave evolution mode if it exceeds its permutation budget, as denoted by m_e, which is known as *dar-probe*.

LEM allows for some implementation variation. For example, the machine learning and evolution modes need not be in the given order, so the evolution mode can come first. Moreover in an implementation variant called *uniLEM mode* the evolution mode is skipped altogether with the entirety of the population being generated from the model. Other options allow for injecting entirely new individuals if they are above a certain fitness threshold, keeping models from previous generations, and using a generate-and-test framework such

that new individuals are randomly generated and kept only if they do not match a low performance description. There are other design variations, but detailed discussion of them is beyond the scope of this work. In any case, there has been no research done on the relative merits of *duoLEM* and *uniLEM*, or for the effects of different approaches for switching between machine learning and Darwinian modes.

Next, I discuss in more detail in how high() and low() divide up the training sets, and then I cover the different mechanisms for instantiating offspring from learned models.

2.1.1 Means of selecting hi- and low-performing training sets

The original LEM specification had two strategies for selecting the best and worst subpopulations: picking by rank or by fitness percentage. Picking by rank entailed taking a static number of the better individuals as the best and a similar set number of the poorest performing individuals as the worst. Picking by fitness percentage is similar to picking by rank, except the individuals are selected based on being above or below a certain portion of the total fitness range of the current generation. Regardless of the training set selection strategy, the population could be evenly split by rank or percentage to make up these training sets, or a middle "mediocre" part of the population could divide the best and least fit individuals.

For example, out of a population of, say, 100 individuals sorted by fitness, if selecting by rank, one might pick the top 20 individuals as the best, and the bottom 20 individuals as the worst regardless of the distribution of fitnesses. Alternatively, one could take the same sorted set of individuals and take the top 20% by fitness as the best, and those in the bottom 20% as the worst. Fig. 2.1 shows an example of both of these strategies. 100 individuals were randomly generated and sorted in descending order by fitness. Vertical dashed lines divide the population into high, mediocre, and low performing individuals as well as using •, ▲, and ■, to depict high, mediocre, and low valued individuals, respectively, as determined by fitness rank. The same subgroups as divided by proportional fitness values are shown divided by horizontal dotted lines and different colors with red, blue, and green

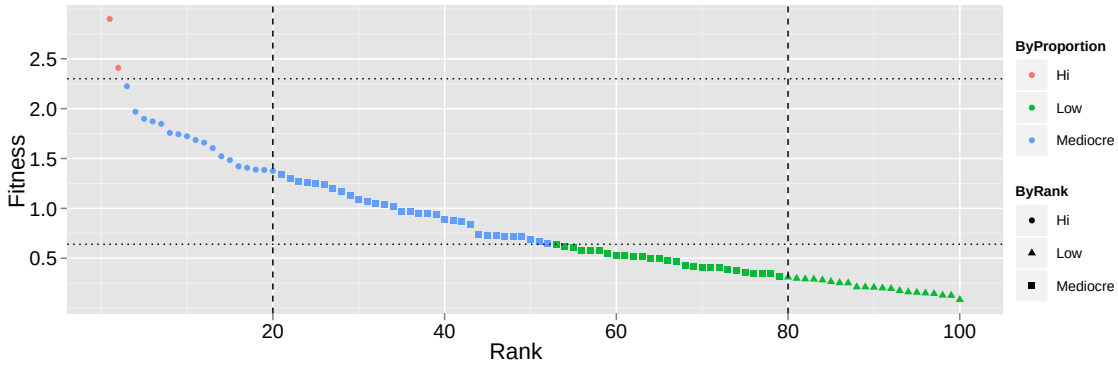


Figure 2.1: This shows the results of the two LEM high and low performance sub-population selection strategies. The three subgroups of high, mediocre, and low performing individuals as selected by rank are depicted by different shapes as well as vertical dashed lines showing the subgroup boundaries. By contrast, the same subgroups as selected by the top and bottom 20% of fitness values are shown in different colors with horizontal dotted lines indicating the subgroup boundaries.

showing high, mediocre, and low fitness individuals, respectively.

The choice of training set selection strategy can have starkly different outcomes in training set composition. In the given example, choosing high and low performing individuals by fitness proportion has only two individuals in the high performing set and 48 in the low performing set, whereas the same set of individuals will always have 20 in each set, respectively, when using the by-rank selection strategy. Naturally this distribution is not necessarily typical for all runs, but does show that this design decision can potentially lead to disparate outcomes with regards to make up of training sets shown the machine learner.

There has been some prior work with regards to the influence of different training set sizes. Specifically, it was shown that increasing the population sizes increased the selection pressure that originated from the machine learner, though this research only employed a simple fitness landscape [Coletti, 2012].

2.1.2 Instantiating offspring from the learned model

How offspring are created from the models inferred from the machine learner depends largely on the machine learner, the representation of the individuals, and the problem to be solved.

These models, according to the original LEM specification, may comprise a set of descriptions for all or a subset of genes. Some of these descriptions can be kept over multiple generations. A single description from the current generation can be selected, and any older descriptions from previous generations that describe higher performing individuals can be applied "in a cascade fashion" [Michalski, 2000].

A description will contain knowledge regarding valid values or ranges for a subset of genes. E.g., that a given gene value should be in a given interval, or be comprised of a set of specific nominal values. When creating offspring, if the gene is a categorical variable, such as from $\{red, blue, green\}$, that gene would have one of the "good" values from the set randomly selected using some distribution; similarly if the description is a real valued interval, then a new gene value will be generated from within that interval using some distribution – likely a uniform distribution, though the text suggests as an alternative a distribution biased towards the center of intervals. Again, this depends on the underlying gene representation and the type of machine learner used, and as yet there has been no research done with regards to different description instantiation mechanisms.

A description may only cite a subset of the genes. That is, some of the genes are "free" variables not covered by any description. In which case, the corresponding offspring genes are either arbitrarily assigned a value from an arbitrarily selected high performing group individual (i.e., implementing a sort of global crossover), or is randomly assigned a value from within the original domain range. However, some implementations choose to apply a mutation operator to these uncited genes. There has yet been no established design decision guidance with regards to dealing with uncited genes.

2.1.3 Historical implementations

The first LEM implementation, LEM1, used a fixed length real-valued vector, a set static mutation step size, uniform crossover, the machine learner AQ15c [Michalski, 1986], and was applied to a number of real-valued optimization problems [Michalski and Zhang, 1999].

LEM2 used a later version of AQ, AQ18 [Kaufman and Michalski, 1999a], which was

designed to handle noisy and inconsistent data; LEM2 also used an adaptive interval mechanism to gradually narrow areas of interest per gene as a run progressed. LEM2 also made some improvements with regards to real-value optimization, and also allowed for multiple rules to generate individuals instead of just the "best" rule. It also added parsimony pressure to rule complexity such that rule fitness' were penalized in proportion to their number of variables. Also new random individuals are injected into the population once they have converged instead of doing a complete restart. Finally, LEM2 introduced the aforementioned "uniLEM" mode whereby the standard EA mutation and crossover operators were disabled, so all novelty was generated from the inferred rules.

LEM3 used AQ21 [Wojtusiak et al., 2006] and allowed some flexibility to generate gene values outside those dictated by learned rules; it also had a local search capability, which made it similar to a memetic algorithm; and LEM3 added or removed new attributes as needed to transform genes into a better fitting representation space [Michalski et al., 2006]. LEM3 also allows for heterogeneous variable types such as cyclic, numeric, nominal, etc. If the population is too homogeneous, LEM3 will just mutate individuals or mutate and adjust the discretization; if there is still no improvement, then the population is restarted with entirely new individuals. Rules for real-value problems are generally in the form of intervals for some genes. New values can be uniformly generated or use a normal distribution centered at the interval; also a bias can be given to those individuals that are furthest from a negative example.

The AQ family of machine learners were not the only machine learners used to implement LEM. C4.5 [Quinlan, 1993] has also been employed in a number of implementations [Coletti, 2009, Coletti, 2002, Coletti, 2012]. LEM has also been implemented using a KNN approach although that is more like a memetic algorithm due to its local search characteristics [Sheri and Corne, 2008]. A multiobjective variant also exists, LEMMO, [Jourdan et al., 2004], which also uses C4.5.

SI3E is an implementation of LEM that uses ID3 [Quinlan, 1986] on problems using a binary representation [Llora and Goldberg, 2003]. The original LEM specification arbitrarily

assigns gene values from one of the high performing individuals for genes not cited by rules. SI3E takes a different approach. It uses a PBIL-like mechanism of updating a probabilistic vector from which it assigns values to genes not covered by rules.

2.2 Other Model-based Evolutionary Algorithms

In this section I describe two different Model-based Evolutionary Algorithms (MBEA): Cultural Algorithms and Estimation of Distribution Algorithms (EDA).

2.2.1 Cultural Algorithms

A Cultural Algorithm (CA) updates a "belief space" from a subset of the better individuals in a population. Depending on the implementation, the belief space is comprised of operators that can influence the EA at the population, individual, and component level. The probability of a given belief space operator being selected is a function of the relative fitness of individuals that was influenced by that operator; the better the individuals the likelier that the belief space operator is selected [Reynolds, 1999].

Belief spaces do not necessarily contain homogeneous operators or objects; for example in the Cultural Algorithm Evolutionary Program (CAEP) they can have normative and situational knowledge [Chung, 1997]. In this work, normative knowledge for real-value optimization problems were viable seeming real-value intervals for each gene, and were also used to influence the mutation rate and step size. Situational knowledge is represented as "exemplars" – actual copies of individuals selected to update the belief space, and these were used to influence the direction of mutation for each variable. Each generation the exemplars, which in this case was just one or two individuals, were replaced by any superior individuals.

2.2.2 Estimation of Distribution Algorithms

An Estimation of Distribution Algorithm (EDA) builds a statistical model of the top performing individuals, and then uses that model to create offspring. There are three broad categories of EDAs: univariate, bivariate, and multivariate EDAs.

Univariate EDAs use probability distributions from allele frequencies computed from an existing population — these allele frequencies are then used as a probability distribution for generating genes for newly created individuals [Mühlenbein and Paaß, 1996]. E.g., Population Based Incremental Learning (PBIL) [Baluja, 1994] and Compact Genetic Algorithms (cGAs) [Harik et al., 1999] update a probability vector based on allele frequencies from a selected individual.

Unfortunately univariate EDAs presume that the information represented by the genes is independent, that there is no inter-gene linkage. As a step towards mitigating this presumption there exist bivariate EDAs that accommodate pair-wise gene dependencies. One such bivariate EDA is Mutual-Information-Maximizing Input Clustering (MIMIC) that computes a chain of pair-wise conditional probabilities that estimates the true joint distribution using a greedy approach [De Bonet et al., 1996]. Also, a pair-wise ordering of conditional probabilities can be represented as trees, which can be computed as a single graph [Baluja and Davies, 1997] or as a forest of disjoint trees [Pelikan et al., 1999].

There are also multivariate EDAs, such as the Extended Compact Genetic Algorithm (ECGA) which uses a Minimum Description Length heuristic to compute inter-gene linkages [Harik et al., 2006]; the Bayesian Optimization Algorithm (BOA) creates a Bayesian network based on selected sub-population from which new individuals are generated [Pelikan et al., 1999]; and, lastly, the Covariance Matrix Adaption Evolution Strategy (CMA-ES) uses a covariance matrix computed from a selected sub-population to generate new individuals [Hansen and Ostermeier, 2001].

2.3 Related Work

LEM uses learned rules that identify regions of higher fitness in which to create offspring. A Tabu search can exclude previous visited solutions [Glover, 1989], and a ML can learn rules that identify regions of inferior fitness to similarly used to wall off regions for exploration. For example, the results of crossover and mutation operators can be rejected in a generate-and-test mechanism if offspring match rules corresponding to inferior solutions [Sebag et al., 1996].

Memetic Algorithms are a blend of evolutionary algorithms and local search. Parents are selected to produce offspring as with legacy evolutionary algorithms, but before offspring are accepted into the population they are moved to a local optima via a hill-climbing operator [Mascato, 1989, Radcliffe and Surry, 1994]. However, sometimes a MA can employ a model to aid in local search, such as using an ANN to learn local problem topology [Guimaraes et al., 2007].

2.4 Summary

In this chapter I described Learnable Evolution Model and I shared some historical perspective on its implementations. Moreover I discussed related Model-based Evolutionary Algorithms, Cultural Algorithms and Estimation of Distribution Algorithms, as well as covering some work related to Model-based Evolutionary Algorithms.

In the next chapter I share my research objectives and the approach I used to achieve them.

Chapter 3: Methodology

The focus of this research is to contribute a deeper understanding of some open issues of these rule-based generative machine learner / evolutionary algorithm hybrids as described in the previous chapter; and, where possible, use this understanding to provide design guidance to practitioners. In this chapter I describe the means by which I will explore these open issues in this work. To that end, I relate the research objectives, describe the algorithm that will be exercised on a set of benchmark functions, enumerate and describe those functions, list the common experiment run-time parameters used in subsequent chapters, and the metrics to be used.

3.1 Research Objectives

Different legacy selection operators have a unique associated selection pressure [Blickle and Thiele, 1996], which can be modulated in one or more ways. For example, tournament selection works by selecting the best individual from a set of individuals randomly drawn from a population — the larger the set of randomly drawn individuals, the higher the selection pressure [De Jong, 2006]. I have shown how using rules learned from a machine learner to create offspring in an evolutionary algorithm adds a novel form of selection pressure [Coletti, 2012]. I have also determined in the same research, as with legacy selection operators, that this attendant selection pressure can also be modulated; in this case, by changing the size of the positive and negative training set sizes and the maximum time that rules can linger .

Unfortunately that research did not go into much depth to explain the observed behavior — it only sufficed to show that this behavior exists and to give hints as to how training set sizes and rule age affects performance. Moreover, only a very simple fitness landscape was

used, the spheroid function, which is not representative of the type of problems a practitioner may encounter. More exploration is needed to determine the underlying mechanisms behind adjusting the selection pressure for these EA/ML hybrids via manipulating training set sizes and rule persistence.

Adjusting the training set sizes and the length of time rules can persist are not the only means of influencing this selection pressure. Given that the rules describe regions of the search space that the machine learner thinks has higher fitness, it then it falls on the practitioner to decide how to explore that space. Typically this is done by sampling those regions of interest, either by using a uniform sampling or some other probability distribution. The intuition is that different sampling strategies will affect how quickly the EA/ML converges; and, if so, it is important to know how and why this happens to give appropriate guidance to implementors.

In short, my research objectives are to further explore the effects of different policies for high fitness area sampling, training sets, and persistent rules. These three areas of focus are described in more detail in the following subsections.

3.1.1 Interval Sampling Policies

The machine learner, which is described later in §3.4, learns rules that represent regions of higher fitness from the training sets of best- and least-fit parents. These rules are conjunctions of $[l, u]_i$ where l is a lower bound, u an upper, and which is applied to the ith gene when creating offspring. So the first major design decision an implementor faces is how to sample regions described by those rules. This is particularly important because how intervals are sampled when creating offspring can influence convergence behavior.

Naturally, the first probability distribution to examine would be the uniform distribution. Indeed, it is the usual interval sampling strategy for prior LEM implementations [Cervone, 1999,Coletti et al., 1999,Wojtusiak, 2008,Coletti, 2002,Cervone et al., 2002]. However, it is safe to assume that the fitness landscapes for interesting problems are not themselves smooth, which suggests that other rule interval sampling approaches might exhibit different

behavior for various fitness landscapes. I consider such probability distributions as Gaussian and histogram-based, which I describe in turn.

The Gaussian probability distribution is commonly used in a number of disciplines, and has been informally tried in previous LEM implementations [Cervone, 2012]. Using a Gaussian distribution to sample rule intervals does naïvely presume that the regions of higher fitness are likelier at rule interval midpoints, but does allow for sampling beyond the rule intervals. The latter may be particularly important for scenarios where the global optimum happens to be outside the initial population. That is, if the ML is the sole source of novelty which learns from the current parent population, it is likelier to learn regions of higher fitness that that population occupies. In real world problems we may not know where the global optimum is located, or even if one exists. So the initial populations for EAs to solve those problems may not necessarily include global optimum because implementors made unlucky guesses as to its location — in which case it is important for the EA to explore beyond those initial population areas. Certainly one approach to compensate for this is to employ mutation, but it is nonetheless important to explore this phenomena so that practitioners are aware of this potential effect.

The uniform and Gaussian rule interval sampling approaches have the common feature of being ignorant of the actual underlying fitness landscape. At best they are likely crude approximations of the underlying terrain. One other approach is to compute a probability density distribution from histograms built from gene values similar to the one used in PBIL. That is, PBIL computes a probability distribution based on a histogram of gene values from fitter individuals that is used to generate offspring [Baluja, 1994]. Though an approach that computed a similar kind of histogram for this kind of EA/ML studied in this work has been done before [Cervone et al., 2010], the effects of this strategy on convergence has not been studied.

Ch. 4 covers the results of using uniform, Gaussian, and probability based distributions for sampling rule intervals when creating offspring.

3.1.2 Training Set Allocation Policies

Prior work has shown that the size of the machine learner training sets of the best and least fit individuals can have an influence on the emergent selection pressure [Coletti, 2012]. But, this was just preliminary work that demonstrated that this special kind of selection pressure is sensitive to training set sizes. Again, only the spheroid test function was used for generating the results, so we are unaware how different training set selection policies generalizes to different types of problems.

Moreover, there are a number of training set strategies for dividing the parents into positive and negative examples. In practice, the top third of the parents are deemed "best" and the bottom third the "worst" training examples. Other strategies that include evenly splitting the parent population as well as "growing examples from the middle" have been explored, but for binary representations [Wallin and Ryan, 2009].

As shown in §2.1.1, and depicted in Fig. 2.1, there are also two approaches for creating the positive and negative training sets – i.e., by rank or by fitness percentage. Most of the work so far with these EA/ML hybrids have used the former and not the latter approaches presumably because the latter is more involved to implement. Nonetheless, it is not known how the two different approaches compare with regards to effects on EA/ML selection pressure, and so needs further scrutiny.

These various training set allocation policies are further covered in Ch. 5 in more detail.

3.2 Simple Learnable Evolution Model

Naturally I need an implementation of a suitable EA/ML hybrid with which to explore my research objectives. What are the requirements for this implementation? Of course it should be an EA with offspring created from rules learned from a ML that observed best and least fit parents. The ML needs to support rules that can describe regions of high fitness as described in §3.1.1. However, it should also be as simple as possible — but no simpler — to make implementing and analysis tractable.

Algorithm 2 General LEM [Wojtusiak, 2007]

Create an initial population of candidate solutions
Evaluate candidate solutions in the initial population
Loop while stop criteria are not satisfied:

    Create new candidate solutions by machine learning:

        Identify groups of high- and low-performing candidate solutions
        Apply machine learning to distinguish between the groups
        Instantiate the learned hypothesis

    Evaluate fitness of the new candidate solutions
    Select a new population

Algorithm 3 Simplified Learnable Evolution Model

1: $P \leftarrow$initialize()
2: ComputeFitnesses(P)
3: $b \leftarrow \varnothing$
4: **repeat**
5: $\quad P^{\oplus} \leftarrow \text{high}(P)$
6: $\quad P^{\ominus} \leftarrow \text{low}(P)$
7: $\quad R \leftarrow \text{learn}(P^{\oplus}, P^{\ominus})$
8: $\quad C \leftarrow \text{cloneAndApplyRules}(P, R)$ ▷ create offspring from parents and rules
9: $\quad$ ComputeFitnesses(C)
10: $\quad b \leftarrow \text{best}(P, b)$
11: $\quad P \subset P \cup C$
12: **until** halt()
13: **return** b

This last requirement rules out using a full LEM implementation. As §2.1 has shown, LEM is a very complex algorithm with a lot of moving parts [Michalski, 2000], which makes it particularly difficult to analyze. Since the original LEM specification's complexity is an impediment to analysis, one approach to gaining such an understanding is to scrutinize a more generalized LEM, instead. Indeed, one such generalized LEM has gotten passing mention [Wojtusiak, 2007], and is shown in Algorithm 2.

This generalized LEM is very similar to a basic evolutionary algorithm with one critical change, that of using the ML to infer a model from the "high- and low-performing candidate solutions" from which new individuals are "instantiated." Unfortunately we have now swung too far from an overly complex specification to one that suffers from too much generality. We need to add back in a little specificity to make analysis tractable.

Algorithm 3 depicts Simplified Learnable Evolution Model (SLEM), which is General LEM with enough added back specificity to make analysis practical as well as providing an algorithm that still achieves the objective — that is, of making computationally burdensome

Algorithm 4 cloneAndApplyRules()

```
1: function CLONEANDAPPLYRULES(P,R)
2:     C ← ∅                                   ▷ initialize set of children to be returned
3:     n ← 0                                   ▷ current child
4:     for all p ∈ P do                        ▷ for each parent, p, from population, P
5:         c ← clone(p)                        ▷ clone a child from current parent
6:         r ← selectRule(R)                   ▷ select a rule, r, from rules, R
7:         for all r_i ∈ r do                  ▷ for each rule interval, r_i, within selected rule that corresponds to a single gene i
8:             c_i ← sample(r_i, D)            ▷ assign sample from distribution, D, within r_i to ith gene of offspring
9:         C_n ← c                             ▷ add new offspring
10:        n ← n + 1
       return C
```

problems feasible for an EA with small populations and limited birth budgets by augmenting the EA with a ML.

Alg. 4 describes how offspring are created using these rules within SLEM. Every parent in turn produces a single offspring since I am using non-overlapping generations and deterministic parent selection to suppress other sources of selection pressure. When a parent is selected it is first cloned to produce a child. The machine learner may have learned more than one rule, or some rules may have been kept for multiple generations, so when creating offspring we must select a rule from this set of rules. Once a rule is selected it is used to perturb one or more of the child's genes.

At first glance, it may seem like a variant of LEM's *duoLEM* in that there is no separate evolution mode — all the offspring are generated from the model. And like the original LEM the available strategies for choosing the best and least fit individuals for learning are the same; that is, by rank or percentage of overall fitness range. However, the critical difference is that the offspring are generated from the model *and* optionally from legacy EA perturbation operators such as mutation and crossover. That is, it has been observed that the descriptions, in the form of learned rules, rarely cite all genes; indeed, once a population becomes sufficiently homogeneous because it has converged near an optima, there may be no difference between the best and worst training sets such that the machine learner cannot learn anything. Consequently it falls to legacy mutation and crossover operators to add novelty to those "free variables" not addressed by the learned model.

SLEM is different from LEM in a number of other ways. First, the machine learner should be a type of generative machine learner, such as AQ [Kaufman et al., 1995,Kaufman

and Michalski, 1999b] or C4.5 [Quinlan, 1993], whereas the original LEM specification stated most any machine learner could be used. Admittedly a discriminative machine learner such as a support vector machine or KNN can be coerced into a generative role by being plugged into a generate-and-test framework, but that means that SLEM then becomes more of a memetic algorithm [Mascato, 1989, Krasnogor and Smith, 2005], which is beyond the scope of this work. SLEM's representation is similar to an evolutionary strategy in that the individuals are a fixed length real-valued vector, whereas the original LEM specification accommodated all manner of representations including heterogeneous representations that may have real, integer, or nominal value types.

3.3 Test suite

Each evolutionary algorithm operator has a unique perspective on a given fitness landscape; e.g., mutation and a crossover operators will navigate a problem space differently. Moreover, their unique possible trajectories through the problem space is dependent on the landscape [Jones, 1995].

Using a ML to create offspring is, in itself, a kind of EA operator, and has the interesting property of combining aspects of mutation, crossover, and selection. That is, it emulates a mutation operator by perturbing gene values; parent genes are, in a way, blended when creating offspring gene values as with crossover, albeit through a layer of indirection via the machine learner; and there is a form of selection in that parents are chosen as positive and negative examples for the ML.

I am interested in learning how this unusual EA operator behaves, particularly within the context of my research objectives. To accomplish this I exercised SLEM, which is an implementation of this operator, on a variety of test problems.

Test problems for evolutionary algorithms are nothing new. De Jong presented five test functions, many of which are still in use today [De Jong, 1975]; Schwefel also shared 62 different test problems [Schwefel, 1995]; Salomon presented a dozen test problems [Salomon, 1996]; and Bäck and Michalewicz also enumerated a series of test functions [Bäck et al.,

1997].

The choice of functions in each of those test sets was not arbitrary. Each test function was intended to exercise an algorithm on some salient aspect of a given fitness landscape; and, as a whole, a test suite was intended to fairly represent the types of real-world problems one might encounter. To that end, Schwefel considered two important aspects of a test function: *efficiency* and *effectiveness* [Schwefel, 1995]. Efficiency denoted how aggressively an algorithm converged; effectiveness described an algorithm's reliability. Whitley also outlined a number of guidelines for test suites. These guidelines state that test suites should include functions resistant to hill climbing, they should have problems that are nonlinear and non separable, include scalable functions, and test functions should adhere to a canonical form [Whitley et al., 1995].

It is with these in mind that Bäck and Michalewicz delineated the following criteria for creating suitable test suites in the *Handbook of Evolutionary Computation* [Bäck et al., 1997]:

- unimodal problems

  Even though unimodal problems are typically easy for hill climbers to solve, they are nonetheless useful for evaluating a given algorithm's efficiency from its convergence rates.

- multi modal problems

  The preference is for multi modal problems with a large number of optima as they better represent real-world problems and provide measures for a given algorithm's effectiveness. However, Bäck and Michalewicz provide a number of caveats with regards to multi modal problems.

  First, they caution that some algorithms may be able to take advantage of problems with regularly spaced optima. Second, some problems may counter intuitively become easier with an increase of dimensionality; for example, the Griewangk function becomes smoother as the number of dimensions increases [Whitley et al., 1995]. Third,

binary representations may have an implicit advantageous bias with test problems with global optima at the origin. And, finally, they warn against using linearly separable problems as they are neither representative of real-world problems nor particularly challenging.

Linearly separable problems are those of the form $f(\vec{x}) = \sum_{i=1}^{n} f_i(x_i)$. That is, each dimension can be solved independently of the others. However, such problems can be made non separable by effecting a rotation [Salomon, 1996]. Bäck and Michalewicz suggest that if one is going to include a linearly separable problem that a corresponding rotated version also be included [Bäck et al., 1997].

- problems incorporating noise

  Many real world problems incorporate some stochasticity so it is important to determine an algorithm's robustness to noise. Though Gaussian noise is the most common probability distribution, others distributions might be considered.

- constrained problems

  It is also the case that many real-world problems of interest have inviolable constraints dictated by physical limitations, and so it follows that algorithms should be exercised with various active constraints. Bäck and Michalewicz recommend that a number of linear and nonlinear constraints should be employed. They also counsel that some optima be near constraint boundaries and not comfortably within viable regions. They also suggest that a number of different ratios between search space sizes and the global search space be exercised; they point out that small viable regions are more challenging than regions that encompass nearly the entire global space [Bäck et al., 1997].

- high dimensionality problems

  As many real-world problems have a large number of dimensions, so should test functions. Indeed, one aspect of a test function's usefulness is its scalability with regards to the number of dimensions.

I took the aforementioned advice seriously when deciding on a suitable set of test functions. For one, I ensured that I had a unimodal function to compare convergence efficiency. For another, most of the test functions I selected are multi modal with a large number of optima; though two are linearly separable, I provide rotated counterparts for comparison. And, though most of the multi modal functions have regularly spaced minima, in one these are irregularly placed.

However, I had to limit the scope of test functions to a set that was realistically attainable in this work — so I do not include test functions with noise nor constraints. I also do not provide multiple instances of scalable problems of differing dimensions. Certainly these types of problems should be explored for this type of EA/ML hybrid in future work.

Next, I describe the test functions used in this work.

3.3.1 Spheroid

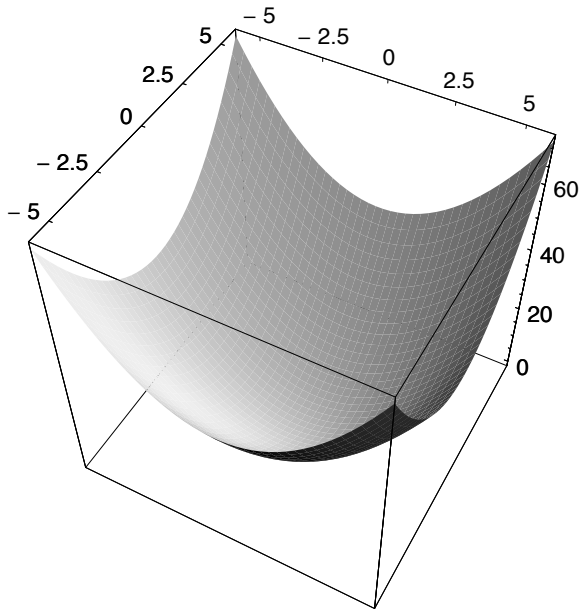


Figure 3.1: Spheroid function (Image courtesy of Sean Luke (sean@gmu.edu).)

The spheroid function, as shown in Eq. 3.1, is linearly separable with a single optimum at the origin and is at the bottom of a steep gradient. This is a common test function for determining convergence velocity [Bäck et al., 1997].

$$f_{sphe}(\vec{x}) = \sum_{i=1}^{n} x_i^2 \tag{3.1}$$

3.3.2 Step

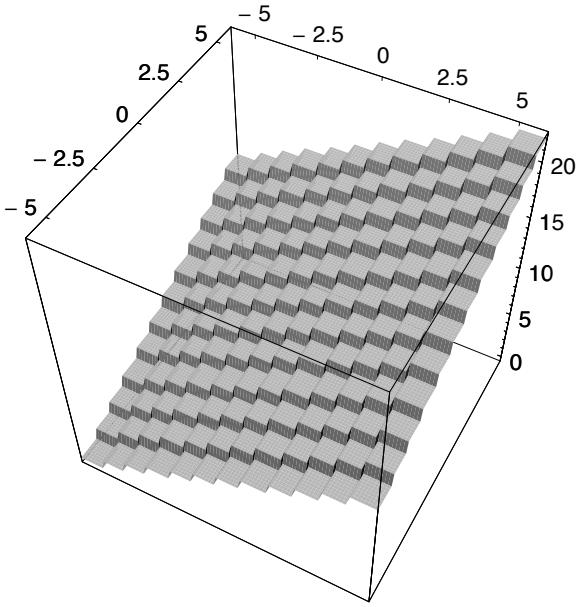


Figure 3.2: Step function (Image courtesy of Sean Luke (sean@gmu.edu).)

$$f_{step}(\vec{x}) = \sum_{i=1}^{n} \lfloor x_i \rfloor \tag{3.2}$$

Eq. 3.2 is commonly referred to as $f2$ in the literature and was another function in De Jong. It is a simple step function that is comprised of a series of descending plateaus

on which populations can be trapped. Though originally constrained to [-5.12,5.12] for a genetic algorithm [De Jong, 1975] I am treating this as an open ended problem; it is possible that SLEM will have an implicit constraint within the original sample boundaries. That is, given that the sole source of novelty is from rules learned from the initial population — i.e., there is no mutation operator to nudge the initial population outside those bounds – then new gene values will remain within the range. This is a potential problem given that we do not necessarily know where the global optimum is, if any, for a real world problem, and so initial gene value ranges may be a guess so that the optimum is outside those ranges. In which case, it would be import for the EA to explore outside the initial population boundaries.

3.3.3 Rastrigin

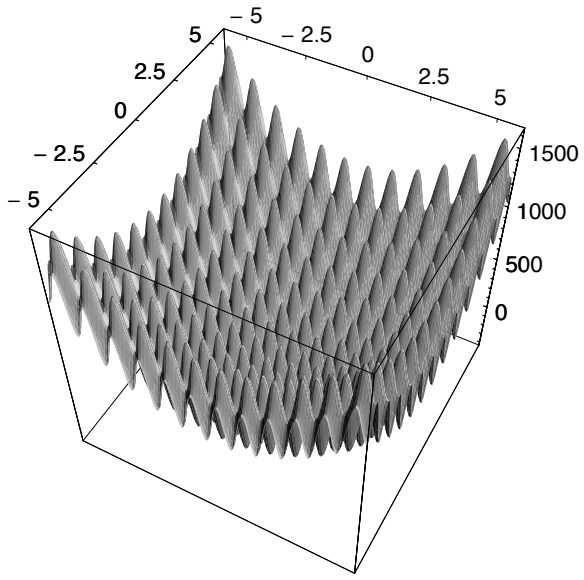


Figure 3.3: Rastrigin function (Image courtesy of Sean Luke (sean@gmu.edu).)

$$f_{rast}(\vec{x}) = 10n + \sum_{i=1}^{n} \left(x_i^2 - 10\cos\left(2\pi x_i\right)\right) \tag{3.3}$$

The Rastrigin function, as given in Eq. 3.3, is a linearly separate multi modal function, and is another function from De Jong that is commonly referred to as f_6. It is comprised of a regular grid of evenly spaced minima aligned along the axes within a quadratic bowl.

3.3.4 Rotated Rastrigin

$$f_{rot\_rast}(\vec{x}) = f_6(\vec{x} * M) \tag{3.4}$$

Some EAs have greater traction for certain fitness landscapes that have an inherent bias along their axes; this bias can be foiled by doing a rotation of the fitness function [Salomon, 1996]. Eq. 3.4 is Eq. 3.3 with just such a rotation applied. M is a rotation matrix of $D \times D$, where D is the problem dimension.

3.3.5 Griewangk

$$f_{grie}(\vec{x}) = 1 + \sum_{i=1}^{n} \frac{x_i^2}{4000} - \prod_{i=1}^{n} \cos\left(\frac{x_i}{\sqrt{i}}\right) \tag{3.5}$$

The Griewangk function, Eq. 3.5, is a non-separable, multi modal function. As with the Rastrigin, the minima are regularly spaced and aligned with the axes.

One danger with the Griewangk is that, unlike with most other test bed functions, the complexity decreases with the number of dimensions [Whitley et al., 1995].

3.3.6 Rotated Griewangk

$$f_{rot\_grie}(\vec{x}) = f(\vec{x} * M) \tag{3.6}$$

Just as with the Rastrigin function, certain EAs can take advantage of Griewangk's alignment along the axes. And, as with the Rastrigin, rotating the Griewangk can eliminate

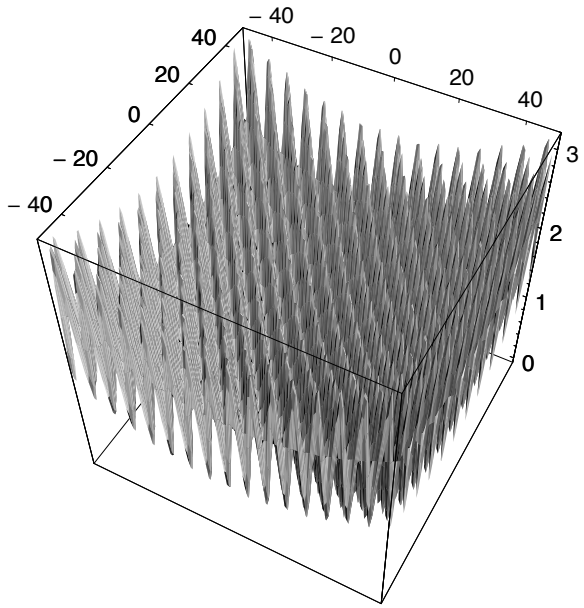


Figure 3.4: Griewangk function (Image courtesy of Sean Luke (sean@gmu.edu).)

this advantage thus making this test function more challenging. Again, M is a $D \times D$ orthogonal rotation matrix.

3.3.7 Langerman

$$f_{lang}(x) = -\sum_{i=1}^{n} c_i \left(e^{-\frac{1}{\pi}\|x-A(i)\|^2} cos\left(\pi \|x - A(i)\|^2\right)\right) \tag{3.7}$$

The Langerman function, as shown in Eq. 3.7, is multi modal, not linearly separable, and has irregularly spaced optima [Bersini et al., 1996].

3.4 Set-up for Experiments

Table 3.1 gives the common run-time parameters used for the experiments, which I describe in more detail in this section. However, other chapters override certain parameters, in which

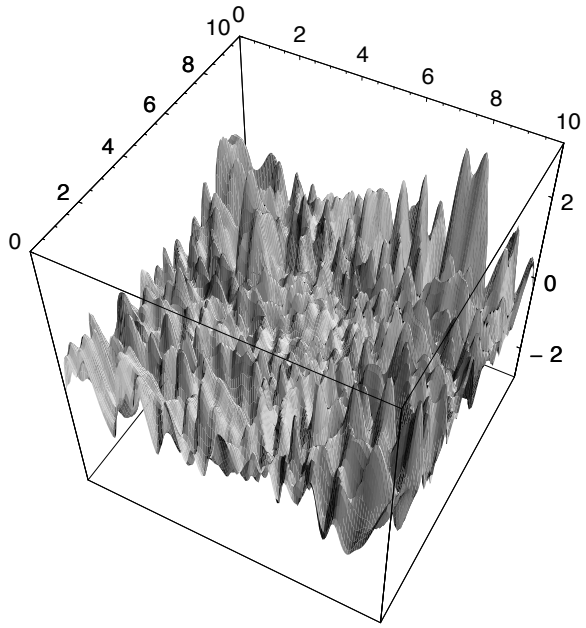


Figure 3.5: Langerman function (Image courtesy of Sean Luke (sean@gmu.edu).)

Table 3.1: Default run-time parameters for experiments.

Parameter	Value
Parents	90
Brood size	1
Best training set size	top 30
Worse training set size	bottom 30
Generation model	non-overlapping
Genome size	5
Mutation rate (ρ)	none
Runs	30
Machine learner	PART [Frank and Witten, 1998]
Random rotation seed	9731297

case those exceptions are shared at that time.

To better study the source of selection pressure due to the ML, I need to minimize other sources of selection pressure. There are two ways to accomplish this. First, I use non-overlapping generations to eliminate the source of selection bias via survivor selection. If the parents are discarded every generation regardless of their respective fitnesses so that the children become the next generation's parents regardless of their quality, that removes one source of selection pressure since there is no survivor selection biased by fitness. Second, the other avenue of selection pressure is via parent selection. If one uses deterministic selection, in this case each parent produces exactly one offspring — that is, has a brood size of one — then there is no fitness bias with regards to parent selection. Therefore, any observed convergence is due to selection pressure generated by the use of the ML to create offspring as had been previously discovered [Coletti, 2012].

The effects from legacy EA related sources of genetic novelty, such as through mutation and crossover operators, need to be eliminated since their influence would obscure source of selection due to the ML. Certainly these do play an important role in such a system, but studying those roles is left to future work. However, I will occasionally explore the effects of mutation and crossover where warranted.

The parent population sizes for the parents and for the best and worst training sets are inspired from prior LEM related research. In particular, most of the LEM work used fixed training set sizes with middle gap of "mediocre" individuals that fell in neither the best nor worst training sets [Wallin and Ryan, 2009]. However, other configurations that evenly split the population between best and worst, as well as having the training sets come from the middle component of the population, are further explored in Ch. 5; but where the training set sizes are not being studied the fraction of $\frac{1}{3}$ will be used for training set sizes. I.e., the top one third of parents will be selected by rank as the "best" training set, and the bottom one third the "worst". The middle third is deemed "mediocre" and not explicitly used for training.

Theory provides some guidance for population sizes for machine learning. In particular,

Probably Approximately Correct (PAC) learning theory can give us a minimum number of individuals to pass to the machine learner as training sets; indeed, this approach has been applied to a similar EA/ML hybrid, SI3E, to come up with training set population sizes [Llora and Goldberg, 2003].

$$\max\left[\frac{1}{\epsilon}\log\left(\frac{1}{\delta}\right), \frac{VC(C)-1}{32\epsilon}\right] \tag{3.8}$$

Eq. 3.8 estimates the lower bound for the number of individuals suitable for training a ML; it states that with likelihood $1-\delta$ that the concept will be learned with error less than or equal to ϵ [Mitchell, 1997]. VC is a measure of the Vapnik–Chervonenkis dimension for, C, the concept class. In this case the concept class includes concepts for positive and negative examples, or "best" and "worst" individuals, so would be 2. So, the lower bound for the number of training individuals necessary with a 95% chance to learn the target concept with an error of 5% or less is $\lceil 59.91465 \rceil$, or 60 individuals. Therefore I use a parent population size of 90 since the top and bottom third population portions sum to 60.

Also, given that each parent produces exactly one offspring — i.e., has a brood size of one — then 90 children are produced each generation. And, again, I use non-overlapping generations to eliminate selection pressure due to survival selection, so the set of 90 offspring become the parents for the next generation.

Tbl. 3.2 gives the run-time parameters for the test functions described in §3.3 and is derived from a similar table found in [Ortiz-Boyer et al., 2005]. This table provides the domains, global optimal values, and other features of each test function I use.

Note that the domain for some test functions differs from what is found conventionally within the literature. E.g., the domain for the spheroid is typically in $[-5.12, 5.12]$ whereas I use $[-10, 10]$. Though this violates Whitley's suggestion of using canonical settings for test functions [Whitley et al., 1995], I decided to stray from the conventional idiom for a number of reasons. First, I am not comparing performance of SLEM with other algorithms; all comparisons made in this work are between various runs involving SLEM. Second, the

Table 3.2: Test function definitions and parameters

Function	Definition	Multi modal	Separable	Regular
Spheroid	$f_{sphe}(\vec{x}) = \sum_{i=1}^{n} x_i^2$ $x_i \in [-10, 10]; n = 5$ $x^* = (0, 0, ...0); f_{sphe}(x^*) = 0$	No	Yes	n/a
Step	$f_{step}(\vec{x}) = \sum_{i=1}^{n} \lfloor x_i \rfloor$ $x_i \in [-30, 30]; n = 7$ $x^* = (-\infty, -\infty, ... - \infty); f_{step}(x^*) = -\infty$	Yes	Yes	n/a
Rastrigin	$f_{rast}(\vec{x}) = 10n + \sum_{i=1}^{n} \left(x_i^2 - 10\cos(2\pi x_i)\right)$ $x_i \in [-10, 10]; n = 5$ $x^* = (0, 0, ...0); f_{rast}(x^*) = 0$	Yes	Yes	n/a
Rotated Rastrigin	$f_{rot\_rast}(\vec{x}) = f_{rast}(xM)$ $x_i \in [-10, 10]; n = 5$ $x^* = (0, 0, ...0); f_{rot\_rast}(x^*) = 0$	Yes	No	n/a
Griewangk	$f_{grie}(\vec{x}) = 1 + \sum_{i=1}^{n} \frac{x_i^2}{4000} - \prod_{i=1}^{n} \cos\left(\frac{x_i}{\sqrt{i}}\right)$ $x_i \in [-600, 600]; n = 5$ $x^* = (0, 0, ...0); f_{grie}(x^*) = 0$	Yes	No	Yes
Rotated Griewangk	$f_{rot\_grie}(\vec{x}) = f_{grie}(xM)$ $x_i \in [-600, 600]; n = 5$ $x^* = (0, 0, ...0); f_{rot\_grie}(x^*) = 0$	Yes	No	No
Langerman	$f_{lang}(x) = -\sum_{i=1}^{n} c_i \left(e^{-\frac{1}{\pi}\|x - A(i)\|^2} \cos\left(\pi\|x - A(i)\|^2\right)\right)$ $x_i \in [0, 10]; n = 5$	Yes	No	No

original domains were for genetic algorithm (GA) implementations that required a mapping from a binary string implementation to real numbers, hence the power of two seen in those domains. I am not using a GA to implement SLEM; instead I am using native real value representations for gene values, so the original power-of-two-based domains are not relevant. Third, I wanted to limit the number of genes, or the dimensionality of the problems, to at most seven genes since that would make visualizations of their values clearer; adding dimensionality would make visualizations of individuals more challenging. However, this may mean problems become too easy at under eight dimensions, in which case I would have to enlarge the domains as needed. I selected the domains based on informal sensitivity tests.

The machine learner I use, PART, is provided by the Weka machine learning and data mining suite [Hall et al., 2009]. PART is similar to C4.5 in that it builds decision trees, but it builds partial decision trees from which it creates rules [Frank and Witten, 1998]. I chose PART from the Weka suite because it generated rules from which I could easily sample intervals, and it was easy to modify the Weka implementation to work with SLEM. For all experiments I used the default PART run-time settings unless otherwise specified. In any case, the design decision to use this particular machine learner imposes an associated

induction bias; it would be useful to explore the influence of other machine learners as well as different permutations of their associated run-time parameters on SLEM. However, such exploration is beyond the scope of this work.

I also used the ECJ evolutionary algorithm toolkit in my implementation of SLEM [Luke et al., 2013].

3.5 Measuring effects of selection pressure

So far I have described this chapter's research objectives, the algorithm I will be using, and on what test bed I will be using it on. Naturally all of this will generate data. Here I describe how I measure how rule interval sampling, training set size, and rule persistence affect this unique form of selection pressure.

Since I want to measure selection pressure-like effects, naturally I turn to extant and relevant selection pressure measures. Indeed, there have been a number of selection pressure measures derived in which to scrutinize legacy selection operators. However, given the unique nature of the selection pressure created by introducing a machine learner as described in SLEM, only a subset of these measures may be relevant. Moreover some novel measures are introduced to accommodate that uniqueness.

Selection Intensity Selection intensity is a term borrowed from population genetics, and was first used by [Schlierkamp-Voosen, 1993] and has since been adopted by others. Eq. 3.9 shows how selection intensity is calculated [Blickle and Thiele, 1996]. The intuition here is that it is a measure of the difference of the average fitnesses between the parents of the ith generation, $\overline{P}_i$, and their offspring, $\overline{P}_{i+1}$. Dividing this difference by the overall standard deviation yields a unit-less value, which makes this convenient for comparing selection intensities between fitness landscapes with differing domains.

$$I = \frac{\overline{P}_i - \overline{P}_{i+1}}{\overline{\sigma}} \tag{3.9}$$

Time to converge. The time to converge will be inversely proportional to the selection pressure — the higher the selection pressure, the more quickly the population settles on an optima. There are a few common approaches for determining when a given population has converged [De Jong, 2006]. First, we can say that the population has converged if the individuals are spatially grouped within a certain minimum distance of one another. Second, if the fitness value variances have been close to zero for an arbitrary amount of time, then the population can be said to have converged. Third, if the best-so-far individual has not changed in a significant way for so many generations.

I have selected the following approach: if the fitness of the best individual so far in a particular run has not changed more than by 0.15 for five generations, then I deem that the run has converged. This does not necessarily mean that the population is completely homogeneous, nor that the population has found and converged on the global optima. What it does mean is that the run is likely not to significantly further improve. Note that these numbers are not entirely arbitrary. The objective was to find the longest run and the smallest tolerance for which all runs would converge. If I was too aggressive with my convergence criteria, then some runs would never converge. I found that going beyond five runs of a tolerance of 0.15 would start to yield runs that never converged.

Population distribution A population will condense to an optima under the influence of selection pressure. Naturally, this means that the distribution of the population will change over time as it approaches a given optima. Measuring the quantiles of a population by generation will depict the change in fitness distribution over time as the system condenses. Furthermore, population quantiles can be readily visualized by generation using box-and-whisker plots as well as any outliers.

Number of rules learned The machine learner will learn zero or more rules per generation. The number and complexity of the rules can be dependent on a number of factors such as training set sizes and the fitness landscape being explored as well as the particular type of machine learner employed.

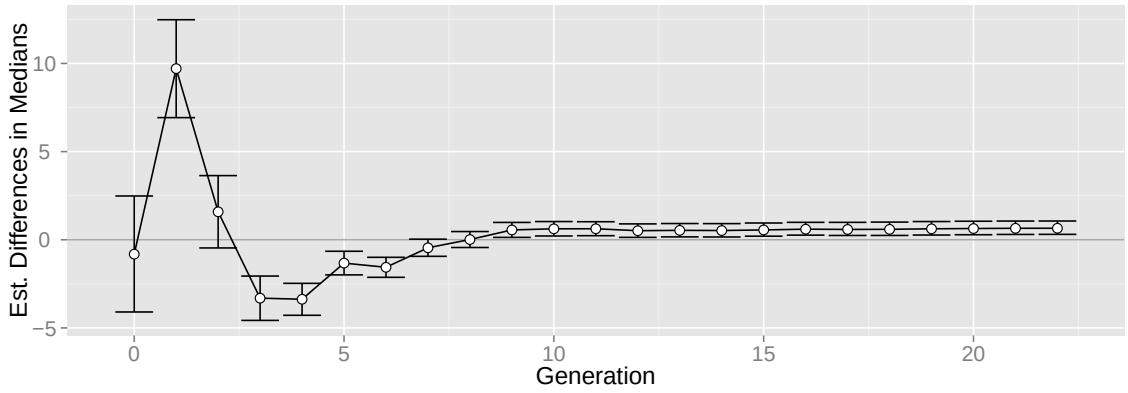


Figure 3.6: This shows the differences between a pair of EA runs by generation. Depicted are 95% confidence intervals for the estimation of differences between medians for the two populations. This type of plot is paired with a plots showing population trajectories of runs as a visual aid to highlight differences between runs that would otherwise be difficult to discern.

Number of genes cited by rules Each rule learned by the machine learner will refer to one or more genes, and is the effective mutation rate. For example, if a given rule refers to a single gene, then the mutation rate is essentially $1/L$ since only one gene, the one denoted in the rule, will be perturbed when creating offspring.

3.6 Data Visualization

For this work I created two data visualization techniques to all the better depict results and serve as analysis aids. The first technique is a means of quantitatively comparing a pair of EA runs. The second is a way of portraying histogram or frequency data on a by generation basis.

3.6.1 Quantitatively comparing EA run pairs

Comparing two different evolutionary algorithm runs is not straightforward. Do we just compare the end of the run results? If we compare convergence times, what arbitrary convergence measure do we use? Do we compare best-so-far curves? Worse yet, part of the difficulty with choosing from this set is that practitioners have different priorities; e.g., one

might be willing to wait for better answers, whereas another might be more interested in quickly converging on solutions even if it means penalizing solution quality.

Moreover, these traditional means of portraying EA runs can miss interesting behavior. That is, probing convergence times or end-of-run statistics observe the runs at single points. Though these are important probe locations, there are a lot of other locations where interesting deviations between runs can occur that would otherwise be missed. To address these short comings, I decided to compare the populations of two given runs by generation. This not only provides meaningful information to the practitioner interested in end-of-run behavior, but also the practitioner interested in convergence-oriented measures, but also serves to portray interesting differences between the runs at other generations.

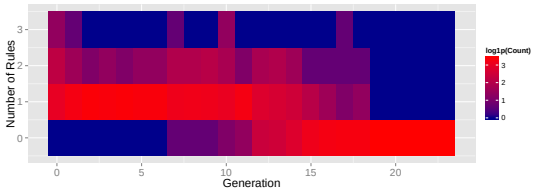
Fig. 3.6 shows an example of a type of plot used throughout this work that shows a comparison of the estimate of the median differences between two sets of runs. For each generation a 95% confidence interval is shown estimating the likelihood of the amount of differences between medians for two populations.

If we had used more traditional measures, we would have looked at the differences between these pairs of runs at the very end, or when they converged at around generation 10. We can see that there is little difference between these two probe points, and that they miss the interesting see-sawing behavior that occurs before convergence.

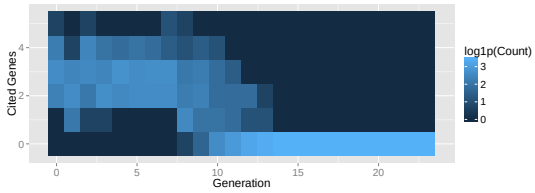
3.6.2 Visualizing by-generation frequency data

Another type of visualization that I use throughout this work involves frequency data by generation. Box-and-whisker plots yielded unsatisfactory results since this entailed visualizing discrete data. What I needed was a better way to depict this kind of data.

Fig. 3.7 provides two examples of the visualization I use for depicting such data. Both sub-figures show frequencies by generation using heat maps — you can think of these heat maps as histograms of frequency data stacked side-by-side with the colors in proportion to the counts of their respective bins. The brighter the color, the higher the frequency for the associated bin. And, since the data tended to be skewed, I log transformed the color scale



(a) This heat map shows the frequency of rules by generation for an entire run. For the start of the runs the ML typically learns one rule, sometimes two, and occasionally three rules. The collective number of rules declines after a dozen generations until the 19th generation by which time the ML learns nothing in all runs.



(b) This heat maps shows the aggregate count of cited genes for rules for a different set of runs. At the beginning of the runs the rules generally referred to two to three genes, but sometimes just one or up to five. When the populations began to converge after several generations, the number of cited genes began to drop until the ML stopped learning anything after 14 generations for all runs.

Figure 3.7: These are two similar examples of showing frequency data by generation for two different sets of runs. Fig. 3.7a shows the frequency of rules over all generations for one set of runs. Fig. 3.7b shows the same kind of information for another set of runs, but instead depicts the sum of counts of the number of genes that rules refer to by generation over the course of the runs. In both cases the color scale is log transformed so that low frequency data is easier to see.

so that the low frequency data could be better seen.

3.7 Summary

In this chapter I gave my research objectives and described the approach by which I attain them. All of these objectives explore means of influencing the emergent selection pressure arising from using a rule-based machine learner to improve an EA's offspring quality.

One of these means examines different strategies for sampling areas of high fitness as identified by the machine learner. Another considers a number of ways of selecting the positive and negative training sets for the machine learner. And, finally, yet another studies various policies for selecting from multiple rules when creating offspring.

I also described the algorithm, SLEM, that is used as a vehicle of exploration for these objectives, and which is applied to a set of well known test functions representative of real world problems.

I related the common experimental design and run-time parameters used in this research with the proviso that some may be over-ridden or modified in subsequent chapters. Lastly, I

described how I measure and analyze the effects the different experimental treatments would have on the emergent selection pressure as well as sharing two different data visualization techniques I use throughout this work.

The next chapters explore each of these research objectives in turn. Ch. 4 looks into rule sampling strategies for offspring creation, and Ch. 5 looks more closely into the influence of various training set strategies.

Chapter 4: Interval Sampling Policy Effects

The focus of this research is to learn more about a unique type of selection pressure that arises when a rule-based machine learner is used to create offspring in an evolutionary algorithm. The machine learner infers rules from the set of the best and worst parents, rules which identify likely areas of higher fitness in which to create offspring. However, one of the first design decisions a practitioner faces when assembling these kinds of systems is how to sample those regions when creating offspring. This chapter explores a number of strategies a practitioner could use to sample rule intervals and their respective influence on this special type of machine learner based selection pressure. These strategies include using uniform, Gaussian, and histogram-based probability distributions to sample regions of higher fitness as identified by the machine learner. Moreover, it is possible for the machine learner to learn a single rule interval bound when both an upper and lower bound are necessary to sample values when creating offspring; this chapter also analyzes a number of approaches for filling in those missing bounds.

4.1 Introduction

As described in §3.1.1 the rules representation I use are conjunctions of $[l, u]_i$ where l is a lower bound, u an upper, and which is applied to the ith gene when creating offspring. As an example the following is output from using the PART machine learner [Frank and Witten, 1998] to learn good gene value ranges from a set of "best" and "worst" parents from a run of the spheroid minimization problem:

```
Gene2 <= 5.608972 AND
Gene2 > -6.128321 AND
```

```
Gene4 > -4.885725 AND
Gene4 <= 3.176545: best (19.0)
```

This rule states what the PART machine learner deems the "best" individual; it specifies that `Gene2` is in $(-6, 128321, 5.608972]$ and `Gene4` in $(-4.885725, 3.176545]$ and that this rule correctly covers 19 training instances. However, for our purposes this corresponds to the SLEM rule $[-6.128321, 5.608972]_2 \wedge [-4.885725, 3.176545]_4$ for describing a "best" individual. That is, children to which this rule is applied will have gene two assigned a value within the interval $[-6.128321, 5.608972]$ and gene four to the interval $[-4.885725, 3.176545]$.

Note that for this particular run that the genome size was five, which means that this rule says nothing about viable values for genes 0, 1, and 3; in which case all offspring to which this rule is applied will inherit copies of their parents' respective values for those genes. It is possible for a rule to cover all genes, in which case offspring using that rule to generate new values will inherit nothing from their parents; in practice this rarely, if ever, happens. However, as we shall see later, it is possible for the machine learner to learn nothing, in which case there will be no rules; in that scenario all the children are clones of the parents and will remain so until the end of a given run.

Line 8 of Alg. 4, which is found on page 26, is the focus of this chapter. This line states that for each rule interval associated with a specific gene that a new value is assigned to the corresponding child's gene from some distribution, D, within the interval described by the rule. The choice of D can influence the apparent selection pressure. Explored here are using a uniform, Gaussian, and a histogram-based distribution to sample rule intervals when creating offspring. I will next describe how these distributions are used to sample rule intervals.

4.1.1 Sampling rule intervals with a uniform distribution

The simplest approach to exploiting a rule to create offspring would be to generate gene values cited by the rules using a uniform distribution. For example, the rule $[-1.08, 2.92]_1 \wedge [5.92, 6.13]_3$ would mean that applying that rule would create offspring that would have their

first gene values generated in U(-1.08,2.92) and the third gene in U(5.92,6.13). This is the typical LEM implementation as exemplified in [Cervone, 1999,Coletti et al., 1999,Wojtusiak, 2008, Coletti, 2002, Cervone et al., 2002].

4.1.2 Sampling rule intervals with a Gaussian distribution

The uniform distribution will more-or-less evenly cover the full range dictated by the rule bounds. Naturally, there are other probability density distributions that we could try. Why would we look at these other distributions? It is likely a rare thing that the portion of the fitness landscape covered by the rule interval is flat; it is more likely to have some sort of bumpiness. Ideally we would like our sampling to match the problem terrain in a style akin to an Estimation of Distribution Algorithm (EDA); and, indeed something similar to that approach is described shortly in §4.1.3.

First let us consider a simple Gaussian distribution with a mean centered over the middle of a given rule interval and with varying standard deviations straddling the interval width. The naïve assumption is that regions of higher fitness will tend to be in the middle of rule intervals, which is not necessarily true. However, of other possible interest is that the Gaussian distribution will sample a little beyond the intervals. Note that the rule interval boundaries are not gospel — it could very well be that areas of interest lie just beyond rule boundaries. Indeed sampling error within given rule boundaries could further contribute to the emergent selection pressure of these systems, something discussed more detail in §7.2.8.

In a sense, this approach is similar to an Evolutionary Strategy (ES) in that Gaussian mutation is applied to each gene with a standard deviation that is modulated over the course of a run [Bäck et al., 1997]; in this case, though, a machine learner adjusts the standard deviation instead of traditional ES self-adaptive mechanisms.

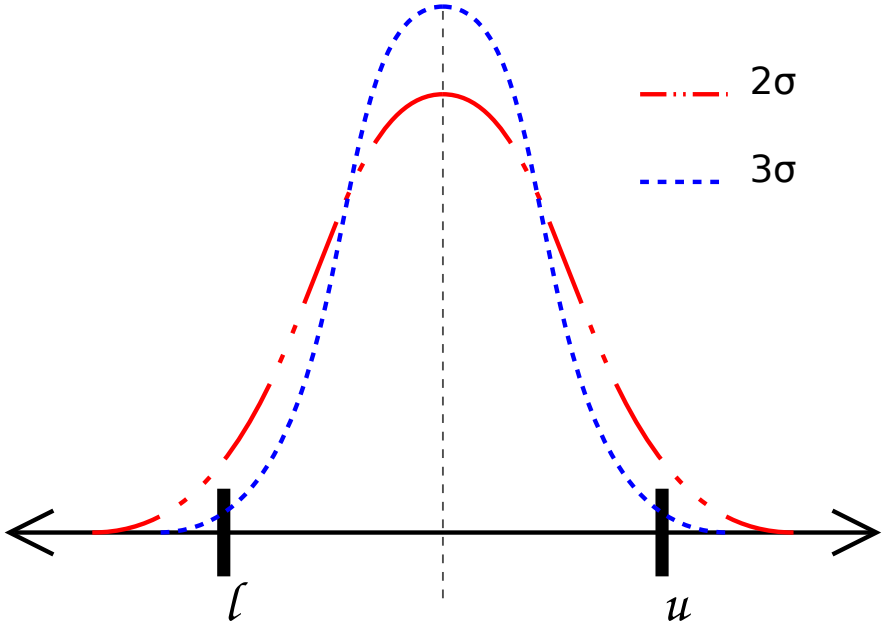


Figure 4.1: This depicts a rule interval, $[l, u]$, sampled using Gaussian distributions of two and three standard deviations.

$$h = \frac{(u - l)}{2} \tag{4.1}$$

$$m = u - h \tag{4.2}$$

$$v = m + \mathsf{N}(m, 1) \times \frac{h}{\sigma} \tag{4.3}$$

Fig. 4.1 shows how I implemented SLEM using a Gaussian distribution to sample within rule intervals. I use two different Gaussian distributions, one using two standard deviations, and the other three, of which both are centered on the mid point of a given rule interval. To set a new gene value based on a Gaussian distribution I first take half the interval size, Eq. 4.1, use that to compute the rule interval midpoint, Eq. 4.2, and then calculate the new

value based on sample from a Gaussian distribution centered round that midpoint with the appropriate σ, Eq. 4.3.

4.1.3 Sampling rule intervals with a histogram-based distribution

So far, the distributions from which offspring gene values have been drawn have been naïve in the sense that they are topology agnostic – offspring gene values are sampled from probability distributions within rule intervals that are derived from an equation and not from information about the fitness landscape. Might we see a difference in the emergent selection pressure were we to take into consideration information learned about the fitness landscape? That is, try a different approach that would be similar to an Estimation of Distribution Algorithm (EDA), which is a type of evolutionary algorithm that does just that. There are myriad EDA implementations, but the common factor among them is computing a statistical model – a distribution – from a population, and then using that distribution to generate offspring [Baluja, 1994, Mühlenbein and Voigt, 1995]. Indeed, something like this has been tried with the types of EA/ML hybrids explored in this work. That is, for a given interval, a histogram is computed from the positive examples from which a probability distribution is computed, and from which offspring gene values are drawn [Cervone and Franzese, 2011].

However, one problem with EDA implementations is that gene value frequencies that drop to zero have no chance to be reproduced for the rest of a run since their associated probabilities become zero. This may not be a problem if the lower frequency is due to poor associated fitness; but given the inherent stochastic nature of evolutionary algorithms, including EDAs, it may be just by happenstance that the frequency is zero regardless of the actual associated fitness. In which case, it would be impossible to explore that region of the fitness landscape unless other measures are taken such as introducing mutation or just restarting the run.

Cervone, et al, took a different approach to get around this problem. For the corresponding learned rule intervals a histogram would be computed, but a second parameter,

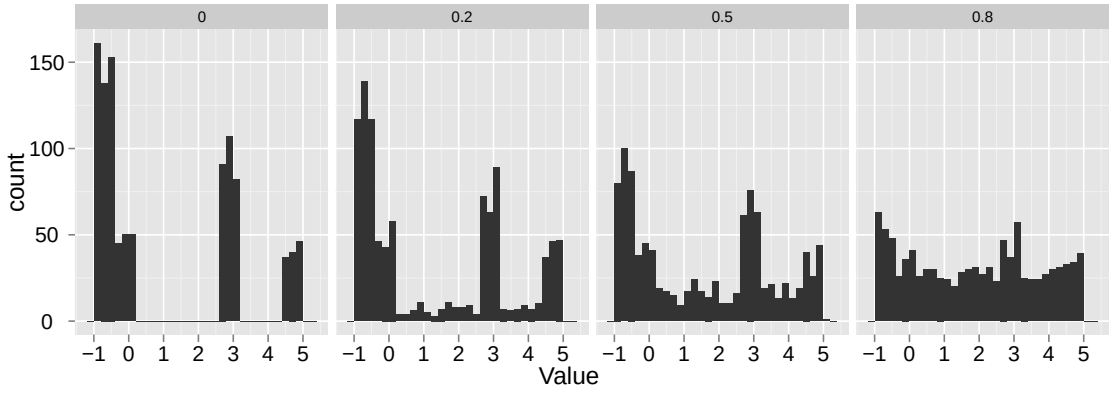


Figure 4.2: The effects of setting the budget for histogram-based distribution for uniform distribution probability budgets, α, of 0.0, .2, .5, and .8. The histogram values were randomly generated from an arbitrary set of values for illustrative purposes with three peaks in the range [-1,5].

which I refer to as α, would dictate how much of the histogram bin budget would be driven by actual gene values vs. a uniform distribution. Fig. 4.2 shows the effects of varying this parameter for a sample histogram for an example where some simple values were randomly generated with three peaks within the range [-1,5]. The first subfigure shows the distribution influenced solely by actual gene values. The remaining subfigures gradually split the influence of the histogram vs a uniform distribution; the second subfigure shows 20% going to the uniform distribution, then the next subfigure 50%, and then, finally, 80%.

Note that the number of histogram bins is important. Too many bins would mean too sparsely populated histograms; too few would mean too coarse grained a sampling. Indeed, the bin size of one would be identical to a uniform distribution. I would need to settle on a reasonable number of bins before proceeding with further exploration. Therefore, I performed sensitivity tests to find bin sizes that would have the right level of granularity. I started with a bin size of 10, which produced sparsely populated bins; I annealed the bin sizes until all the bins were usually getting at least one gene value. For the spheroid fitness landscape and for the population size of 90 parents and 90 offspring I was using, I determined that this bin size was five.

In the next section I cover a problem whereby the machine learner only learns a single

rule bound, and three different approaches for assigning a value for that unspecified bound.

4.2 Strategies for Closing Unbounded Rule Intervals

Some machine learners will infer rules that have unbounded intervals describing areas of higher fitness. That is, a given rule may have only an upper or lower bound, and not both, which poses a problem if one wishes to sample in some way a given interval when creating offspring.

This section discusses the influence that three strategies for closing unbounded rule intervals have on the emergent selection pressure derived from the machine learner — one strategy that clamps them to initialization boundaries, another to parent population gene value extrema, and another to the most extreme gene values of the "best" parents.

4.2.1 Introduction

One problem that practitioners may encounter are cases where the machine learner only infers one rule-interval bound. As an example, the following output is from using the PART machine learner [Frank and Witten, 1998] to learn good gene value ranges from a run of the spheroid minimization problem:

```
Gene3 > -5.132166 AND
Gene3 <= 6.423435 AND
Gene0 <= 8.38355: best (25.0/2.0)
```

This corresponds to the rule $[?, 8.38355]_0 \wedge [-5.132166, 6.423435]_3$ for describing a "best" individual. The second part of the rule is easy to implement — all offspring to which this rule is applied will have their fourth gene values created in [-5.132166,6.423435]. However, implementing the first part is problematic since PART only learned an upper bound. That is, it thinks that the first gene should be less than 8.38355, but has no idea what the values for that gene should be greater than. Since we need two bounds for generating a value in

a uniform or Gaussian distribution we need to come up with a way to supply the other bound.

This kind of behavior where just a single rule interval bounds has been learned has been previously observed in LEM implementations that use C4.5 as the machine learner [Coletti, 2002, Coletti, 2009, Coletti, 2012]. Like PART, C4.5 is a decision-tree-based machine learner [Quinlan, 1993], which suggests that this might be a phenomena associated with those types of machine learners. LEM has also been implemented using the machine learner AQ [Kaufman et al., 1995, Kaufman and Michalski, 1999b], but those implementations do not experience this problem [Coletti et al., 1999, Cervone et al., 2000a, Cervone et al., 2002, Wojtusiak, 2004].

The aforementioned C4.5-based LEM implementation had lower and upper bounds associated with each gene for all individuals. When a population was initialized these bounds were set for all individuals to startup values chosen by the practitioner. During a run offspring were created similarly to what was described for SLEM in §3.2 on page 24. That is, each parent was deterministically selected, the parent cloned to create an offspring, a rule from a rule collection chosen, and then that rule applied to the newly created offspring to generate new gene values. However, here is where that implementation differs from the one used in this work. That is, in the LEM/C4.5 implementation the intervals associated with the individual stored with each gene were updated from the chosen rule, and then the offspring got new gene values based on each gene's intervals using a uniform or Gaussian distribution. In a sense, gene intervals were inherited from the parents, then a subset of which updated from a selected rule, and then new gene values were generated within associated intervals with a uniform or Gaussian distribution. By contrast, in this work I do not save interval bounds in genes. Instead, via mechanisms which I describe shortly, I repair any unbounded intervals, and then sample them with a given probability distribution.

I decided not to emulate LEM/C4.5's behavior here because keeping and updating by-gene interval information implemented a form of persistence, which was likely to lower selection pressure. That is, I knew from prior work that keeping rules for more than one

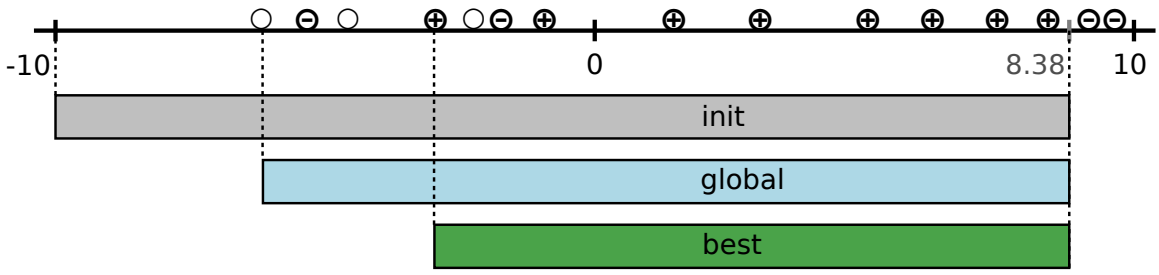


Figure 4.3: Example showing three different approaches — *init*, *global*, and *best*— for resolving the unbounded rule interval $[?, 8.38355]_0$ for a set of gene values corresponding to gene 0. The corresponding repaired rule intervals would be $[-\mathbf{10}, 8.38355]_0$, $[-\mathbf{6.4}, 8.38355]_0$, and $[-\mathbf{3.0}, 8.38355]_0$.

generation had that effect [Coletti, 2012], and so it was likely that this form of persistence would behave similarly. However, the influence on selection pressure from this form of persistence does merit further exploration and so is further discussed in Ch. 7.

So how do I resolve this problem with unbounded rule intervals? Fig. 4.3 shows three different approaches I used for repairing intervals that were missing a bound. In that figure is a number line corresponding to current population values for gene 0 and presuming that the initial population was sampled within $[-10, 10]$. One approach to resolve this problem is simply to use the gene value bounds that the initial population was initialized with; for this example the initial gene values were stochastically generated in U(-10,10), so we can fall back on those bounds for all rules that only specify one bounds. So we could substitute -10 in the unknown lower bounds for the first rule, which now becomes $[-\mathbf{10}, 8.38355]_0 \wedge [-5.132166, 6.423435]_3$. Similarly 10 would be substituted for any missing rule interval upper bounds. In the rest of the work I will refer to this approach for closing unbounded rule intervals as the *init* strategy, and results as applied to the figure are shown as the grey bar.

I also consider two other strategies that close unbounded rule intervals, strategies which are based on the parent population. For the *global* strategy I choose the extreme-most gene value from all the parents of the current generation, which is shown as the light blue bar. There, regardless of whether the most extreme gene value corresponds to the best, worst, or mediocre individuals, that is the value substituted for the missing rule bound. So with that

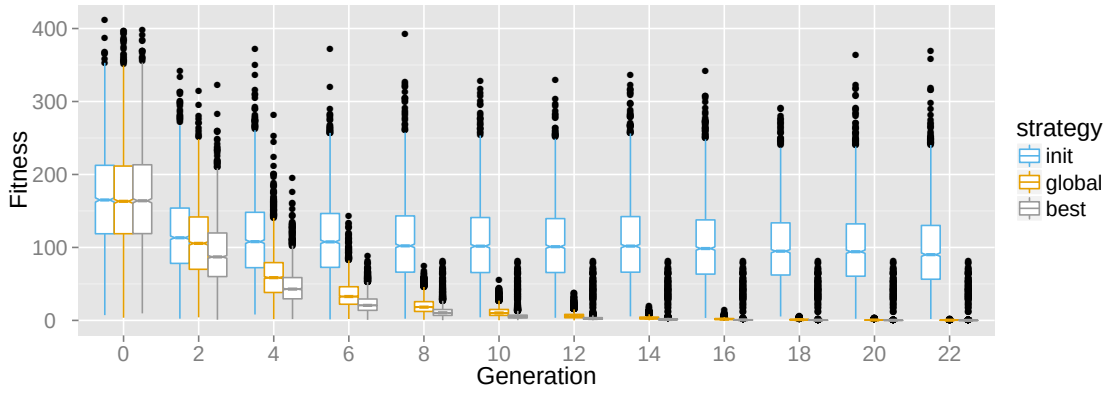


Figure 4.4: Population trajectories for f_{sphe} for three different unbounded rule interval clamping strategies using a uniform distribution to sample rule intervals. The *init* strategy converges within a couple generations comparatively far from the optima; by contrast the *global* and *best* strategies have similar population trajectories with the *best* strategy being slightly more aggressive.

approach the repaired rule would be $[-\mathbf{6.4}, 8.38355]_0 \wedge [-5.132166, 6.423435]_3$. For the *best* strategy I choose the extreme-most gene value from parents in the machine learner "best" training set, which are shown as the green bar in the example. For the *best*, the repaired rule would be $[-\mathbf{3.0}, 8.38355]_0 \wedge [-5.132166, 6.423435]_3$

What kind of behavior associated with each strategy might we see? For one thing, the *best* is likely to have higher selection pressure than *global* since the former approach only considers the extrema of the "best" parents, and hence is greedier, than the latter which uses the entire population. The *init* should manifest the least selection pressure of the three strategies since it should have the effect of resetting one or more rule interval bounds to the original bounds used to create the initial parent population. As we shall soon see, the next two sections provide evidence to support these hypotheses.

4.2.2 Using *init* strategy is disruptive

Fig. 4.4 shows the population trajectories for the three sets of runs for the spheroid test function using a uniform distribution to sample rule intervals. What stands out is that the *init* runs converge comparatively quickly to the *global* and *best* runs, but yet fail to eventually converge near the global optimum. Given that this is the spheroid function we

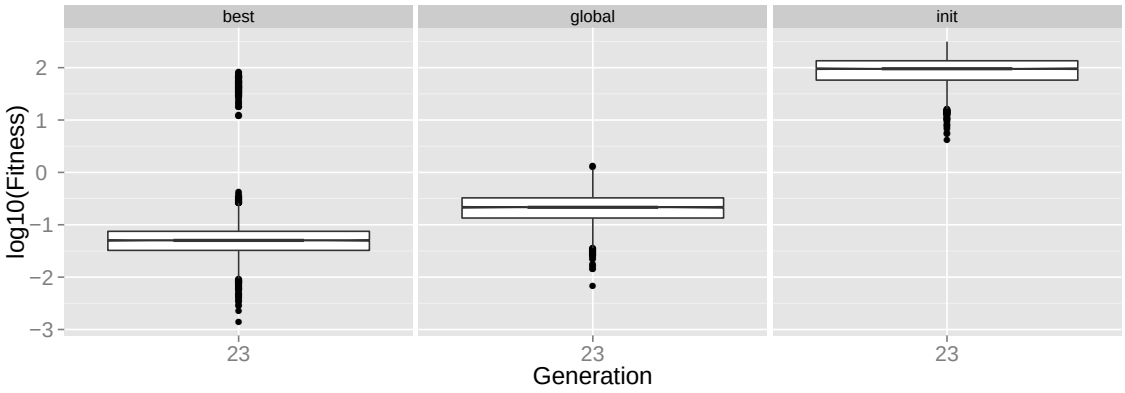


Figure 4.5: This compares the log transformed fitness distributions for the last generation between the *best*, *global*, and *init* unbounded rule interval clamping strategies for the f_{sphe} using a uniform distribution to sample rule intervals. The *best* is much closer to the global optimum, followed by *global*, and then *init*.

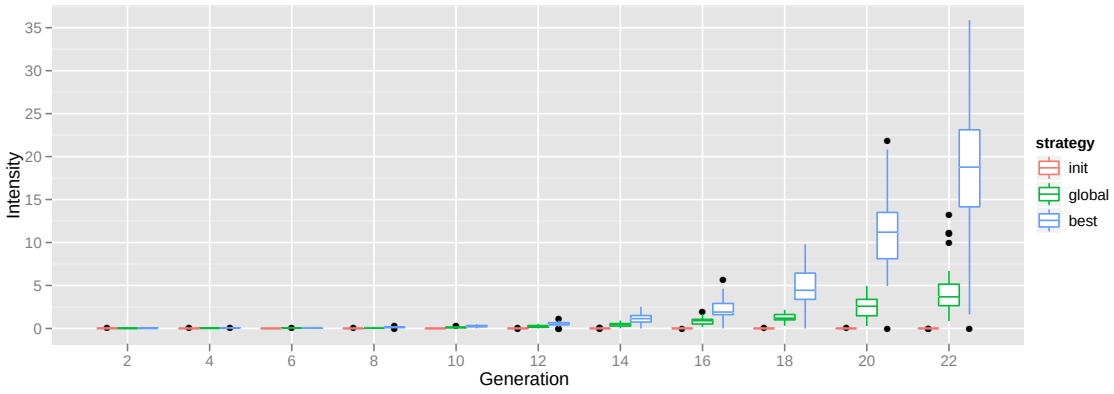


Figure 4.6: Selection intensities for f_{sphe} for three different unbounded rule interval clamping strategies using a uniform distribution to sample rule intervals. The *init* strategy selection intensities are nearly nonexistent whereas the intensities for *global* and *best* strategies monotonically increase. The *best* strategy has the highest selection intensity, particularly at the end of runs.

know that the global optimum is at the origin so the best fitnesses will be at or near zero; it appears that the center of mass only gets so far before getting stuck without condensing the rest of the way to the origin. Fig. 4.5 shows the population distributions for the last generation for all three sets of runs; there we can see that the *best* has converged most closely to the global optimum, followed by the *global*, trailed by the *init* unbounded rule interval strategies. The corresponding Bonferonni adjusted Wilcoxon rank sum tests have p

Table 4.1: This shows a summary of statistics for convergence times and fitnesses of the best-so-far as well as final best-so-far fitnesses for runs where uniform sampling of rule intervals was performed for the *init* unbounded rule interval closing strategy.

Convergence times		Convergence fitnesses		Final fitnesses	
Minimum	1.000	Minimum	1.405	Minimum	1.405
1st Quantile	3.000	1st Quantile	4.277	1st Quantile	3.293
Median	4.500	Median	7.506	Median	4.512
Mean	5.633	Mean	7.739	Mean	4.680
3rd Quantile	7.750	3rd Quantile	9.505	3rd Quantile	5.346
Maximum	15.000	Maximum	16.150	Maximum	9.796
σ	3.891	σ	4.000	σ	2.102

Table 4.2: This shows a summary of statistics for convergence times and best-so-far fitnesses as well as final best-so-far fitnesses for runs where uniform sampling of rule intervals was performed for the *global* unbounded rule interval closing strategy.

Convergence times		Convergence fitnesses		Final fitnesses	
Min.	0.000	Min.	0.0879	Min.	0.00923
1st Qu.	3.250	1st Qu.	0.1746	1st Qu.	0.01791
Median	12.000	Median	0.2283	Median	0.02838
Mean	9.967	Mean	1.9390	Mean	0.03702
3rd Qu.	15.000	3rd Qu.	3.2280	3rd Qu.	0.04709
Max.	19.000	Max.	8.4960	Max.	0.13440
σ	6.488628	σ	2.776345	σ	0.02744918

values below 0.0001 showing there is a statistically significant difference between the three runs.

Fig. 4.6 shows the corresponding selection intensities for the same three sets of runs. Note, that the selection intensity for the *init* runs are very low — even negative in places – indicates that there is not much difference between parents and their offspring. In many cases the offspring can even be worse than their parents as indicated by negative selection intensities.

Table 4.1 shows the convergence times for these runs as well as the associated fitnesses of the best-so-far at the time the runs converged and at the end of the runs. This corroborates with what we see in the population trajectories in that this shows that the fitnesses for the *init* runs are still a fair distance from the global optimum when the run converges and generally has not improved much at the end of the runs; moreover, there is a high variance not only with the convergence times, but also of the fitnesses at the time the run converges.

Normally when an EA converges at the end of a run, the variances tend to become small. That the there is high variance when the runs converge and at the end of the runs indicates that the algorithm is struggling to narrow down to the global optimum. Indeed we can see this behavior in the *global* and *best* runs. Tbl. 4.2 shows the convergence statistics for the *global* runs, and there we can see that, though the runs converged later than the *init* runs, the fitnesses are much closer to the global optimum.

This mediocre behavior is an artifact of offspring genes resetting back to the initialization boundaries, thus imposing a force pulling the population away from the origin. Imagine that a population is condensing towards an optimum when one of the rule intervals is reset to the original initialization limits. That means that all of the subsequent offspring will now have values within that interval, which will be well away from the optimum the population was approaching.

This same behavior is observable for the Rastrigin and Griewangk functions, both rotated and non-rotated. Their plots can be found in Appendix A. However, this behavior is not noted for the Langerman test function because SLEM's performance was equally flat for all runs. The behavior of the step function merits its own discussion in §4.3.2 and §4.3.2.

4.2.3 *best* unbounded rule interval strategy generally greedier than *global*

Since *init* is demonstrably problematic, what about the *global* and *best* unbounded rule interval closing strategies? The intuition is that the *global* strategy is less greedy than *best* in that it presumably allows for greater exploration of search spaces. If this is true then we should expect to see that the *global* runs converge to smaller fitnesses than *best* for suitably complex fitness landscapes, and that *best* should similarly converge more quickly on the global optima for fairly simplistic fitness landscapes.

We have already seen that for the simple f_{sphe} function using a uniform distribution that it is indeed the case that the *best* appears to converge more quickly, again, as shown in Fig. 4.4. Fig. 4.7 shows that there are wide differences between the medians of the two strategies further corroborating what we observe. And we can see that the last generation

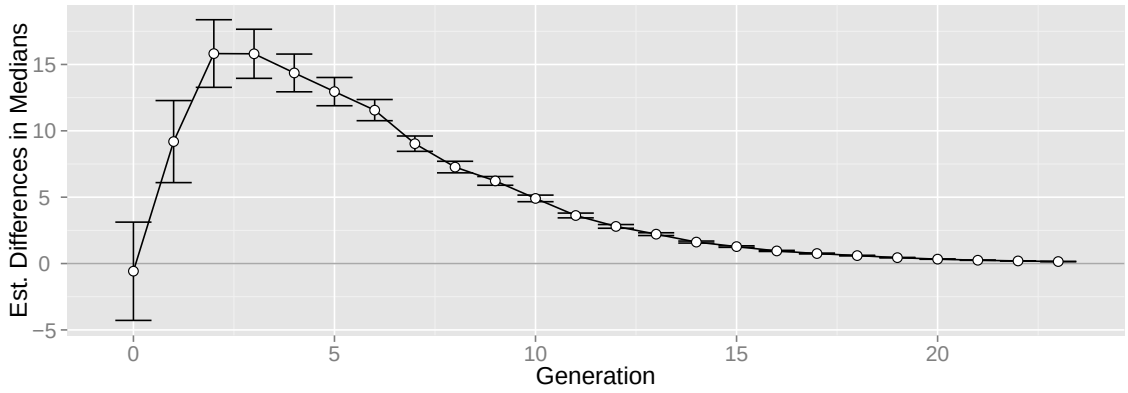


Figure 4.7: These are the by-generation 95% confidence intervals for comparisons between population medians of the *global* and *best* unbounded rule interval closing strategies; this was for the f_{sphe} test function using a uniform distribution to sample rule intervals. Values above zero favor *best*. As expected there is no difference in the first generation since that reflects that the initial populations for both sets of runs are essentially identical. However, thereafter the *global* runs immediately lag the *best* peaking at the second generation; after that the differences gradually lessen as the populations converge.

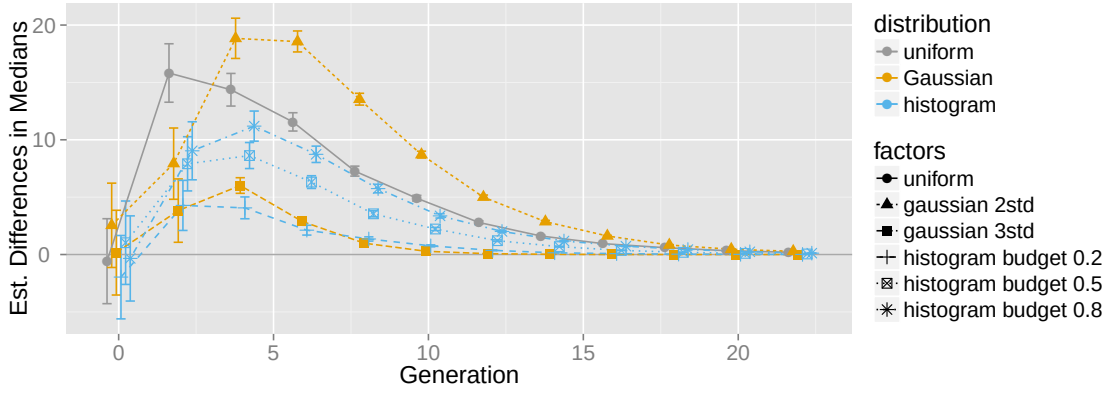


Figure 4.8: These are the 95% confidence intervals by every other generation for comparisons between population medians of the *global* and *best* unbounded rule interval closing strategies; this was for the f_{sphe} test function using uniform, Gaussian, and histogram-based distributions to sample rule intervals with each group shown in different colors and specific factors given unique shapes and line types. Again, values above zero favor *best*. We can see that the *best* has higher selection pressure until the runs converged regardless the rule interval sampling method. Note that this uniform *global* vs. *best* differences are the same as shown in Fig. 4.7.

the *best* strategy is closer to the global optimum than the *global* strategy as shown in Fig. 4.5. Looking at the other distributions, as shown in Fig. 4.8, there are similar differences for the Gaussian and histogram-based distributions rule interval sampling runs.

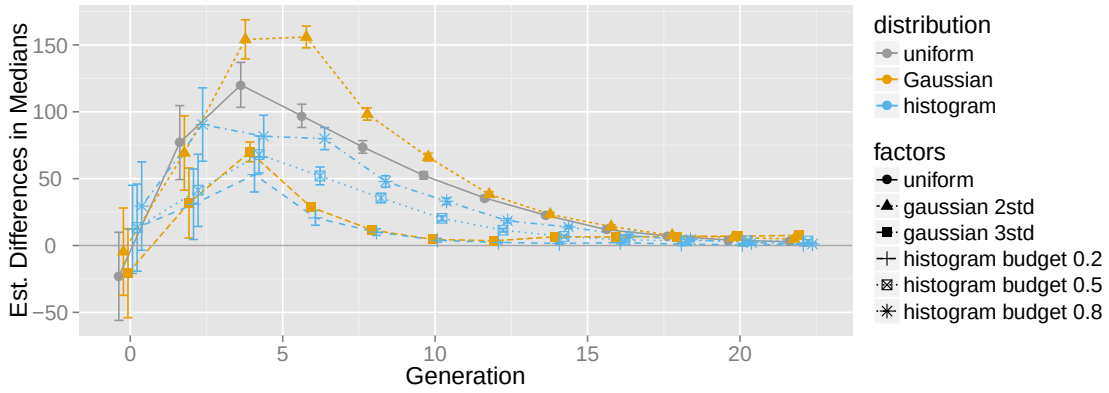


Figure 4.9: These are the 95% confidence intervals by every other generation for comparisons between population medians of the *global* and *best* unbounded rule interval closing strategies; this was for all the f_{rast} test function runs using uniform, Gaussian, and histogram-based distributions to sample rule intervals. Values above zero favor the *best* unbounded rule interval strategy. Once again, the *best* strategy has higher selection pressure than *global* strategy throughout all the runs until the have converged regardless of the rule interval sampling method.

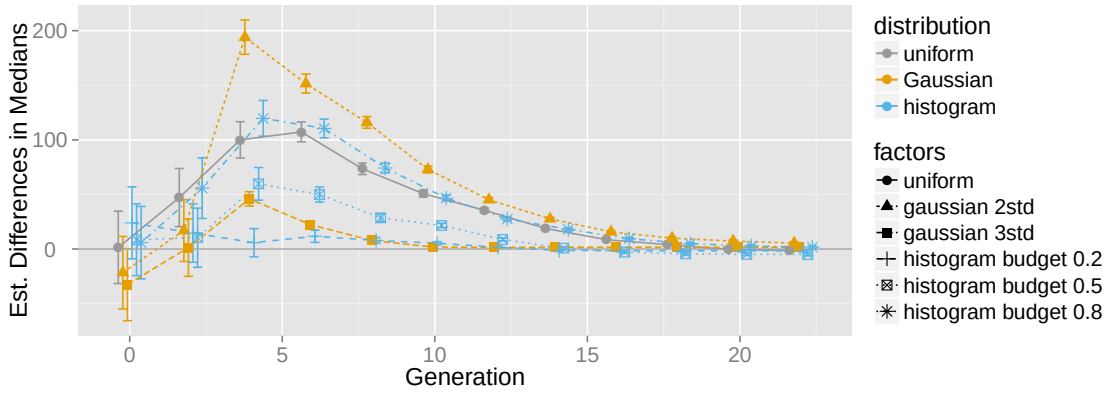


Figure 4.10: These are the 95% confidence intervals by every other generation for comparisons between population medians of the *global* and *best* unbounded rule interval closing strategies; this was for all the f_{rot_rast} test function runs for which a uniform, Gaussian, and histogram-based distributions were used to sample rule intervals when creating offspring. Values above zero favor the *best* unbounded rule interval strategy. The *best* unbounded rule interval strategy is more aggressive until the runs have converged for all rule interval methods.

For the last generation for all the rule interval sampling distribution types the *best* strategy runs were closer to the global optimum than the *global* (Wilcoxon rank sum tests $p < 0.0001$).

Note that the other f_{sphe} plots may be found in §A.1 on page 174.

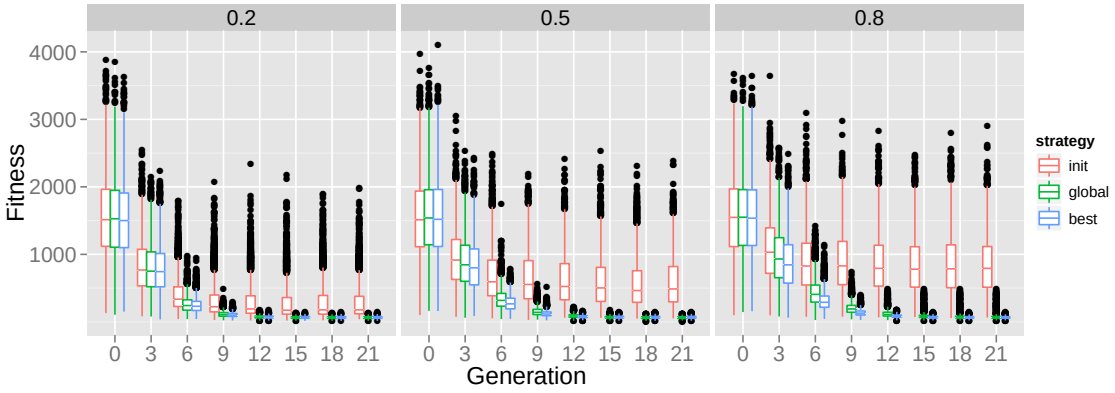


Figure 4.11: Population trajectories for f_{rot_rast} using a histogram-based distribution to sample rule intervals broken down by budget dedicated to a uniform distribution. We can see that the *init* once again lags behind *global* and *best* unbounded rule strategies, though the amount by which it does so is inversely proportional to the amount of uniform distribution mixed in. The *global* and *best* strategies have similar trajectories with the *best* appearing closer to the global optima than the *global* strategy.

The f_{rast} test function is similarly easy as the f_{sphe} so runs show similar characteristics as shown in Fig. 4.9. This pattern is, as with f_{sphe}, repeated for the Gaussian and histogram-based rule interval sample runs. Please note that the population trajectories for these runs can be found in §A.3 on page 182. Of particular note is that the *best* runs converged more quickly at the onset for the runs that had the least aggressive sampling policies. I.e., the uniform, Gaussian runs using two standard deviations, and the histogram runs with a 0.8 uniform distribution budget seemed to give the *best* unbounded rule interval closing strategy the most traction. However, note that the *best* strategy runs final fitnesses were closer to the global optimal than the *global* strategy — except for the 0.2 and 0.8 histogram based rule interval runs. (Wilcoxon rank sum test p-value = 0.2446 for histogram-based 0.2 runs, and p-value = 0.4735 for the 0.8 runs; all other runs $p < 0.001$ in favor of *best* strategy runs being closer to global optimal at the end of runs.)

Fig. 4.10 shows the results of rotating the Rastrigin test function to increase its difficulty; we observe that the performance characteristics between *global* and *best* echo their non-rotated counterparts with one notable exception, that of the histogram 0.2 budget runs. There we can see that there is no notable difference between the *global* and *best* for the first

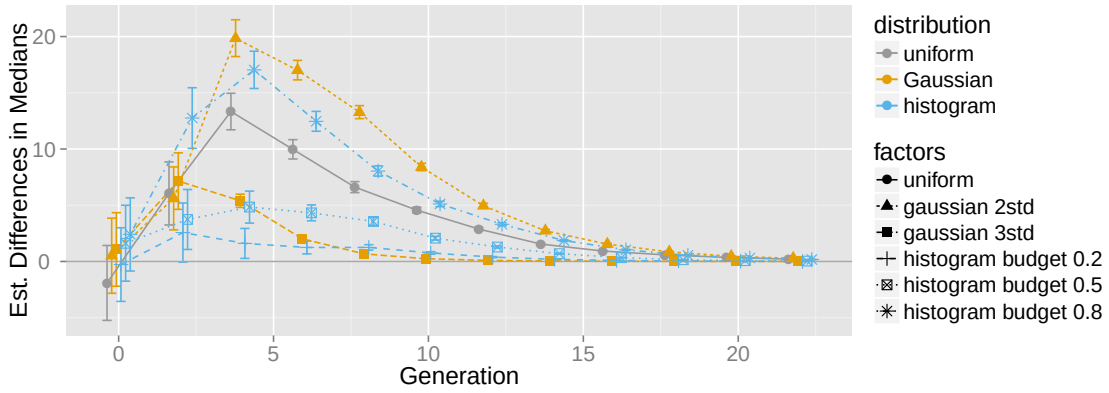


Figure 4.12: These are the 95% confidence intervals by every other generation for comparisons between population medians of the *global* and *best* unbounded rule interval closing strategies for all f_{grie} runs. The median comparisons for the three interval sampling methods are shown in differing colors; they are further broken down by specific factors by shape. Values above zero favor the *best* unbounded rule interval strategy. We can see a repeat of the established pattern whereby the *best* strategy has higher selection pressure regardless the rule interval method for most of the runs until the runs have converged.

start of the runs before the *best* runs ahead, but just for about ten generations when *global* is barely better. Fig. 4.11 shows the corresponding population trajectories where we can see that given the range of fitneses, [0,4000], that a difference of medians between the two strategies is small. However, the uniform, Gaussian two standard deviation, and histogram runs of 0.8 uniform distribution budget still do well for *best*.

For the rotated Rastrigin end of runs, which strategy converged more closely to the global optimum is mixed. The *best* strategy converged more closely for both sets of the Gaussian (Wilcoxon rank sum $p < 0.001$), and for the histogram runs with α 0.8 runs (Wilcoxon rank sum test $p = 0.02332$). For all the other runs the *global* open rule interval closing strategy converged more closely to the global optimal (Wilcoxon p-value = 0.002818 for the uniform runs and $p < 0.001$ for histogram runs for α 0.2 and 0.5 runs).

Fig. 4.12 shows the comparisons between *global* and *best* strategy medians for all the f_{grie} runs, which shows that the *best* either outright converges more quickly than *global*, or there is no significant difference. One notable features, though, is that the two σ Gaussian *best* runs are way ahead of the *global*. With regards to the population fitnesses at the end of runs, the *best* converges more closely to the global optimum than the *global* strategy

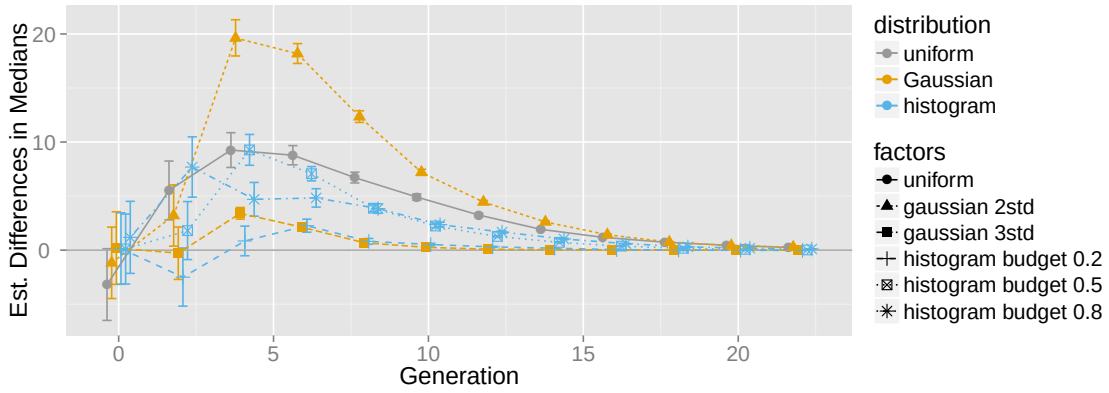


Figure 4.13: These are the 95% confidence intervals by every other generation for comparisons between population medians of the *global* and *best* unbounded rule interval closing strategies for all f_{rot_grie} runs. The median comparisons for the three interval sampling methods are shown in differing colors; they are further broken down by specific factors by shape. Values above zero favor the *best* unbounded rule interval strategy. We can see that the *best* has higher selection pressure at the beginning of the runs for all rule methods, and that this difference anneals over time. Of note is that the 0.5 histogram approach the *global* briefly has higher selection pressure.

(Wilcoxon p-value < 0.001) except for Gaussian runs using three standard deviations and the histogram runs using an α of 0.2 where there was no statistical difference between *best* and *global* (Wilcoxon p-values of 0.3263 and 0.2645, respectively).

Rotating the f_{grie}, as shown in Fig. 4.13, we see that there is still a wide margin between the *best* and *global* runs for the two σ Gaussian runs. Regardless, as with the non-rotated version, the *best* converges more quickly and closer to the global optimum than *global*. However the end of the run differences between fitness distributions is, again, mixed. The *best* strategy converges closer to the global optimum for all the runs (Wilcoxon $p < 0.001$, for those runs except for the Gaussian 3 standard deviation runs, which had a $p = 0.02546$), except for the histogram-based runs using α of 0.2 and 0.5. In the latter two cases, for the histogram-based runs using an α of 0.2, the *global* converged more closely (Wilcoxon $p < 0.001$), and for the α 0.5 histogram-based runs, there was no significant difference between the *best* and *global* strategy runs (Wilcoxon $p = 0.101$).

Fig. 4.14 tells a different story than we have seen so far. So far the *best* strategy has been greedier by consistently having trajectories that move closer to the global optimum

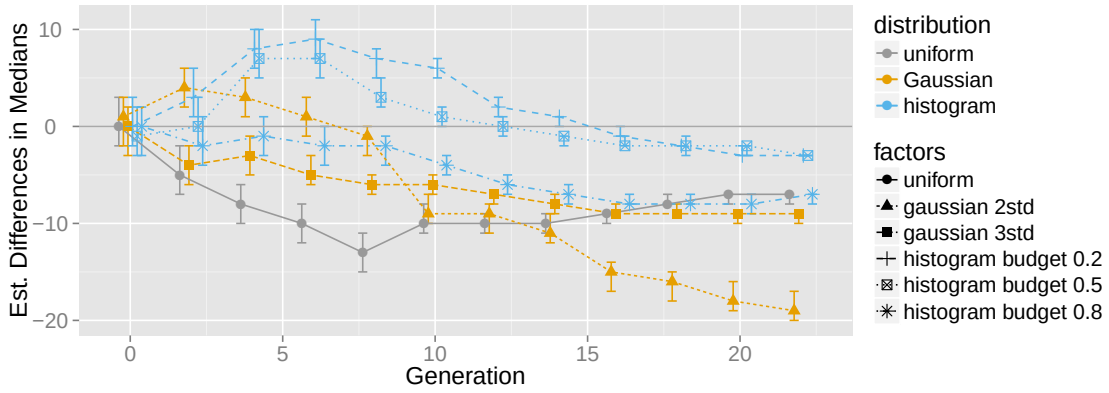


Figure 4.14: These are the 95% confidence intervals by every other generation for comparisons between population medians of the *global* and *best* unbounded rule interval closing strategies; this was for all the f_{step} test function runs using uniform, Gaussian, and histogram-based distributions to sample rule intervals. Values above zero favor the *best* unbounded rule interval strategy, those below the *global*. *global* strategy was consistently closer to the global optimum for the uniform and histogram α 0.8 runs as well as the Gaussian 3 σ runs; for the other three runs the *best* strategy was closer at the start, but then the *global* strategy runs once again converged more closely to the global optimum.

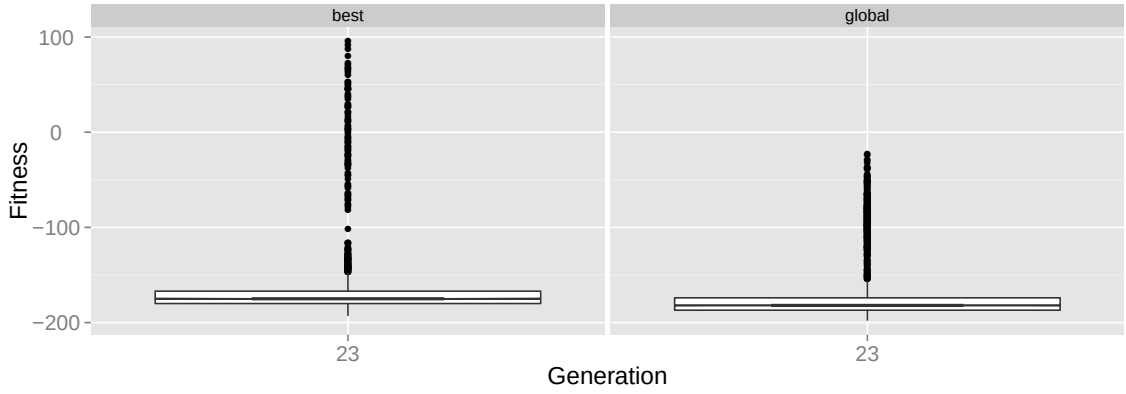


Figure 4.15: This shows that for the last generation of the f_{step} runs, the *global* runs converged more closely to the global optimum than the *best* open rule interval closing strategy.

than the *global* strategy. With the f_{step} function, however, we see that is not the case. Though *best* still converges more quickly at the start of some of the runs, eventually it is the *global* strategy that overtakes *best* sooner or later. Indeed, we see at the end of the run, as shown in Fig. 4.15, the *global* is closer to the global optimum than the *best* strategy. (The medians for the *global* less than *best* strategy runs Wilcoxon rank sum test p ¡ 0.0001

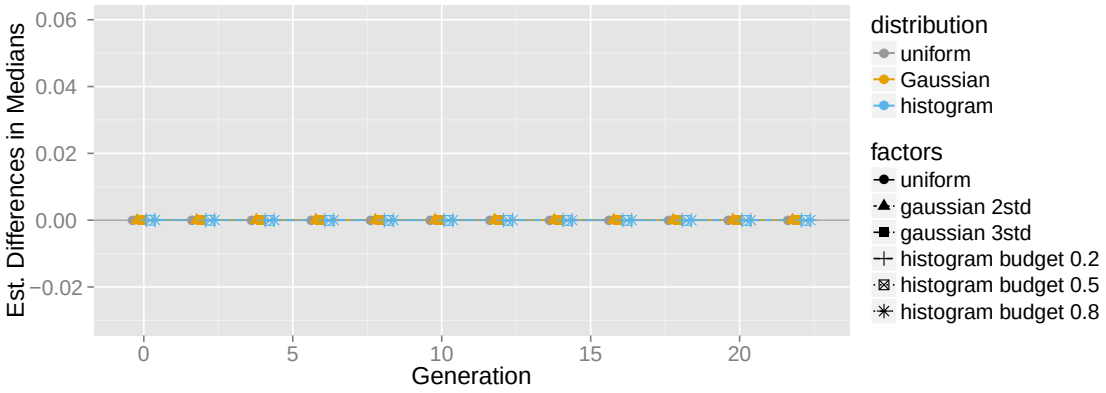


Figure 4.16: These are the 95% confidence intervals by every other generation for comparisons between population medians of the *global* and *best* unbounded rule interval closing strategies for all f_{lang} runs. The median comparisons for the three interval sampling methods are shown in differing colors; they are further broken down by specific factors by shape. This shows that there is no discernible difference between *global* and *best* for all experiments.

for all rule interval sampling distributions.)

What is going on here is that the step function does not have basin shaped "bowls" attraction within the space of exploration. Instead there are large plateaus stepping down towards negative infinity. The *global* strategy has a higher probability of creating individuals the next "step" down than the *best* strategy.

This premise is supported by observing the uniform and histogram-based approaches. If we think of the uniform approach as just yet another histogram-based run with an α of 1.0 we can see that gradually reducing the α — i.e., narrowing the search — that for the 1.0 and 0.8 runs the trajectories are entirely in *global*'s favor, and for the 0.5 and 0.2 runs the trajectories cross over from *best* to *global* later and later.

When doing search, we typically have an exploration phase followed by an exploitation phase. Essentially for the f_{step} runs the population never leaves the exploration phase as they steadily march towards the global optima at negative infinity they will never reach.

Fig. 4.16 shows the comparisons between the *global* and *best* runs for the Langerman function. There we can see that there is no difference between all the runs in stark contrast to the previous runs. Fig. 4.17 tells us why. With a few outliers as exceptions, SLEM finds no traction on the Langerman function. As Fig. 4.18 shows, the machine learner fails to

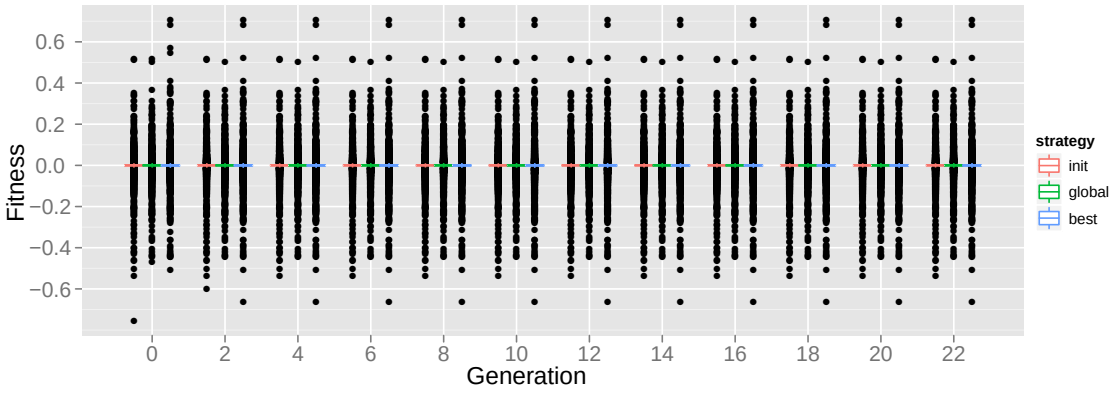


Figure 4.17: Population trajectories for f_{lang} using a uniform distribution to sample rule intervals, which shows that all runs get trapped in local minima.

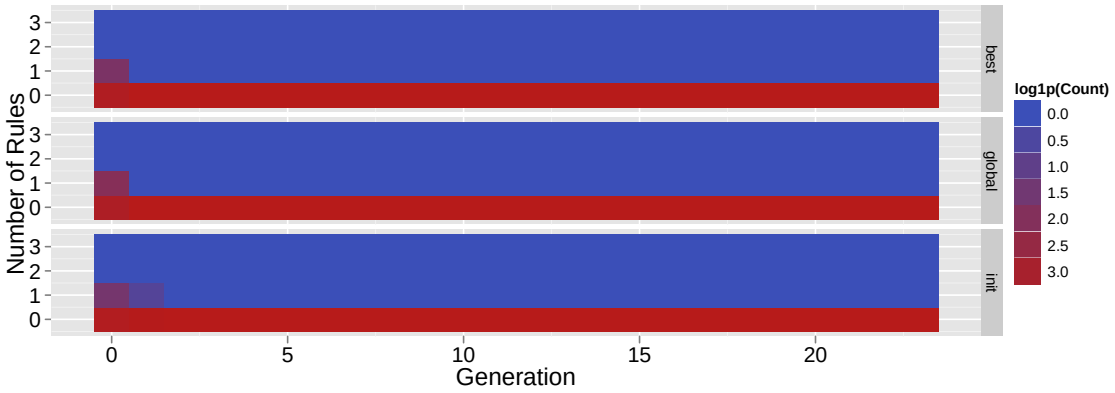


Figure 4.18: Heat maps for the aggregate number of rules for f_{lang} using a uniform distribution to sample rule intervals. The counts of the rules have been $log + 1$ transformed. This shows that the machine learner did not learn anything after the first generation for all runs.

learn anything after the first generation; and if there are no rules, then the system reverts to successive cloning of parents with no change. This same behavior occurs for the other rule interval sampling strategies.

4.2.4 Conclusions

This section addressed a problem regarding scenarios in which the machine learner infers only one bounds for a given rule interval. That is, we need both a lower and upper bound in rule intervals in which to sample values when creating offspring. So, if one of the bounds

is missing, we need to substitute a value for that missing bound.

I explored three different approaches for "closing" unbounded rule intervals. The first approach substitutes one of the original initialization bounds for missing interval values. The second substitutes the most extreme gene value from the current population of parents. The third substitutes the most extreme gene value from only the "best" parents. I refer to these approaches, respectively, as the *init*, *global*, and *best* unbounded rule interval strategies.

We learned that the *init* strategy is disruptive. As a population approaches an optima and the ML learns a rule that requires one or more intervals to be closed, closing those unbounded intervals to the original bounds that the population was initialized to had the effect of resetting the population. So with the *init* strategy the population would converge for a few generations before progress would stagnate well away from the global optima regardless of the fitness function or the rule interval sampling method. Though practitioners might be tempted to use this approach due to its easy implementation, I would recommend one of the other two methods, *global* or *best*, instead, to avoid this disruptive effect.

With regards to the *global* and *best* unbounded rule interval strategies, as we might expect the *best* strategy is demonstrably greedier except for the very end of the runs. This holds for all fitness functions and rule interval sampling methods, with the notable exception of the Langerman function, which I will address shortly.

That is, by substituting the extreme values from just the "best" parents for unbounded rule interval bounds we are naturally being more greedy, which is reflected in higher selection pressure. However, the difference between the *best* and *global* diminish as the runs progress because the population becomes more homogeneous; i.e., at the end of runs there is not that much difference between the most extreme gene values from the entire population and those from the highest performing parents.

There were two notable exceptions to this. The first was the step function where the *global* runs were able to converge more closely to the global optima because the wider net case for those runs more often than not allowed populations to find the next "step" down than the *best* strategy runs. The second was the Langerman function. A very large

basin of attraction of local optima trapped the population from the onset, so regardless of the unbounded rule interval strategy — or even of rule interval sampling method — the populations were unable to progress towards the global optima beyond even the first generation. This is phenomenon is further discussed in §5.2.2.

4.3 Rule Interval Sampling Approaches

Now that we have explored various strategies for resolving unbounded rule intervals, we can now look at the effects different means of sampling rule intervals have on selection pressure. In this section we will look at the three types of distributions that were described earlier, namely the uniform, Gaussian, and histogram-based probability distributions, and the significant differences in selection pressure that result from choosing one over the others.

We can get an intuition on the influence that each of these distributions might have on the selection pressure. For one thing, the uniform distribution is likeliest to have the least selection pressure given that interesting problems tend not to have featureless, flat fitness landscapes. Indeed many optima for various problems have bowl-like characteristics for which a Gaussian distribution might have a better chance of matching. And the histogram approach is the likeliest to match the underlying topology of the fitness function given that the histograms are created from actual gene values. Moreover, the histogram approach α parameter allows for specifying how much of a uniform distribution is blended with the histogram values. So, given my earlier assertion regarding the uniform distribution, the higher the α the less selection pressure we should see since higher α values more closely resemble the uniform distribution.

4.3.1 Results

In this section I share the results from the various rule interval sampling method runs. The results are broken down by fitness function.

However, before I go into the results, I need to relate what open rule interval closing strategies I chose for all of the runs. That is, I selected a single open rule interval closing

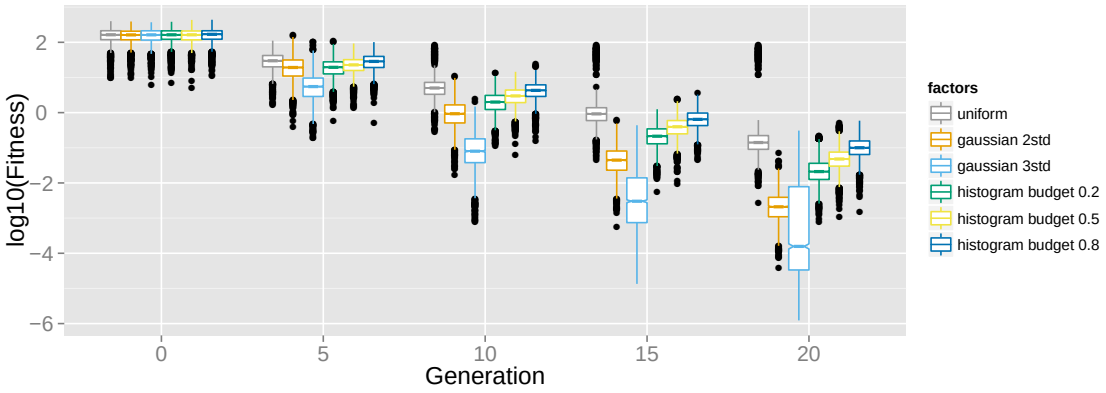


Figure 4.19: f_{sphe} fitnesses scaled logarithmically by all rule interval sampling distributions for every fifth generation. The uniform rule interval sampling method converged the most slowly of all method. The Gaussian-based distributions converge more quickly than the rest with the runs using rule intervals encompassing a distribution of three standard deviations converging the most quickly. The histogram-based approach convergence velocity is in proportion of greediness; i.e., a 0.2 uniform distribution budget means that 80% of the distribution is based on actual gene frequencies — the higher the budget the more the histogram approach comes to resemble a uniform distribution.

strategy for each test function because this section is not focused on comparing the different strategies because that has already been done in the previous section. In any case, there are three different unbounded rule interval strategies from which to choose: *init*, *global*, and *best*. I did not consider *init* for the disruptive effects described in §4.2.2 on page 54, so that leaves *global* and *best* for consideration. Given that previous LEM implementors preferred greedy behavior, I surmised that it is likelier that future practitioners would be biased in favor of greedier behavior, so I chose to use the *best* unbounded rule interval closing strategy for all the runs in this section. In a later section I do a more comprehensive comparison of the different rule interval sampling methods and the unbounded rule interval strategies.

I will now share the results by test function, followed by a discussion of those results.

Spheroid

Fig. 4.19 shows the logarithmically transformed population trajectories for f_{sphe} for all the rule interval sampling methods. There were can see that the Gaussian runs had the highest selection pressure as compared to the highest runs, with the Gaussian runs that

Table 4.3: One tailed pairwise Wilcoxon rank sum p-values for fitnesses of the last generation for f_{sphe} with Bonferonni adjustment with the hypothesis being that the median fitnesses of the row values are less than the column. This corroborates what we see in 4.19 — that the Gaussian 3σ runs converge the most closely to the global optima and the uniform the furthest.

	uniform	Gaussian 2σ	Gaussian 3σ	histogram 0.2	histogram 0.5
Gaussian 2σ	< 0.001				
Gaussian 3σ	< 0.001	< 0.001			
histogram 0.2	< 0.001	1	1		
histogram 0.5	< 0.001	1	1	1	
histogram 0.8	< 0.001	1	1	1	1

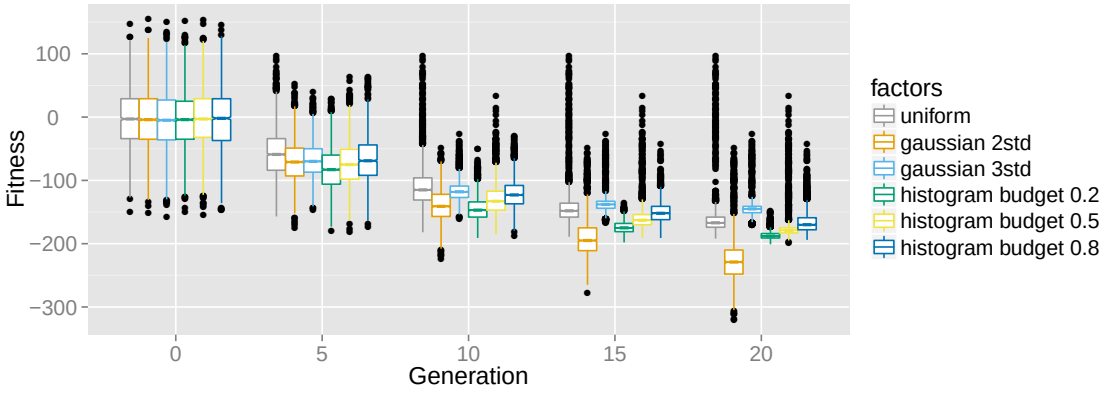


Figure 4.20: f_{step} fitnesses for all rule interval sampling distributions for every fifth generation. The Gaussian rule interval sampling method that uses a two standard deviations to sample intervals is the only set of runs that move past the implicit -210 constraint dictated by the initial population, which is further discussed in §4.3.2. The Gaussian runs that sample rule intervals within three standard deviations converges towards a more conservative implicit constraint.

sampled intervals within three standard deviations being the most aggressive. By contrast, the uniform sampling method had the least selection pressure. The histogram-based rule interval sampling method had selection pressure midway between the uniform and Gaussian, with the selection pressure inversely proportional to α.

Tbl. 4.3 shows the end-of-run statistics for f_{sphe} and echoes what we have already seen in Fig. 4.19; i.e., that the Gaussian 3 σ runs converge more closely to the global optima, and the uniform runs the furthest.

Table 4.4: One tailed pairwise Wilcoxon rank sum p-values for fitnesses of the last generation for f_{step} with Bonferonni adjustment with the hypothesis being that the median fitnesses of the row values are less than the column. This corroborates what we see in Fig. 4.20; i.e., the Gaussian 2σ runs can continue to move towards negative infinity, whereas the Gaussian 3σ end up the furthest from all other runs.

	uniform	Gaussian 2σ	Gaussian 3σ	histogram 0.2	histogram 0.5
Gaussian 2σ	< 0.001				
Gaussian 3σ	1	1			
histogram 0.2	< 0.001	1	< 0.001		
histogram 0.5	< 0.001	1	< 0.001	1	
histogram 0.8	< 0.001	1	< 0.001	1	1

Step

Fig. 4.20 shows the population trajectories for f_{step} for all rule interval sampling methods. The Gaussian interval methods converge both the furthest and closest to the global optima — i.e., the two σ runs continue to converge steadily towards the global optima at $-\infty$ whereas the three σ runs converge towards a fairly conservative implicit barrier. The uniform and histogram-based approaches asymptotically converge to another, less conservative implicit constraint at -210, which is discussed further in §4.3.2 on page 75. The uniform rule interval sampling method is the second most most conservative; and, again the histogram-based approach's selection pressure is inversely proportional to α. That is, the higher the α, the more similar the histogram-based rule interval sampling method is to the uniform.

Tbl. 4.4 compares the respective fitness medians between the runs and tells the same story. That is, the Gaussian 2σ runs are far closer to the global optimum than the other runs because that distribution allows populations to move through the implicit bounds dictated by the initial population. Moreover, the narrower distribution of the 3 σ Gaussian runs has a more conservative asymptote due to likely getting caught on a local minima, and so is furthest from the global optima than all other runs.

Rastrigin

Fig. 4.21 depicts the population trajectories for all rule interval sampling methods for f_{rast}. At the start of the runs it is the Gaussian methods that converge the most closely to the

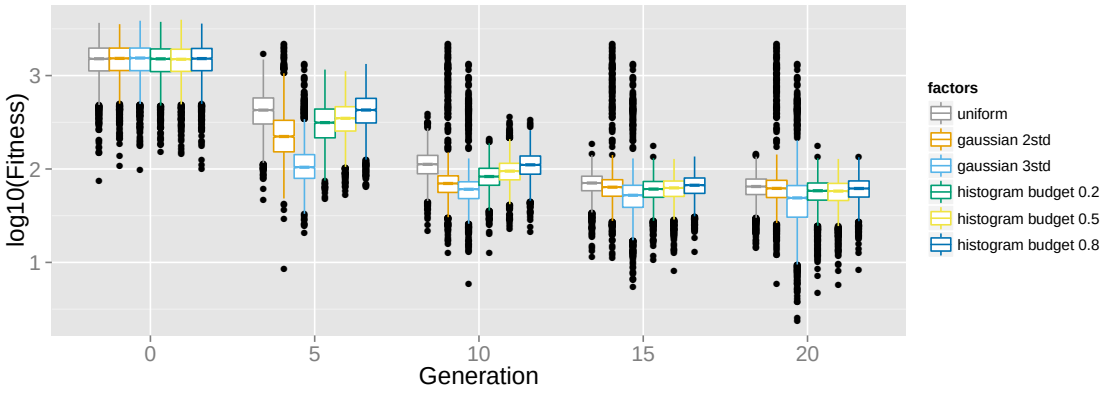


Figure 4.21: f_{rast} fitnesses scaled logarithmically by all rule interval sampling distributions for every fifth generation. The Gaussian three σ rule interval sampling method converged the most quickly and the closest to the global optima. The uniform and histogram 0.8 converged the most slowly. As the amount of uniform sampling in the histogram method was reduced, the selection pressure increased.

Table 4.5: One tailed pairwise Wilcoxon rank sum p-values for fitnesses of the last generation for f_{rast} with Bonferonni adjustment with the hypothesis being that the median fitnesses of the row values are less than the column. The runs that used a uniform distribution to sample rule intervals had a fitness median furthest from the global optima whereas the Gaussian 3 σ runs were closest.

	uniform	Gaussian 2σ	Gaussian 3σ	histogram 0.2	histogram 0.5
Gaussian 2σ	< 0.001				
Gaussian 3σ	< 0.001	< 0.001			
histogram 0.2	< 0.001	< 0.001	1.0		
histogram 0.5	< 0.001	< 0.001	1.0	1.0	
histogram 0.8	< 0.001	1.0	1.0	1.0	1.0

global optima, but after the runs have converged the two standard deviation-based runs are on par with the other methods, whereas the three standard deviation-based runs converge a little more closely to the global optima. The uniform runs converge the most slowly of all the methods. However, the histogram-based approach convergence rate is, as before, inversely proportional to α with the highest value, 0.8, resembling the uniform distribution runs.

Tbl. 4.5 shows the pairwise one-tailed Wilcoxon rank sum statistics with Bonferonni adjustment comparing the last generation fitnesses for each of the f_{rast} runs. From there we can observe that the uniform runs had median values furthest from the global optima, whereas the Gaussian 3σ runs were the closest.

The randomly rotated Rastrigin runs, as shown in Fig. 4.22, are almost identical to

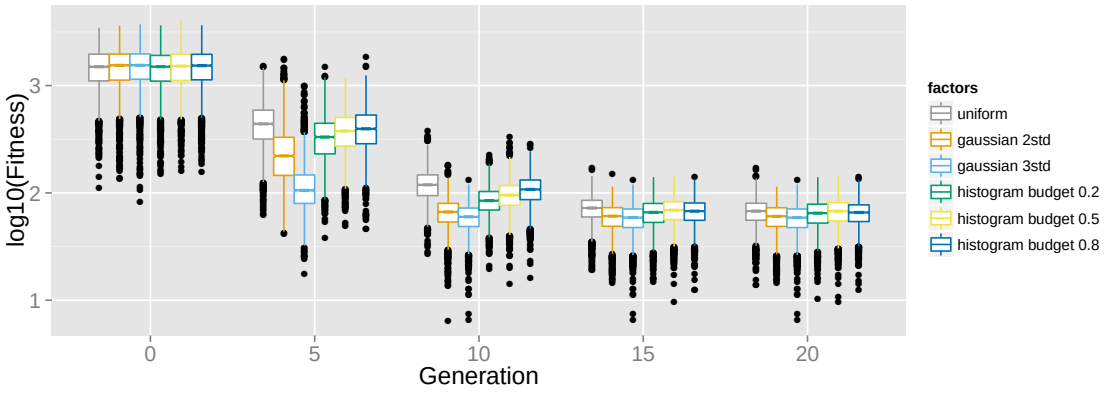


Figure 4.22: f_{rot_rast} fitnesses scaled logarithmically by all rule interval sampling distributions for every fifth generation. The trajectories are similar to those in Fig. 4.21 except the Gaussian three σ runs do not converge as closely to the global optimum after the Rastrigin function is randomly rotated.

Table 4.6: One tailed pairwise Wilcoxon rank sum p-values for fitnesses of the last generation for f_{rot_rast} with Bonferonni adjustment with the hypothesis being that the median fitnesses of the row values are less than the column. The histogram based runs with an α of 0.5 were the furthest from the global optimum, whereas the runs that used a uniform distribution were the second furthest. The Gaussian 3σ runs were the closest to the global optima of all the runs for the last generation..

	uniform	Gaussian 2σ	Gaussian 3σ	histogram 0.2	histogram 0.5
Gaussian 2σ	< 0.001				
Gaussian 3σ	< 0.001	0.034			
histogram 0.2	< 0.001	1.0	1.0		
histogram 0.5	1.0	1.0	1.0	1.0	
histogram 0.8	< 0.001	1.0	1.0	1.0	0.001

non-rotated Rastrigin. The most visible difference is that the non-rotated Rastrigin three standard deviation Gaussian rule interval runs were able to converge more closely to the global optima than the randomly rotated version.

Tbl. 4.6 shows the results of a one-tailed pairwise Wilcoxon rank sum test between the different rule interval sampling approaches for f_{rot_rast} using a Bonferonni correction. Here was that the uniform runs were not the furthest from the global optima, but instead the histogram α 0.5 runs were furthest. However, the Gaussian 3σ runs were the closest to the global optima.

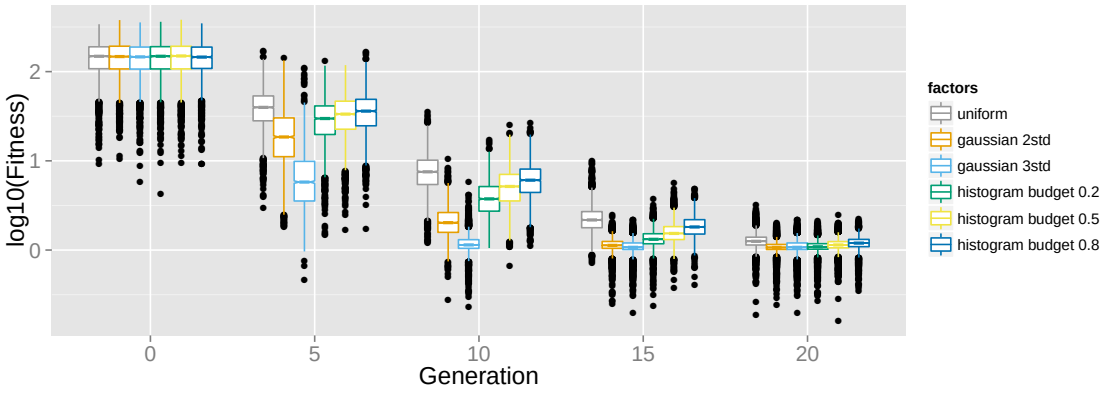


Figure 4.23: f_{grie} fitnesses scaled logarithmically by all rule interval sampling distributions for every fifth generation. The uniform rule sampling method converges the most slowly and furthest from the global optimum. The histogram based approach selection pressure is inversely proportional to the amount of uniform distribution blend. The Gaussian-based methods have the highest selection pressure with the three σ runs being the most aggressive of the two.

Table 4.7: One tailed pairwise Wilcoxon rank sum p-values for fitnesses of the last generation for f_{grie} with Bonferonni adjustment with the hypothesis being that the median fitnesses of the row values are less than the column. The uniform and histogram α 0.8 runs are somewhat similar, and both are furthest from the global optimum. The Gaussian 2σ runs are the closest to the global optimum.

	uniform	Gaussian 2σ	Gaussian 3σ	histogram 0.2	histogram 0.5
Gaussian 2σ	< 0.001				
Gaussian 3σ	< 0.001	1			
histogram 0.2	< 0.001	1	1		
histogram 0.5	< 0.001	1	1	1	
histogram 0.8	0.052	1	1	1	1

Griewangk

Fig. 4.23 depicts the population trajectories of the f_{grie} runs. There we see a repeat of the well established pattern: the uniform runs converge the most slowly, the Gaussian the most quickly with the three standard deviation runs being the most aggressive, and the histogram runs distance to the global optima inversely proportional to the α parameter settings.

Tbl. 4.7 shows the end-of-run comparison between all the runs. Here we see that the uniform and histogram α 0.8 runs are somewhat similar ($p = 0.052$) and are both furthest from the global optimum. However, unlike other runs, this time the Gaussian 2σ runs are closest to the global optimum, not the Gaussian 3σ runs as seen earlier.

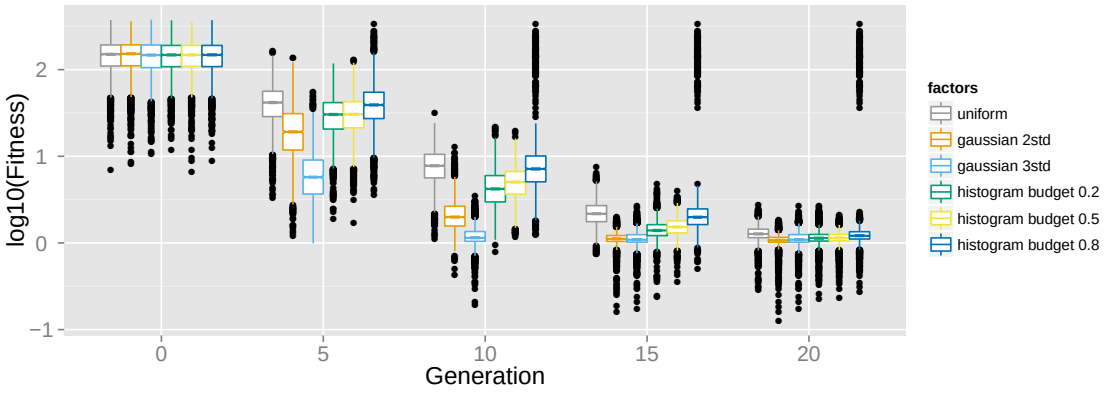


Figure 4.24: f_{rot_grie} fitnesses scaled logarithmically by all rule interval sampling distributions for every fifth generation. These runs are essentially identical to the non-rotated Griewangk runs seen in Fig. 4.23.

Table 4.8: One tailed pairwise Wilcoxon rank sum p-values for fitnesses of the last generation for f_{rot_grie} with Bonferonni adjustment with the hypothesis being that the median fitnesses of the row values are less than the column. The runs that used a uniform distribution to sample the rule intervals ended their runs furthest from the global optimum, whereas the runs that used a Gaussian of 2 σ were the closest.

	uniform	Gaussian 2σ	Gaussian 3σ	histogram 0.2	histogram 0.5
Gaussian 2σ	< 0.001				
Gaussian 3σ	< 0.001	1			
histogram 0.2	< 0.001	1	1		
histogram 0.5	< 0.001	1	1	0.082	
histogram 0.8	< 0.001	1	1	1	1

Randomly rotating the Griewangk function, for which the population trajectories are shown in Fig. 4.24, did not have a noticeable impact on the population trajectories.

Tbl. 4.8 compares the fitnesses for the last generation between the different rule interval sampling approaches. Here we see that the uniform distribution was generally the furthest from the global optima, whereas the Gaussian of 2 σ was the closest.

Langerman

Fig. 4.25 shows the population trajectories for f_{lang}, which shows that the populations for all the runs are not significantly different from one another.

4.3.2 Discussion

Here I discuss the results for the various rule interval sampling methods.

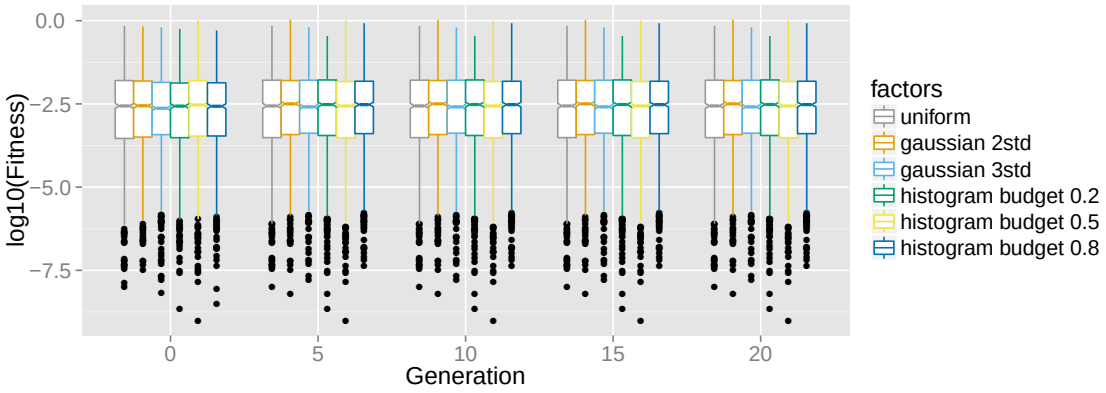


Figure 4.25: f_{lang} fitnesses scaled logarithmically by all rule interval sampling distributions for every fifth generation, and shows that the population does not escape a large basin of attraction for a local optima.

Initial population defines implicit exploration bounds

For SLEM the sole source of novelty is from the machine learner — there is no mutation nor crossover. However, the machine learner only knows what it is given as training data of the best and least fit parents, and so it may be the case that the initial parent population may define implicit bounds from which subsequent offspring may not escape. That is, the ML will learn from the initial population a set of rules, and the rule intervals will be within the bounds dictated by that initial population. And, since offspring gene values are either cloned from parents or sampled from rule intervals, it follows that all offspring will be within the bounds defined by that initial population.

The f_{step} test function is unique in the test suite in that its global optimum is not only outside the initial population, but is also unattainable given that it is at negative infinity. This captures real world scenarios where we do not know where the global optimum is for a problem, and so suitable initial population bounds is a guess; we would hope that the EA would be free to wander outside those bounds as necessary in seeking better solutions, barring any appropriate explicit constraints. Ideally for f_{step} fitnesses should steadily decrease over the course of the run without converging. However, given my earlier assertion that SLEM will stay within bounds dictated by the initial parent population, we should see SLEM runs converge on that border edge closest to the global optimum.

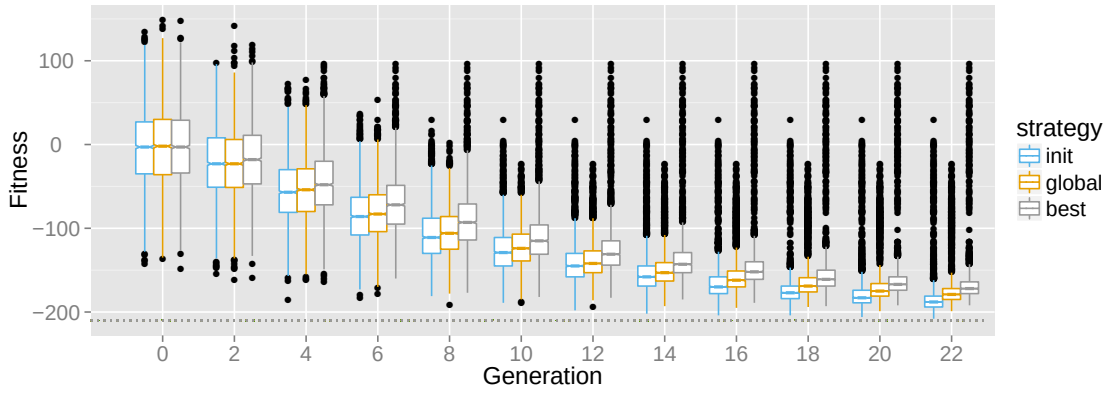


Figure 4.26: Population trajectories for f_{step} for three different open rule interval clamping strategies using a uniform distribution to sample rule intervals that all asymptotically approach an implicit bounds as -210 dictated by the initial parent population.

Indeed that is exactly what is shown in Fig. 4.26, which shows the population trajectories for the f_{step} runs where a uniform distribution was used to sample rule intervals. Given that the f_{step} runs had seven genes in the genome and that the initial population was initialized in $[-30, 30]$, then we would expect the population to asymptotically approach -210; i.e., -30 $\times$ 7, which appears to be the case. Of note is that this is one of the few times that *init* converges more quickly than either *global* or *best*. Since the implicit asymptote is along one of the edges, this is one time where the *init* strategy's bias works in its favor by resetting search spaces to that edge of the search space.

However, as shown in Fig. 4.1, it is possible that using a Gaussian distribution to sample rule intervals will allow for pushing through this implicit bounds since a few gene values will fall outside rule intervals. Fig. 4.27 shows that is true for runs using a Gaussian distribution of two standard deviations. The *global* and *best* runs pass the -210 boundary and appear to keep going without converging, as expected. However, this is not the case for the runs where three standard deviations was used. There an altogether different implicit constraint manifests, and one well below -210.

Intuitively sampling rule intervals with Gaussian distributions of three standard deviations will generate gene values in a much narrower range than those generated from two

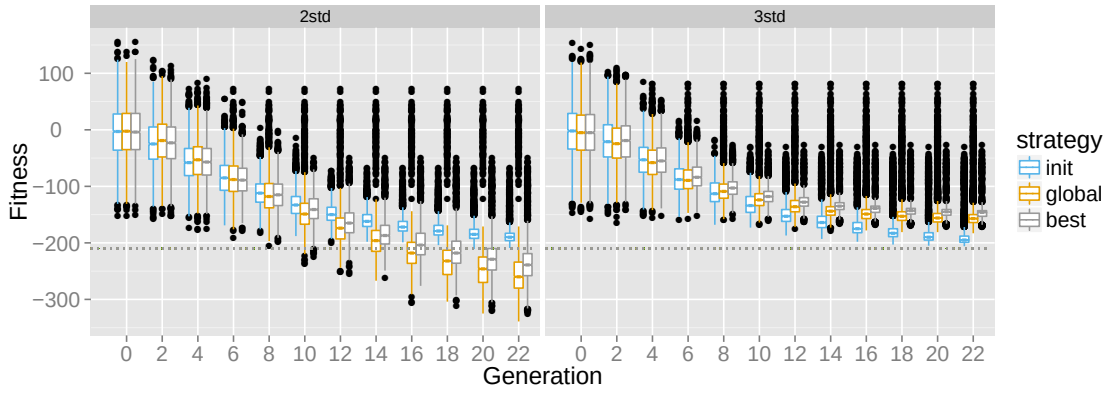


Figure 4.27: Population trajectories for $f_{step}()$ where Gaussian distributions of two and three standard deviations were used to sample rule intervals. Note that both the *global* and *best* runs for the 2 σ runs were able to move past the implicit -210 bounds whereas the 3 σ runs not only were unable to do so, but had a different, more conservative bounds that was asymptotically approached.

standard deviations. And, again, the ML only knows what it is shown. The initial population will still be within the implicit original bounds of -210; but because we are sampling within a slimmer band dictated by a narrower Gaussian distribution, that means that the next training sets will be for a smaller region. This subsequently leads to iteratively smaller regions identified as having higher fitness, and regions more centered within the previously defined rule intervals. I.e., by having such a narrow sampling of the search space, we are introducing a distorted view of that space. Moreover, the step function was originally designed to trap populations on plateaus [De Jong, 1975], and the slimmer sampling eventually does just that — the population becomes mired on one of the "steps" emblematic of f_{step}, and no further progress is possible.

Fig. 4.20 shows a side-by-side comparison of all the rule interval sampling distributions including the aforementioned uniform and Gaussian distributions. This figure shows that this phenomena similarly affects the histogram-based distributions.

A suitable mutation operator should enable the population to escape the bounds, but the focus here is to observe that just by using a Gaussian distribution to sample rule intervals is no guarantee that SLEM will be able to escape the bounds dictated by the initial population. Worse yet, a narrow Gaussian distribution may make runs more susceptible to being trapped

in local minima.

Gaussian runs have higher selection pressure

The runs that used a Gaussian distribution to sample rule intervals converged more quickly and closely to the global optimum than the other distributions. One might be tempted to think this is because of the distributions used the Gaussian distributions are unique in that they allow for sampling outside the rule intervals, albeit for a small percentage of individuals. I.e., these approaches to rule interval sampling cover a larger area than the alternatives, so one might think that the system is more readily able to find the global optimum. However, that might be true if there was external selection pressure via parent or survival selection that could take advantage of optimum discovery in that way. In this case, all other external forms of selection pressure have been intentionally removed; any observed convergence behavior is entirely from the influence of the machine learner and interval sampling. Besides, if this was truly the case, the wider distribution of two standard deviations would have converged more quickly than its three standard deviations counterpart when the opposite is true.

A hint to why the Gaussian distributions converge more quickly is offered by the histogram-based approach. Though all three runs for all histogram uniform budgets were very similar, the uniform budgets clearly influenced the convergence trajectories. The speed of convergence appears to be inversely proportional to the histogram budget dedicated to a uniform distribution. Indeed, the budget of 0.8, which has the largest uniform distribution influence, most closely resembles the trajectories of the actual uniform distribution runs. Again, the histogram approach, as depicted in Fig. 4.2 on page 50, shows that the rule interval is divided into bins into which counts from the 'best' parents' genes are accumulated and from which a probability distribution is calculated; however, a uniform distribution is blended with this distribution to mitigate problems associated with zero frequency bins. When the bins are filled for f_{sphe}, the preference will be the pick gene values nearer the origin where the global optimum is located since these are populated from the 'best' parents.

That is, the rule intervals will presumably straddle the center, the bins mapping over that interval will be chosen from the 'best' parents, which intrinsically are located near the center. And this means that the histogram, to a certain degree of fidelity, will roughly mirror that of the underlying landscape — the less the uniform budget for the histogram runs, the more aggressive this mirroring is for the gene values generated from the histogram.

This is essentially what is happening with the Gaussian approach. The three standard deviation runs happen to take advantage of the underlying fitness landscape. Using two standard deviations is too broad; it still incurs a sampling advantage, but not as aggressive as its three standard deviations counterpart.

This same kind of behavior can be observed in both the unrotated and rotated Rastrigin runs as show in Figs. 4.21 and 4.22. That is, again, the Gaussian runs generally converge more quickly than the other distributions.

However, there are some notable differences between the f_{sphe} and f_{rast} experiments. Though the Gaussian three standard deviation runs converge more quickly and closer to the global optimum, its two standard deviation counterpart gradually falls back to be similar to the other distributions after the system has more-or-less converged.

Uniform Runs Have Least Selection Pressure

It should come as no surprise that the uniform runs consistently demonstrated the least level of selection pressure. For all but the Langerman runs, which will be discussed later, the uniform rule interval sampling method populations lagged behind the other methods until convergence — with the notable exception of f_{step}, which was discussed earlier in §4.3.2 on page 75. The different methods were then similar at the end of the runs except for f_{sphe} and f_{step}, which were still converging when the runs were terminated.

Consider that all the fitness landscapes have a non-uniform shape to them so it will be the case that the uniform distribution will almost never match the underlying topology. Whereas the Gaussian- and histogram-based rule interval sampling methods have a better chance than the uniform of having distributions that at least partly fit the problem topology.

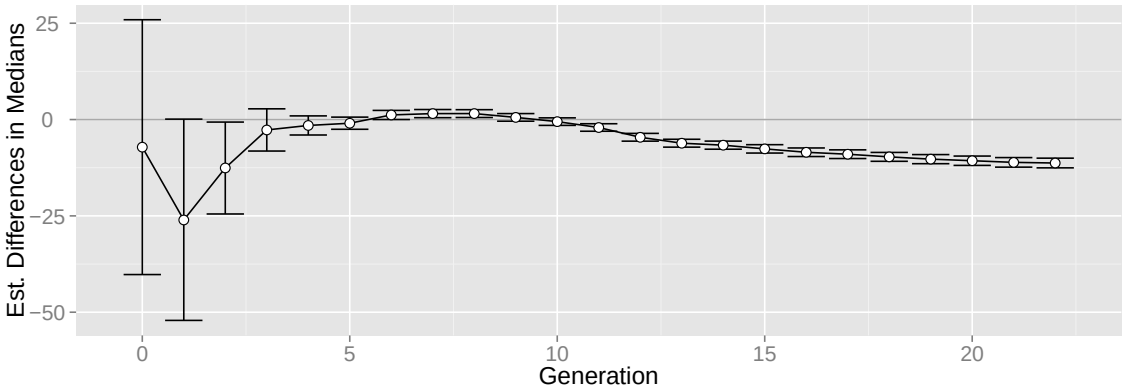


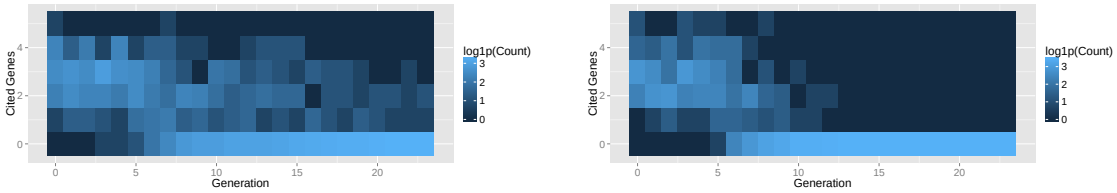
Figure 4.28: 95% confidence intervals of estimates of differences in medians by generation between non-rotated and rotated Rastrigin test function population fitnesses for the three σ Gaussian rule interval sampling methods. Values above zero indicate a preference for f_{rot_rast} for a given generation, where those below zero depict a preference for f_{rast}. For a single generation, the non-rotated Rastrigin runs are closer to the global optimum, then from generations seven through nine the randomly rotated Rastrigin runs are closer, then as the system converges the non-Rotated version steadily approach the global optimum.

We get a better sense of that fit between distributions and the fitness landscape in §5.3.2 on page 135 with regards to machine learner training sets that are dynamically adjusted to fitness values.

Effects of Rotation

From [Salomon, 1996] we know that rotating the Rastrigin and Griewangk functions can foil algorithms that have a built-in bias for axially aligned test functions. Does effecting a rotation change the difficulty of the Rastrigin and Griewangk functions for SLEM? I discuss the rotations of these functions in turn.

Fig. 4.28 shows the 95% confidence intervals of fitnesses by generation between unrotated and rotated Rastrigin test function runs for the Gaussian three standard deviation rule interval sampling method. We observe that the third generation shows that f_{rast} runs are better than the corresponding f_{rot_rast} runs. Then there is no difference between them for three generations; then the rotated runs are slightly better for three generations; then there two more generations of no significant differences. And then for the rest of the runs the non-rotated Rastrigin converges more closely to the global optimum (Wilcoxon p <



(a) This shows the aggregate count of cited genes by generation for f_{rast} using a Gaussian distribution with 3 σ rule interval sampling method. Though for most of the runs the ML has stopped learning anything after several generations, there are still many runs where the ML is still learning when the run terminates.

(b) This shows the aggregate count of cited genes by generation for f_{rot_rast} using a Gaussian distribution with 3 σ rule interval sampling method. We see here that the number of cited genes drops to zero after a dozen generation because the ML has stopped learning anything.

Figure 4.29: These heat maps with a log transformed color scale show the collective count of cited genes for all the non-rotated and rotated Rastrigin runs. For many of the runs the ML is able to continue learning rules — and thus cite genes — until terminated for the non-rotated Rastrigin runs. By contrast, the ML was unable to learn anything after a dozen generations when the Rastrigin function is randomly rotated.

0.001). These results suggests that SLEM may be gaining some leverage from the rectilinear configuration of the optima when the runs converge, particularly after the population has converged near the global optima.

Salomon recommended as a means of compensating for function rotation to increase the mutation rate to allow for more effective exploration, which was informally typically $1/L$ where L is the genome length. That is, a $1/L$ mutation rate was similar to performing a beam search, which may gain leverage from axially aligned problems, but will suffer when a given problem is rotated [Salomon, 1996]. Fig. 4.29 shows the number of cited genes by generation for all the Rastrigin runs. The number of cited genes is the effective mutation rate for SLEM, and we can see that it varies from generation to generation; moreover, generally more than one gene is cited until the runs converge, which means an effective mutation rate greater than $1/L$. So the greater mutation rate should mean that rotating the Rastrigin function should have no performance change, but we see that, instead, the end of the runs that the randomly rotated Rastrigin converges further from the global optima.

We can see that randomly rotating the Rastrigin function frustrated the ML in that after it converges it is no longer able to progress, whereas in the non-rotated Rastrigin the ML is able to make refined improvements, probably due to taking advantage of regularly

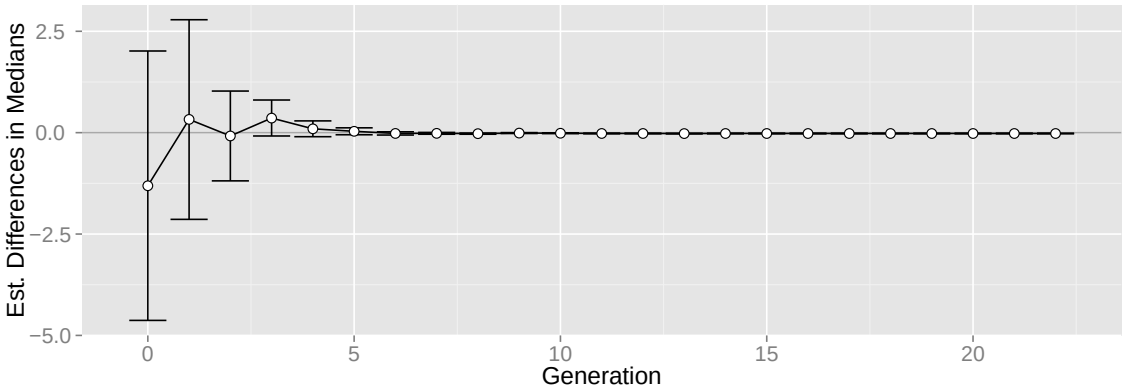


Figure 4.30: 95% confidence intervals of estimates of differences in medians by generation between non-rotated and rotated Griewangk test function population fitnesses using the Gaussian three σ rule interval sampling method. Values above zero indicate a preference for f_{rot_grie}, whereas those below zero show a preference for f_{grie}.

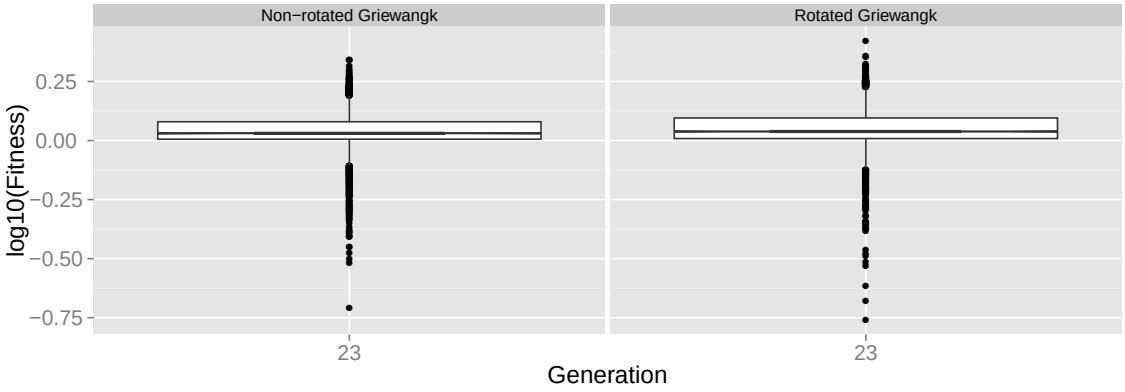
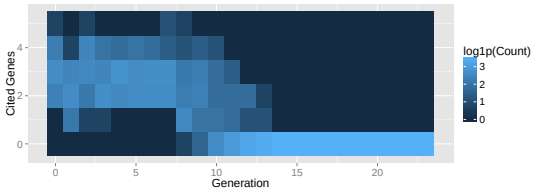


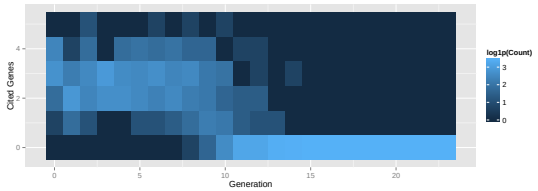
Figure 4.31: This shows the last generation differences between the population fitnesses between the f_{grie} and f_{rot_grie} runs. We can see that the non-rotated Griewangk runs are slightly closer to the global optima (Wilcoxon $p < 0.001$).

arranged topology near the global optimum.

As with the Rastrigin function, the Griewangk test function has both non-rotated and rotated versions. Fig. 4.30 compares these two runs. This plot, as with the Rastrigin function, shows the estimated difference between medians by generation. Unlike with the Rastrigin, there is are very small differences between the two runs — note that, as shown in Fig. 4.31, there is a very small, but statistically significant difference between the two sets of runs (Wilcoxon p ¡ 0.001); so randomly rotating the Griewangk function does not



(a) This heat map depicts the counts of the number of genes cited by rules by generation for the Gaussian three σ f_{grie} runs; the color scale for the counts is log transformed to make the lower number counts easier to see. For the first ten generations the rules usually cover two to three genes, occasionally four or single genes, and rarely five genes. Starting from the eighth generation the machine learner for a few runs does not learn anything, with the number of runs where this occurs increasing in numbers until by the 14th generation there are no rules learned for all runs.



(b) This heat map depicts the counts of the number of genes cited by rules by generation for the Gaussian three σ f_{rot_grie} runs. The pattern of cited rules for the randomly rotated Griewangk is very similar to that of the non-rotated version.

Figure 4.32: These heat maps with a log transformed color scale show a comparison of the pattern of the number of genes cited by generation between the non-rotated and randomly rotated Griewangk function. This shows a similar pattern whereby the ML learns between one and five genes, and mostly two to three, for the first several generations before the number of cited genes drops off between 10 and 13 generations; after 14 generations the the populations remain unchanged because the ML is unable to learn further.

have as adverse an effect on convergence characteristics. However, later in this work we see that there are different results when using another training set configuration as discussed in §5.3.2 on page 141.

Fig. 4.32b hints as to why there may be so little performance degradation when the Griewangk test function was rotated. Unlike for the Rastrigin function, we can see that rotating the Griewangk has no change in the ML's learning pattern.

α modulates selection pressure in histogram-based Method

One persistent pattern that emerges with regards to the histogram-based method for sampling rule intervals is that the α parameter does modulate the selection pressure. With exception for the f_{lang} function, which will be discussed later, the higher the α, the more similar to the uniform distribution based rule interval sampling method. This is expected given that this parameter controls the amount of a uniform distribution "blended" into the

histogram-based distribution. So naturally the higher the α, the more similar the uniform and histogram-based trajectories.

For the end of the runs, there was no appreciable difference between the different α runs for the Rastrigin and Griewangk functions, both rotated and non-rotated, as well as for the Langerman function runs. However, for the f_{sphe} and f_{step} runs, the lower α runs were able to converge more closely to the global optima even at the end.

Populations unable to escape local optima in f_{lang}

As we see in Fig. 4.25 on 75 that no matter the rule interval sampling method, SLEM makes no progress after the initial generation. Essentially, the machine learner fails to learn anything after the first couple generations, and thereafter all individuals are clones. That is, the initial population was trapped in a large basin of attraction for local optima from which it was unable to escape. This is further discussed in §5.2.1 on page 108.

4.3.3 Conclusions

In this section I covered the impact of three different rule interval sampling strategies on the emergent selection pressure that arises when a machine learner is used to create offspring in an EA — i.e., that of using uniform, Gaussian, and histogram-based distributions for sampling rule intervals.

We observed in the f_{step} experiments that SLEM populations will stay within bounds dictated by the initial parent population unless a Gaussian distribution of suitably wide standard deviation is used. And, even then, that is no guarantee that the population will be able to escape that initial bounds because runs with a narrower Gaussian distribution asymptotically converged on an bounds more conservative than the one described by the initial population.

We also observed, with the exception of the aforementioned f_{step} runs, that the Gaussian distribution with three standard deviations sampling rule intervals tended to converge the most quickly, whereas the uniform distribution proceeded the most slowly. Moreover the α

parameter for the histogram-based approach allowed us to modulate the selection pressure. This is likely due to that Gaussian and histogram-based distributions taking advantage of topological features of the underlying fitness landscape as well as allowing for modest exploration beyond rule intervals.

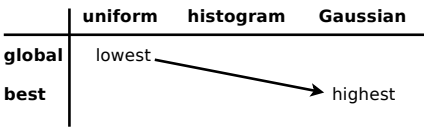
We have a mixed perspective on the effects of rotation on SLEM. For the Rastrigin function we found that when there was a difference, which was at the very beginning and at convergence, that the populations tended to converge more closely to the optima suggesting that SLEM garners some benefit from axially aligned optima for the Rastrigin, particularly when the runs have converged near the global optima. This is contrasted by the Griewangk function where there was no significant difference between the non-rotated and randomly rotated Griewangk runs.

And, finally, we saw that SLEM was unable to make any progress on the Langerman function beyond the initial population regardless of the rule interval sampling method, which suggests a large basin of attraction for local optima. This is explored further in §5.2.2.

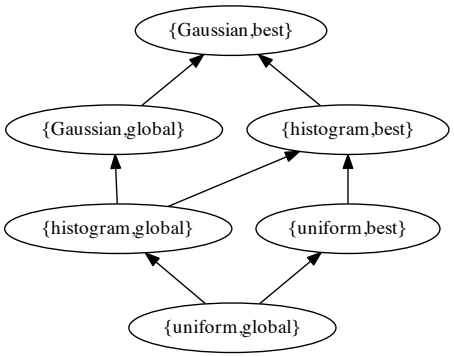
In the next chapter I look at the influence that different training set configurations have on this unique emergent form of selection pressure.

4.4 Combined Effects of Unbounded Rule Interval Strategies and Rule Interval Sampling Approaches

By now we have a sense of the relationship between different unbounded rule interval strategies and rule interval sampling methods on selection pressure, albeit independently. We know that selection pressure increases going from the *global* to *best* strategy, or *global* < *best*. Similarly, we know that selection pressure generally increases going from the uniform to histogram-based to Gaussian rule interval sampling methods, or uniform < histogram < Gaussian. Moreover, within the histogram-based approach the selection pressure is inversely proportional to α, or the amount of uniform distribution blended with the histogram-based distribution; and the selection pressure increases when going from two to three standard



(a) This shows that the selection pressure increases from *global* to *best* unbounded interval closing strategies; similarly the selection pressure increases from uniform to histogram-based to Gaussian means of sampling rule intervals. The least selection pressure uses the *global* unbounded rule interval closing strategy and a uniform distribution to sample rule intervals, and the highest selection pressure uses a Gaussian distribution to sample rule intervals and the *best* strategy for closing unbounded rule intervals.



(b) This Hasse diagram shows the poset, or partially ordered set, that represents the relationship between the closed unbounded rule interval strategies and rule interval sampling methods. One can traverse the graph from the node with the least selection pressure, {uniform,global}, to the end node with the highest selection pressure, {Gaussian,best} with the selection pressure increasing as one moves along the graph in the direction of the arrows.

Figure 4.33: This depicts the relationship of selection pressure to unbounded rule interval closing strategies and rule interval sampling methods. The selection pressure is weakest using a uniform distribution to sample rule intervals and using the *global* unbounded rule interval closing strategy, and is strongest using a Gaussian distribution and the *best* strategy. The relationship between these different configurations can be represented as a partially ordered set, as shown in Fig. 4.33b.

deviations to sample rule intervals within a Gaussian distribution.

However, a practitioner will have to make both design choices when implementing these types of EA/ML hybrids so the effects of both approaches have to be taken into consideration. Can we posit combined effects from these independent relationships? Fig. 4.33a shows a posed general relationship between the unbounded rule interval strategies and the different rule interval sampling methods with regards to the influence on selection pressure. By and large the least selection pressure occurs with a uniform rule interval sampling distribution and the *global* unbounded rule strategy and the highest using a Gaussian distribution and the *best* unbounded rule interval strategy. The intuition is that selection pressure increases to the right and down in the table.

In a sense the table describes a partial ordering of selection pressure for open interval strategy and interval sampling pairs; this partial ordering is depicted in Fig. 4.33b. So starting from {uniform, global} within that Hasse diagram to {histogram, global} increases selection pressure; and then moving from {histogram, global} to either {Gaussian,global} or {histogram, best} also increases selection pressure. The poset's antichain — subsets that are not comparable with one another — is given by {{Gaussian, global}, {histogram, best}} and {{histogram, global}, {uniform, best}}.

What evidence do we have to support this combined effects model? We already have evidence from the results in §4.2. There we observed the increase in selection pressure when going from *global* to *best* within rule interval distributions, so this covers the links for {uniform, global} → {uniform, best}, {histogram, global} → {histogram, best}, and {Gaussian, global} → {Gaussian, best}. We also learned in §4.3 where we effectively traversed from the minimal element, {uniform, global}, to the maximal element, {Gaussian, global}, along the rightmost path in Fig. 4.33b ... i.e., the path where only the *best* unbounded rule interval strategy was used. Again, we found that the relationship dictated in that graph tour held with the caveat that the standard deviation used for the Gaussian rule interval sampling method can have an impact as with f_{step} as discussed in §4.3.2 on page 78.

But what of the remaining relationships described in that graph? That is, we need corroborating evidence that the relationships {uniform, global} → {histogram, global} and‘ {histogram, global} → {Gaussian, global} are true. We will need to examine new data to support the claims that the selection pressure increases when moving along those links.

To that end, we previously observed that the test functions for the *best* related links more or less behaved the same, so we can make a reasonable assumption that the same behavior will manifest for the rest of the poset. Therefore we can pick a single representative test function to verify that the untested poset links hold and observe its behavior for the identified traversals. We do not want a function that is too difficult, which eliminates the Langerman function; nor do we want something too simple, such as the spheroid, since that

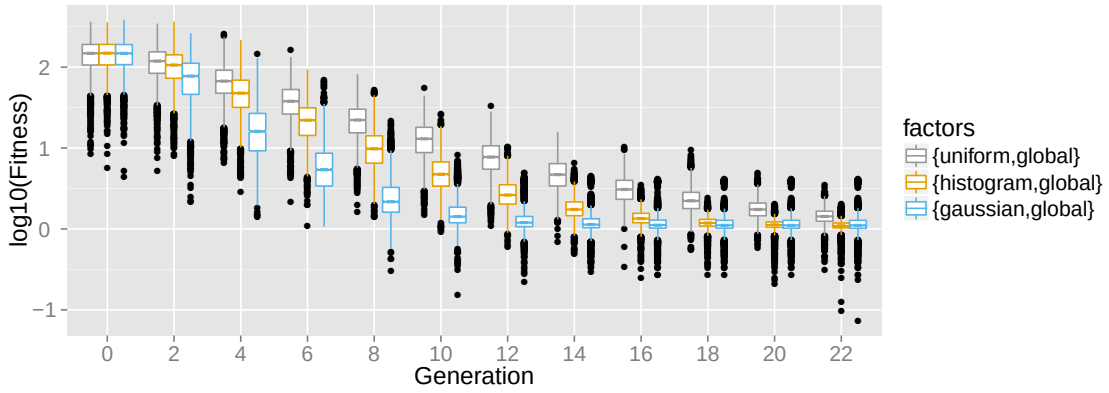


Figure 4.34: This shows the log adjusted population trajectories for {uniform,global}, {histogram,global}, and {Gaussian,global} for the f_{rot_grie} runs. The convergence rates gradually increase from uniform to histogram-based to Gaussian as predicted.

Table 4.9: One tailed pairwise Wilcoxon rank sum p-values for fitnesses of the last generation for the {uniform,global}, {histogram,global}, and {Gaussian,global} runs with Bonferonni adjustment with the hypothesis being that the median fitnesses of the row values are less than the column. Here we see that the {histogram,global} approach mostly closely converged to the global optimal of the runs, followed by {Gaussian,global}, and then {uniform,global}.

	{uniform,global}	{histogram,global}
{histogram,global}	< 0.001	
{Gaussian,global}	< 0.001	1.0

function is not very representative of real world problems. That leaves the Rastrigin and Griewangk functions for consideration. Of the two, the Griewangk is the more complex since it is not linearly separable; moreover, randomly rotating this function makes it more challenging. So for exploring the rest of the links of the graph I will use the rotated Griewangk function. Note I also use an α of 0.2 for the histogram-based runs and 3 σ for the Gaussian runs.

The respective population trajectories for {uniform, global}, {histogram, global}, and {Gaussian, global} for f_{rot_grie} are depicted in Fig. 4.34, and shows that the selection pressure does increase moving from uniform to histogram-based and to Gaussian rule interval sampling methods using the *global* unbounded rule interval closing strategy. This is corroborated by Fig. 4.35 which shows the differences in selection pressure when traversing from {uniform, global} $\rightarrow$ {histogram, global} and {histogram, global} $\rightarrow$ {Gaussian, global}

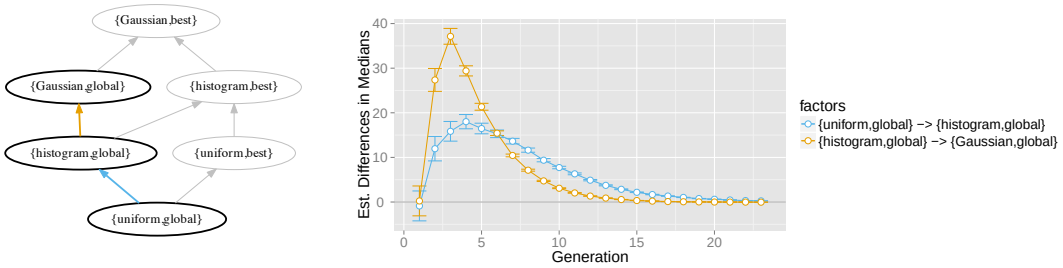


Figure 4.35: This shows the 95% confidence intervals for by generation differences between medians for the {uniform,global} $\rightarrow$ {histogram,global} and {histogram,global} $\rightarrow$ {Gaussian,global} runs, which are highlighted in the Hasse diagram on the left, using the f_{rot_grie} test function. Values above zero favor the target node with regards to selection pressure; so this shows that the selection pressure increased with each graph traversal corroborating what was seen in the previous figure.

runs using the f_{rot_grie} test function. With each graph traversal the selection pressure increased as predicted. Of interest is the detailed comparison of the last generation for all the runs, which is given in Tbl. 4.9. Here we see the uniform-based approach is not able to close in as close to the global optima as the other two approaches. And though the Gaussian-based rule interval sampling runs are able to quickly converge as compared to the other other runs, it is the histogram-based approach to is able to converge a little more closely to the global optima, even though it initially did not converge as quickly as the Gaussian-based runs.

We now have enough corroborating evidence to claim this partial ordering of selection pressure as depicted in the original Hasse diagram is valid. Moreover, the relationships described by that poset should give practitioners guidance for adjusting selection pressure by iteratively altering either the rule interval sampling method or unbounded rule closing strategy to move up or down the graph as desired.

In future work we could move from the partial ordering of rule interval sampling methods and unbounded rule interval strategies described here to a complete ordering. Additionally we could get a better sense of the relative intensities associated with each configuration for more precise guidance.

4.5 Summary

The theme of this chapter was how certain rule interval sampling design choices influenced selection pressure. There were two kinds of rule interval sampling issues explored. The first regarded mitigating instances where learned rules would specify only a single bounds; the second explored the influence of sampling the intervals with different probability distributions.

I looked at three approaches for resolving unbounded rule intervals; one approach used original initialization boundary values, another would use extrema from the current set of genes, and, lastly, another used the extrema from only the best parents' genes. I found that using the first approach was disruptive to the runs because regions of interest were reset to original initialization bounds during runs. Meanwhile I determined that setting the bounds to the global gene extrema had less selection pressure than doing so only from the best.

I also looked at how uniform, Gaussian, and a histogram-based distributions for sampling rule intervals influenced selection pressure. I found that the uniform approach had the least selection pressure, followed by the histogram-based distribution, and the Gaussian distributions had the highest. Moreover, the histogram-based rule interval sampling method could be further tailored by manipulating the amount of a uniform distribution blend. Similarly, generally the higher the standard deviation used for the Gaussian distribution to sample rule intervals, the higher the selection pressure.

From these results I derived a map for practitioners whereby selection pressure could be modulated by manipulating the unbounded rule interval closing strategy and the distribution used to sample rule intervals.

I also determined that the initial population describes an implicit boundary for exploration. I observed that populations would not explore beyond the boundary unless a Gaussian distribution with a wide enough standard deviation was used.

Additionally I found that populations were unable to escape a very large basin of attraction for local optima using the ML induced selection pressure.

Furthermore I learned that effecting a rotation slightly impaired performance for the

Rastrigin function, but had no significant effect with regards to the Greiwangk function.

In the next chapter I examine how certain training set configurations can also influence the selection pressure associated with using a machine learner to create offspring in an evolutionary algorithm.

Chapter 5: Effects of Training Set Selection Strategies

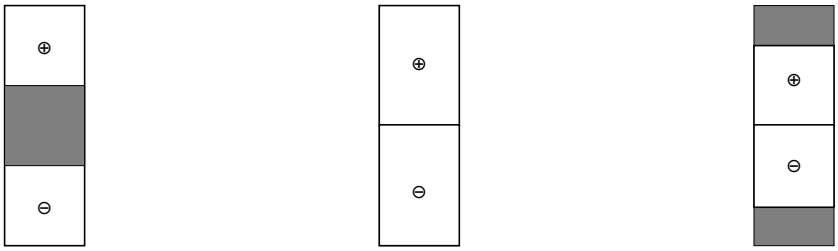
In the previous chapters we covered the effects of different rule interval sampling techniques on the emergent selection pressure associated with this rule-based EA/ML hybrid. Another important design decision a practitioner will need to make when assembling these kinds of hybrids concerns the training sets of positive and negative examples given to the machine learner.

This chapter examines the impact on the unique emergent selection pressure of two training set configuration approaches — training sets derived from sorted parent populations by the rank of parents or by percentage of the range of fitnesses. Moreover, within these two approaches the practitioner has to further decide how the training sets are allocated. That is, within the two over-arching training set strategies of by-rank or by-fitness-percentage training sets one can configure training sets by splitting the parent populations for the "best" or "worst" training sets, having a gap of mediocre individuals between the "best" and "worst", or a novel from-the-middle training set configuration approach.

5.1 Introduction

Subsection 2.1.1 on page 14 described two approaches for selecting the positive and negative examples to give the machine learner. One approach selects the best and worst individuals by rank, and the other by fitness percentage as shown in Fig. 2.1 on page 15. Another approach where the middle set of individuals was split between the best and worst individuals has also recently been explored [Wallin and Ryan, 2009].

Emergent selection pressure responds to the changes in how the training sets are selected — at least with regards to choosing training sets by fitness rank instead of proportion. In particular, out of an overall population of 50 individuals, evenly splitting the population



(a) In the *gap* training set configuration the "best" and "worst" training sets are separated by a gap of mediocre parents.

(b) In the *split* training set configuration the parents are evenly divided for the "best" and "worst" training sets.

(c) In the *middle* training set configuration the "best" and "worst" are respectively $\frac{1}{3}$ of the parents and split from the middle leaving out the extrema.

Figure 5.1: These are the three training set configurations — denoted as *gap*, *split*, and *middle*; and shown respectively as subfigures (a), (b), and (c) — used in this chapter. The "best" are denoted by $\oplus$ and the "worst" by $\ominus$.

increased the apparent selection pressure over runs where only the top- and bottom-most 5 individuals were used. So, the number of individuals in the "best" and "least" subsets of the population and how they are selected influence this emergent selection pressure [Coletti, 2012].

To further explore the influence of training set sizes on this emergent selection pressure, I will look deeper into the two strategies for selecting these subsets. That is, the one strategy that selects a static, fixed number of top and bottom individuals ranked by fitness, and the other by percentage of fitness. For example, the top five individuals can always be selected for the "best" and the bottom five for "worst" out of a population; or, for a given fitness range, say, of [0,1000], the top 80% may fall in the range [937,1000] and the bottom 20% in [0,137], so individuals with fitness values in those ranges will be assigned "best" and "worst," respectively. Again, these two approaches are described in more detail in section 2.1.1 and are depicted in Fig. 2.1 on page 15. In this chapter I will first look into selecting ML training sets by fitness rank and then by fitness percentage.

Fig. 5.1 depicts the three training set configurations used in this work and are inspired from [Wallin and Ryan, 2009]. These training set configurations are further described below:

- A *gap* training set configuration where the top and bottom 30 individuals corresponded to the "best" and "worst" training sets; the middle 30 were a "mediocre" population subset that were ignored. Note that this was the configuration used for the experiments for Ch. 4. This is depicted in Fig. 5.1a.

- A *split* training set configuration where the top 45 were the "best" and the bottom 45 the "worst." There is no mediocre set. This is shown in Fig. 5.1b.

- A *middle* training set configuration, which is a novel approach whereby the "best" and "worst" training sets are created from the middle outwards [Wallin and Ryan, 2009]. So, the top 15 parents were "mediocre" and ignored; the next 30 down ranked by fitness the "best"; the next 30 down from that the "worst"; and the remaining balance of 15 of the total of 90 parents were another set of "mediocre" that are ignored. That is, the most extreme best and worst individuals were ignored with regards to training the ML. This is depicted in Fig. 5.1c.

The *middle* training set configuration merits further discussion because the notion of discarding population extrema when creating training sets is counterintuitive. Wallin and Ryan wanted to determine the necessity of mediocre parents typical of LEM and similar EA/ML hybrids [Wallin and Ryan, 2009]. They wanted to challenge the intuition that the mediocre individuals provided a buffer between the "best" and "worst" training sets that minimized their overlap, an overlap that might impair the ML's ability to learn. However, until their research, no one had empirically determined if this was true.

They found that their *middle* implementation discovered better solutions than their equivalent of *gap* and *split*, though *middle* converged more slowly. They reasoned that by eschewing the very best parents for consideration as positive training instances that "the population is given a chance to *mature*." That is, though the absolute best parents are in the parent population, they make no contribution to learning, but via truncation survival

selection they will regardless linger in subsequent populations. However, the machine learner has the opportunity to, at some point, possibly add offspring that are superior to these previously best individuals thus "pushing them down into the H-group;" i.e., to finally be considered by the ML. In this way, the very best individuals are, in a sense, held in reserve as insurance for escaping local optima.

By including *middle* as a training set configuration, I determine not only its impact on the convergence trajectories emergent from the machine learner, but also if their hypothesis has merit. Since I use non-overlapping generations then, by definition, the very best parents are discarded with each generation and thus have no opportunity to linger and thereby potentially make a contribution. If their hypothesis is correct, then we should see eliminated the ability of the *middle* training set configuration to overtake the other configurations.

Moreover, Wallin and Ryan used a binary representation for their individuals, the machine learner Tree Augmented Naïve Bayes (TAN) [Friedman and Goldszmidt, 1996], and a number of difficult binary test problems. By contrast, I use a real-value representation, the PART machine learner, and the real-valued test problems described in §3.4 on page 34. So one of my contributions in this work will be to extend their research to new problem domains.

The next two sections will, in turn, cover training set configurations based on rank and fitness percentage.

5.2 By Rank

Again, there are two broad approaches to creating training sets of positive and negative examples for the machine learner: sorting the parent population by fitness and then taking the N top- and bottom-most individuals by rank as positive and negative training sets, or taking the top- and bottom-most individuals by percentage of the overall fitness range. In practice, the former approach, that of using fitness rank, is the most common, and is the approach explored in this section. The by-percentage approach will be covered in the next section, §5.3.

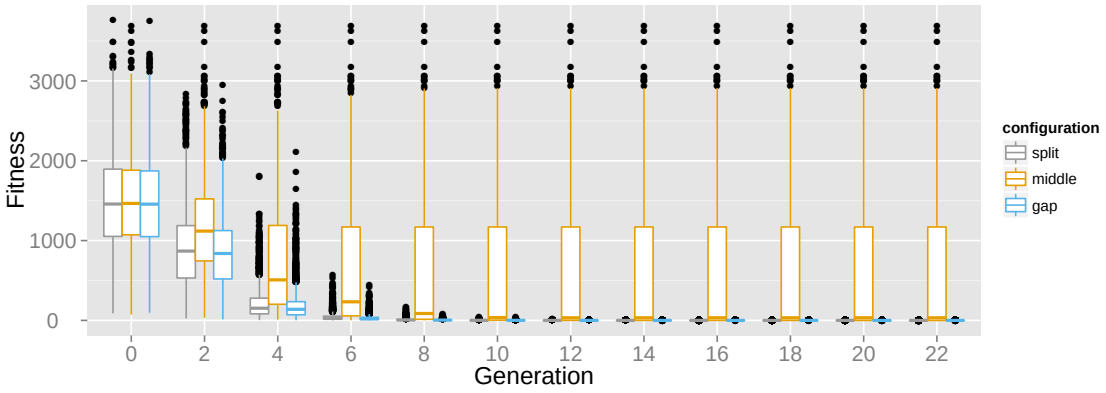


Figure 5.2: This shows the aggregate population trajectories for f_{sphe} for the three training set configurations using three σ Gaussian rule interval sampling, *split*, *gap*, and *middle*. The *middle* slowly converges for several generations before progress stagnates. By contrast the *split* and *gap* runs readily converge towards the global optimum.

In previous research, we learned that adjusting the training set sizes did have an influence on the emergent selection pressure [Coletti, 2012]. This research used a rank-based approach for training sets that included *gap* and *split* training set configurations, but not *middle*. Moreover, only the effects against the spheroid function, f_{sphe}, were explored. Here, I will not only consider *gap* and *split* but also *middle*, and the other fitness landscapes described in §3.3 on page 27.

5.2.1 Results

Here I will first share the results by the various fitness landscapes described in §3.4 on page 34 using the by-fitness-rank approach for training set configurations.

Spheroid

Though the machine learner generated emergent selection pressure f_{sphe} has been explored in prior work [Coletti, 2012], it was not done so with the *middle* configuration. Fig. 5.2 shows the combined population trajectories for f_{sphe} for the *gap* and *split* training set configurations that also includes *middle*. There we can see that the new training configuration, *middle*, stagnates a fair distance from the optima. By contrast, the *gap* and *split*

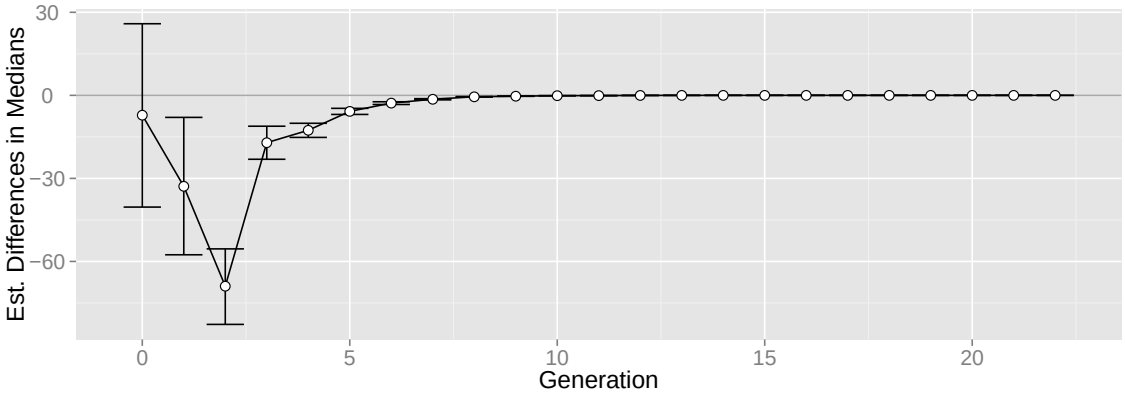


Figure 5.3: Comparison with 95% confidence intervals by generation between the *gap* and *split* runs for f_{sphe}. Values below zero favor *gap*, and those above *split*. So this shows that the *gap* converges more quickly than *split* for the first several generations.

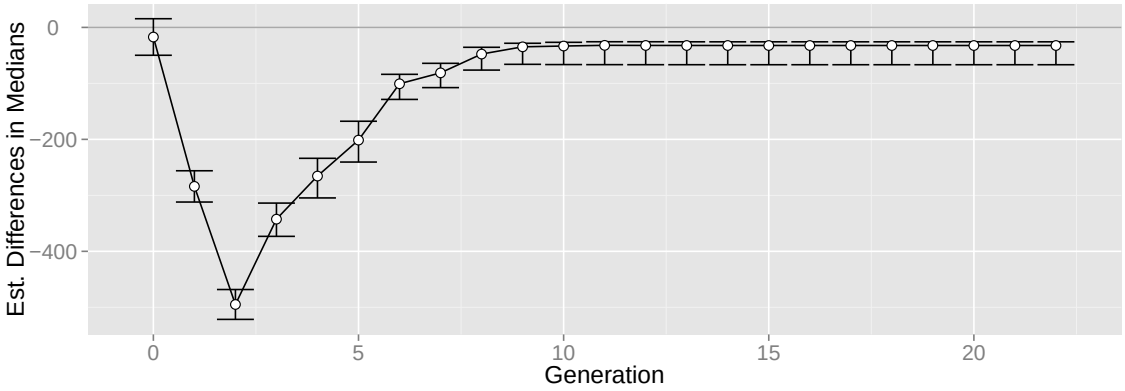


Figure 5.4: Comparison with 95% confidence intervals between *gap* and *middle* for f_{sphe} using a Gaussian rule interval sampling strategy with three σ spans. Values below zero favor *gap* and those above *middle*. The *gap* converges more quickly for the first several generations and then gradually annealing to about 40 as both systems converge.

configurations readily converge towards the global optima.

After the run has converged the differences between the *gap* and *split* for the f_{sphe} runs are a little difficult to discern. However, we can use a Wilcoxon rank sum test between the runs for the last generation, generation 22, for which we get an estimated differences in medians between the runs in the interval [0.00010, 0.00023] with a p value of less than 0.001 and with positive values favoring *split*. So, at the very end of the runs the *split* does slightly better than *gap*.

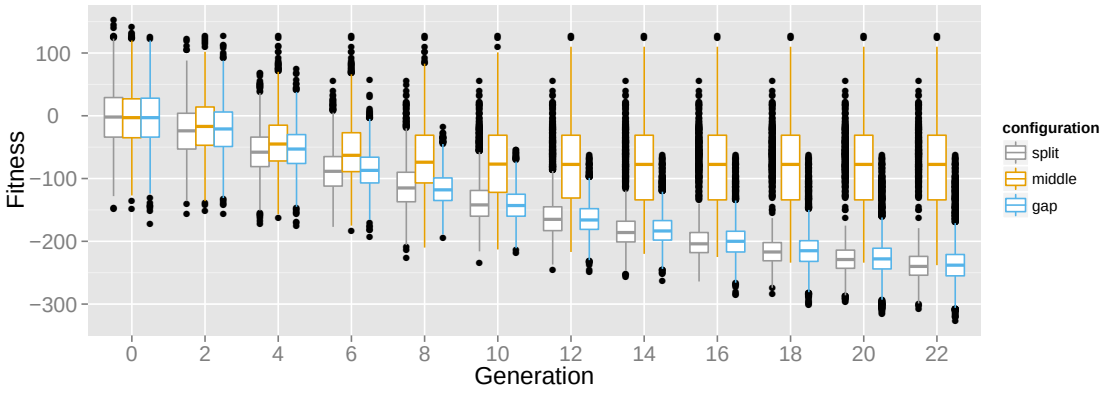


Figure 5.5: This shows the aggregate population trajectories for f_{step} for the three training set configurations, *split*, *gap*, and *middle* using a **two** σ Gaussian rule interval sampling. The *middle* slowly converges for several generations before progress stagnates. By contrast the *split* and *gap* runs readily converge towards the global optimum.

Fig. 5.4 depicts the trajectory differences between the *gap* and *middle* runs for f_{sphe}. This shows *gap* population trajectories consistently being closer to the global optima with the widest differences occurring during the first several generations.

Step

We learned in §4.3.2 that for f_{step} the populations will tend to stay within implicit bounds defined by the initial parent population. However it is possible for population trajectories to explore beyond these bounds if the rule interval sampling strategy allows for stochastically generated values beyond them as happens with a Gaussian distribution. Even then, if the Gaussian distribution is too narrow, then it will not only not explore beyond the initially defined boundaries but can have implicit bounds that are even more conservative.

We also learned that for f_{step} that the Gaussian rule interval sampling strategy that used two standard deviations to sample the rule intervals was sufficient to explore beyond the initially defined boundaries. But what of the effect on this rule interval sampling strategy of different training set configurations? The aggregate population trajectories of the three training sets strategies on using a Gaussian distribution of two standard deviations for f_{step} is shown in Fig. 5.5. There we can see that the *middle* converges for a few generations

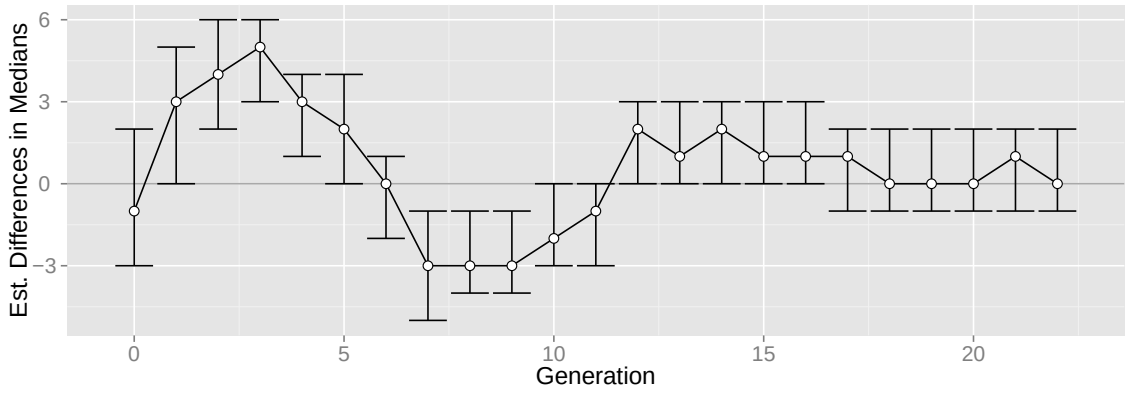


Figure 5.6: Comparison with 95% confidence intervals by generation between the *gap* and *split* runs for f_{step}. Values below zero favor *gap*, and those above *split*. The *split* converges more quickly for the first several generations, then *gap*, then switches back to *split* before ending on several generations where there is no significant difference between the two.

before it stagnates as with f_{sphe}; and, more importantly, does not exceed the implicit bounds defined at -210 even with the wider sampling area compared to the other rule interval sampling strategies. By contrast, both the *gap* and *split* manage to explore beyond these bounds, though it is difficult to discern convergence differences.

Fig. 5.6 shows the differences between the *gap* and *split* runs for f_{step}. There we can see that the *split* converges more quickly than *gap* for the first several generations, then *split* for several more, then back to *gap* again. There is no significant difference between the two training set training set strategy over another configurations for the last several generations. (Wilcoxon rank sum test for last generation p-value = 0.6005.) This phenomenon is discussed further in §5.2.2 on page 109.

Fig. 5.7 shows the differences between the *gap* and *middle* runs for f_{step}, and we can observe that the *gap* runs consistently pull away through the length of the runs.

Rastrigin

The Rastrigin function, as previously described in §3.3.3, is a multi-modal, linearly separable function with regularly spaced modalities aligned along the axes inside a quadratic bowl. The challenge for SLEM is to avoid getting trapped within any one of the many local optima

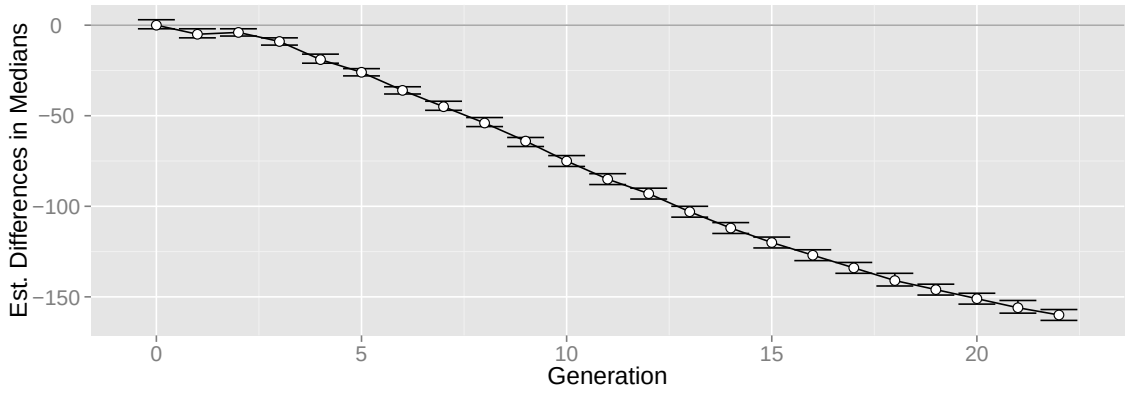


Figure 5.7: This shows the differences by generation between the *gap* and *middle* runs for f_{step}. Values below zero favor *gap*, whereas those above favor *middle*. The *gap* runs steadily pull away from *middle* for the entire run duration.

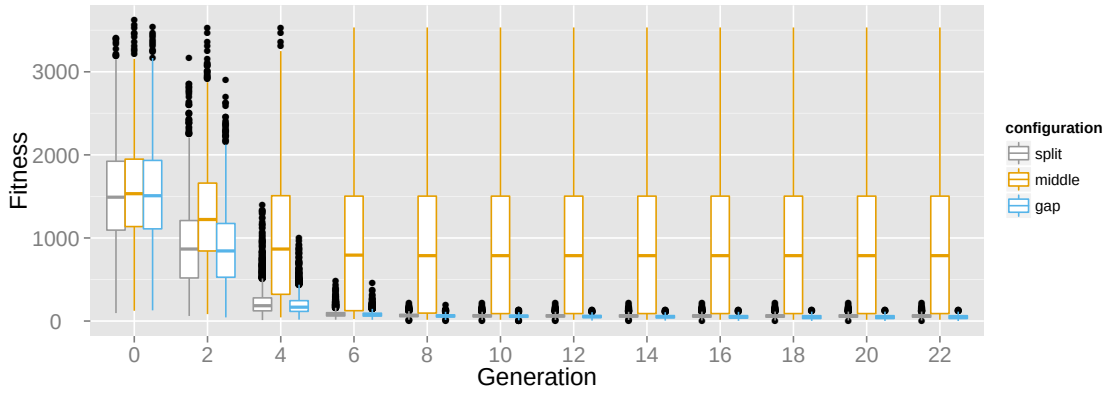


Figure 5.8: This shows the aggregate population trajectories for f_{rast} for the three training set configurations, *split*, *gap*, and *middle* using a three σ Gaussian rule interval sampling. Once again the *middle* populations converge for several generations before stagnating, while the *gap* and *split* configurations readily converge towards to global optimum.

as it converges towards the global optimum located at the origin.

Fig. 5.8 shows the aggregate population trajectories for f_{rast} for all three training set configurations. It is essentially a repeat of what have seen so far: that the *middle* stagnates after so many generations while the *gap* and *split* both readily converge towards the global optima. And, once again, the differences between the latter two are slight enough to be difficult to discern.

Fig. 5.9 shows the differences by generation between the *gap* and *split* runs for f_{rast}. This shows that the *gap* runs consistently converged more quickly and more closely to the

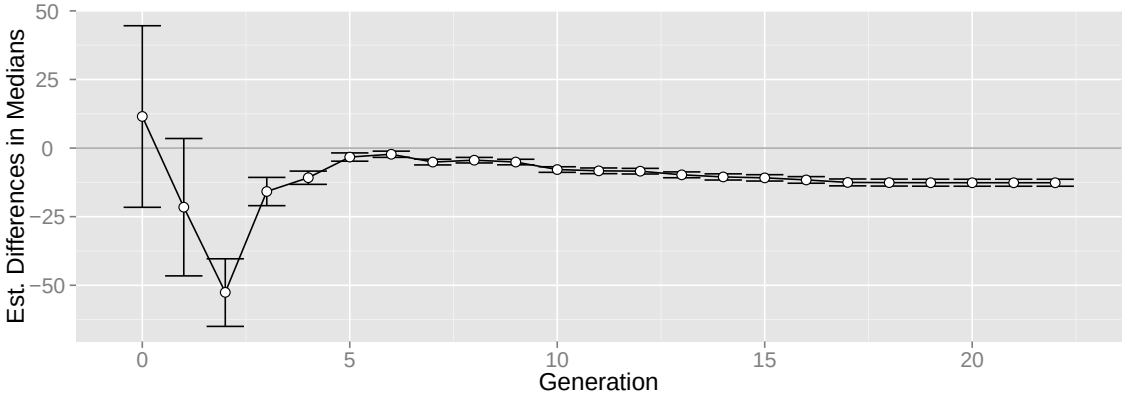


Figure 5.9: Comparison with 95% confidence intervals by generation between the *gap* and *split* runs for f_{rast}. Values below zero favor *gap*, and those above *split*. Except for the initial two generations where there is no difference between the two training set configurations, the *gap* configurations converge more quickly and closer to the global optimum than the *split* configuration.

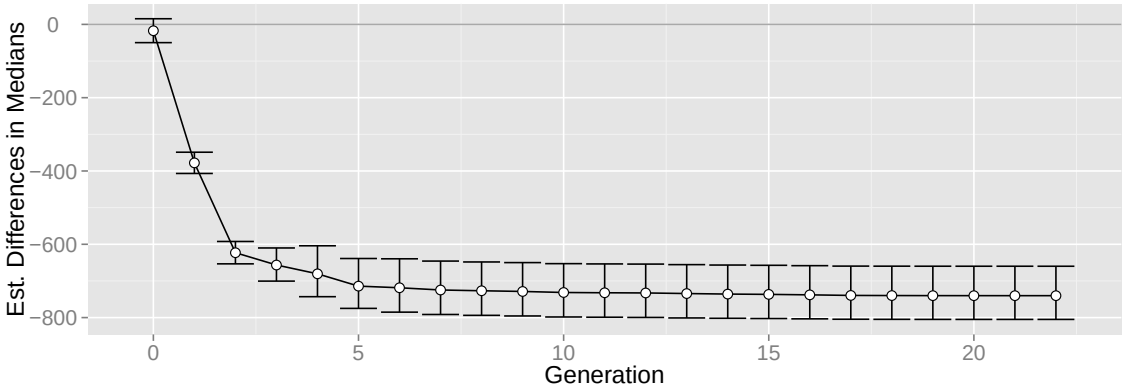


Figure 5.10: This shows by generation comparisons between the f_{rast} *gap* and *middle* runs. Negative values favor *gap* whereas positive values favor *middle*. This shows that the *gap* trajectories move closer to the global optimal than *middle*.

global optimum than the *split* runs, and that this gap gradually widens at the end of runs. For the last generation, the *gap* training set configuration is significantly closer to the global optimum than the *split* (Wilcoxon rank sum test $p < 0.001$.)

Fig. 5.10 shows the comparison between the *gap* and *middle* runs for f_{rast}. Again, we can see that the *gap* runs are consistently closer to the global optimum than *middle*.

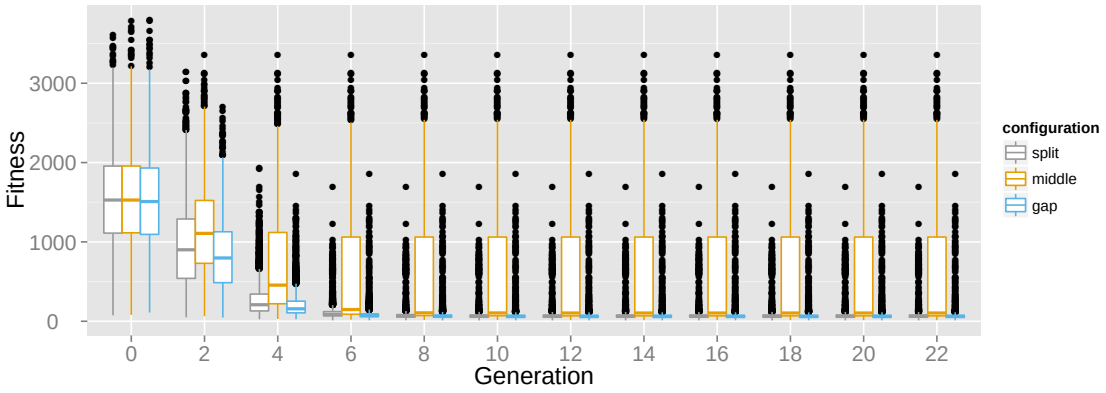


Figure 5.11: This shows the aggregate population trajectories for f_{rot_rast} for the three training set configurations, *split*, *gap*, and *middle* using a three σ Gaussian rule interval sampling. Once again the *middle* populations converge for several generations before stagnating, while the *gap* and *split* configurations readily converge towards to global optimum.

Rotated Rastrigin

As described in §3.3.4, some algorithms gain some advantage from axially aligned optima, an advantage that can be foiled by rotation[Salomon, 1996], and the Rastrigin function is just one of those functions. Fig. 5.11 shows the population trajectories for all the f_{rot_rast} runs, which is f_{rast} with a random rotation, using a three σ Gaussian rule interval sampling for all three training set configurations. We can see that the *middle* has converged after several generations and then stagnates whereas the *gap* and *split* training set configurations continue to converge towards the global optima.

However, as is the case with the previous sets of runs, it is difficult in Fig. 5.11 to discern differences between the *gap* and *split* training set configuration runs. So, once again we turn to a by-generation comparison between the two runs, which is shown in Fig. 5.12, and where we can observe that the *gap* runs are consistently closer to the global optimum than the *split* through to the last generation (Wilcoxon $p < 0.001$).

Fig. 5.13 shows the comparisons by generation between the *gap* and *middle* runs for f_{rot_rast} where once again we see that the *middle* runs are converge further from the global optimum than the *gap* runs.

Earlier, in Fig. 4.28 on page 80, we observed that randomly rotating Rastrigin has some

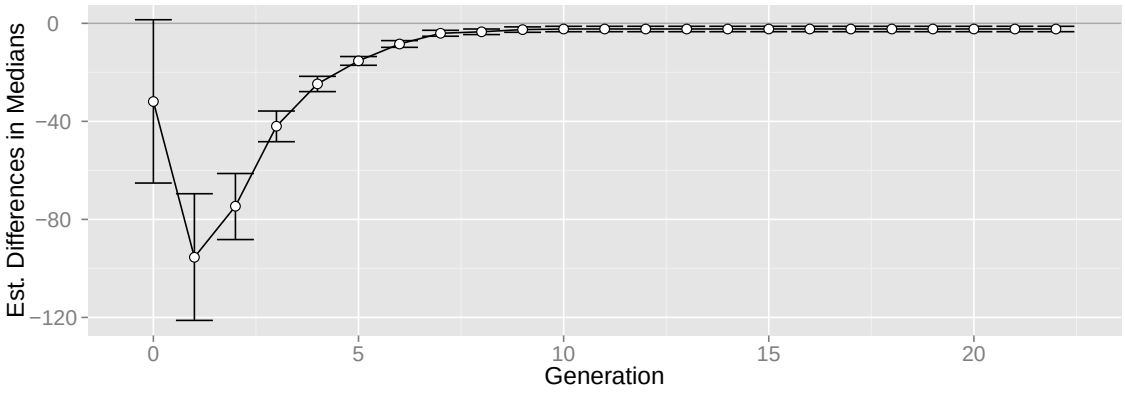


Figure 5.12: This shows the by generation differences between the *gap* and *split* for the f_{rot_rast} runs. Values below zero favor *gap*, those above *split*. We can observe here that the *gap* runs converge more closely to the global optimum than *split*.

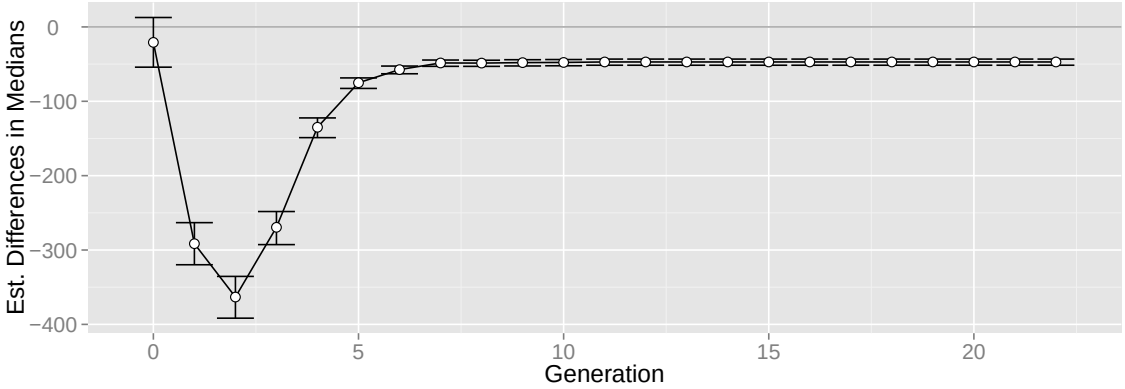


Figure 5.13: This shows the by generation difference between the *gap* and *middle* f_{rot_rast} runs. Negative values favor *gap*, whereas positive values favor *middle*. Here we can see that the *gap* runs consistently converge more closely to the global optimum.

impact on performance. However, that figure showed the results for various rule interval sampling strategies; the only training set configuration used was *gap*. What of the effects of the different training set strategies? Fig. 5.14 shows the impact of randomly rotating Rastrigin using the *split* training set configuration. We can see that the trajectories are similar to those we saw earlier in Fig. 4.28; for the last generation the non-rotated runs are closer to the global optima (Wilcoxon $p < 0.001$). That is, SLEM does appear to take some slight advantage of the axially aligned fitness landscape of the unrotated Rastrigin, though once the system has converged the differences between the non-rotated and rotated

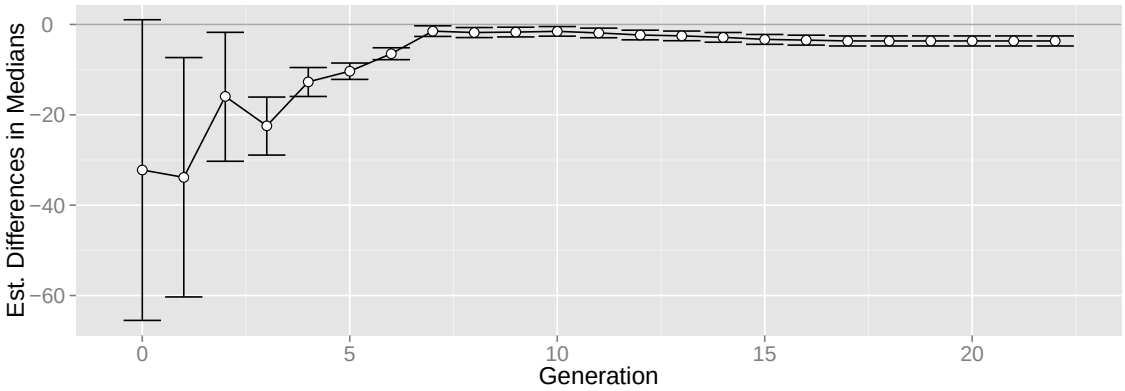


Figure 5.14: This shows the by-generation differences between the non-rotated and randomly rotated Rastrigin runs use a *split* training set configuration. Values above zero favor f_{rot_rast} while those below favor f_{rast}. The f_{rast} runs converge more closely to the global optimum for all but the first generation with the most significant differences within the first several generations.

are, again, small though still statistically significant.

Griewangk

As with the Rastrigin test function (f_{rast}), the Greiwangk (f_{grie}) is comprised of regularly space minima that are aligned along the axes. Unlike the Rastrigin, the Griewangk is not linearly separable. And so as with the Rastrigin function there is ample opportunity for SLEM to get trapped in inferior local minima.

Fig. 5.15 shows the aggregate population trajectories for the f_{grie} runs by the three training set strategies of *split*, *gap*, and *middle*. There we can see that, once again, the *middle* exhibits the same pattern of converging after several generations a comparative fair distance from the global optima, whereas the *gap* and *split* training set runs converge much more closely.

To get a better perspective on the differences between the *gap* and *split* for f_{grie} we can look at Fig. 5.16 that shows the differences by generation between the those runs. We can see that the *gap* start out significantly closer to the global optimum than *split*, and this difference gradually anneals to the point that they are statistically identical. Though not visible that figure, in the last generation the *gap* configuration is closer to the global

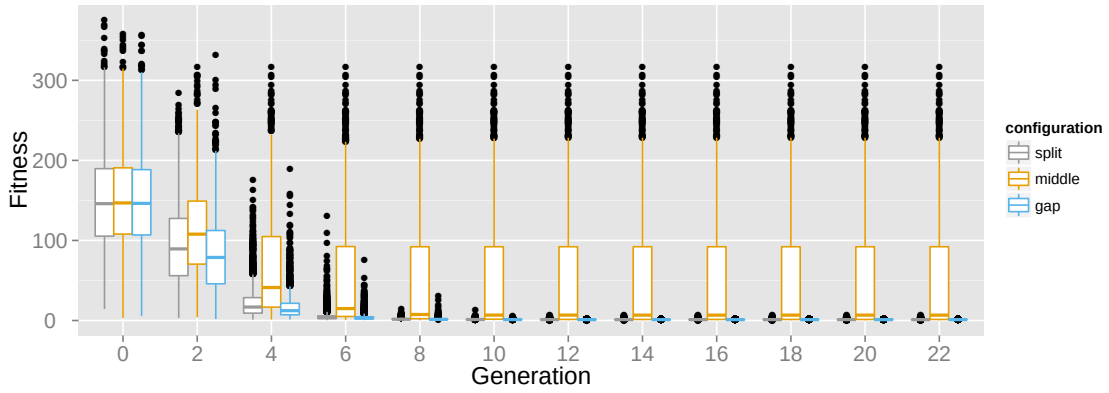


Figure 5.15: Population trajectories for f_{grie} for the three training set configurations of *split*, *gap*, and *middle*. Continuing the established pattern, *middle* converges slowly after several generations and then stagnates while the *split* and *gap* readily converge towards the global optimum.

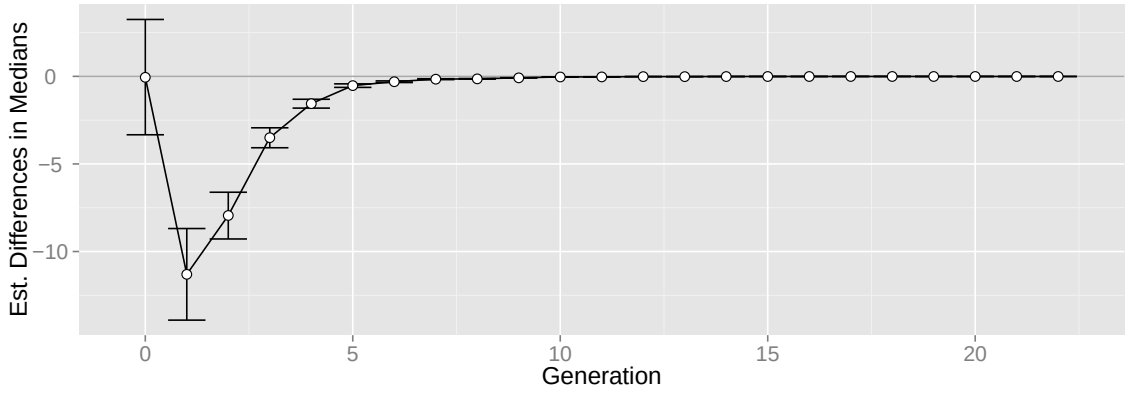


Figure 5.16: Comparison by generation between the *gap* and *split* runs for f_{grie}. Values below zero favor *gap*, and those above *split*. We can see that the *gap* starts out significantly closer to the global optimum than *split*, but that the difference between the two runs gradually anneals to the point where they are no longer statistically distinguishable.

optimum (Wilcoxon $p < 0.001$).

Though visually we observe that the *middle* runs are generally further from the global optima from the previous plots, we get statistically significant confirmation in Fig. 5.17, which shows the differences between the *gap* and *middle* f_{grie} runs by-generation. We confirm that, once again, the *middle* are further from the global optima than the *gap* for all the runs.

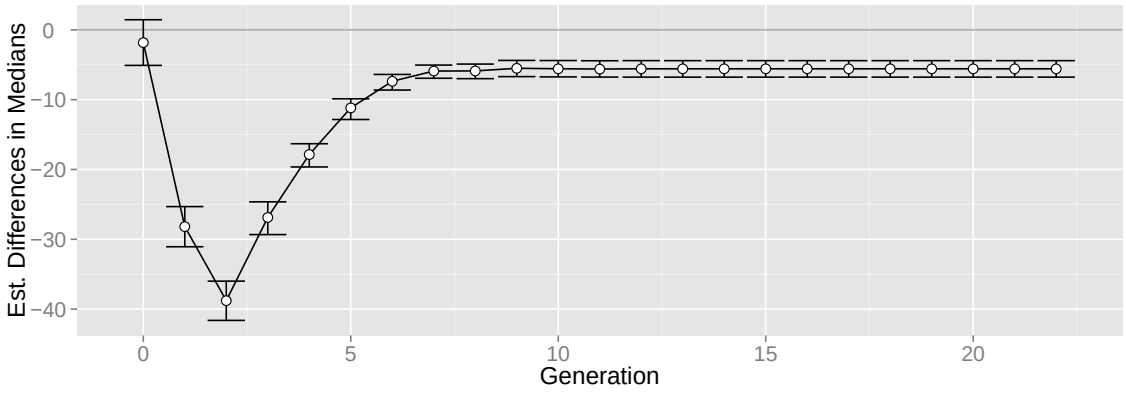


Figure 5.17: Comparison of the *gap* and *middle* runs for f_{grie} by generation. Values below zero favor *gap*, whereas those above favor *middle*. Shown is that the *gap* runs are closer to the global optimum for all runs.

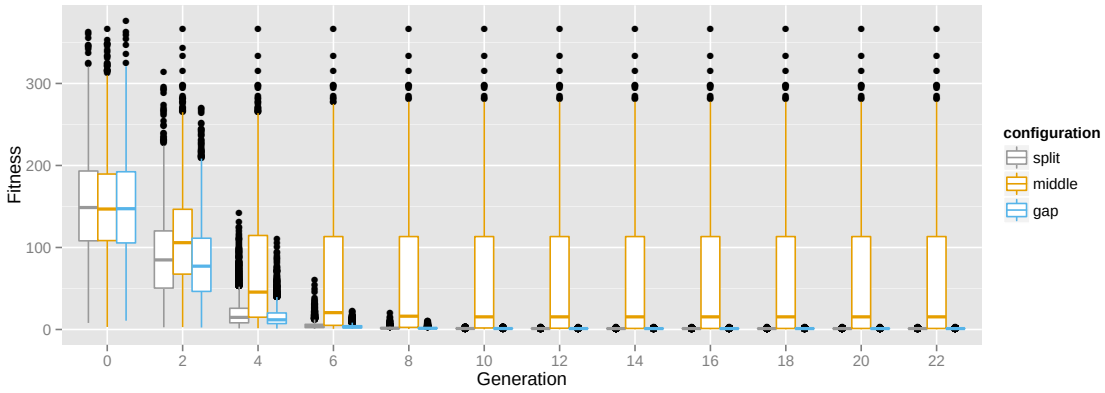


Figure 5.18: Population trajectories for f_{rot_grie} for the three training set configurations of *split*, *gap*, and *middle*. Continuing the established pattern, *middle* converges slowly after several generations and then stagnates while the *split* and *gap* readily converge towards the global optimum.

Rotated Griewangk

As with the Rastrigin function, the Griewangk can be similarly rotated to foil algorithms that can take advantage of axially aligned structures in the fitness landscape. Fig. 5.18 shows the population trajectories for f_{rot_grie}, which is, again, f_{grie} with a random rotation, for the three training set configurations. This plays out by now an established pattern: the *middle* runs slowly converge for several generations before ceasing forward progress, whereas the *gap* and *split* runs continue apace towards the global optimum.

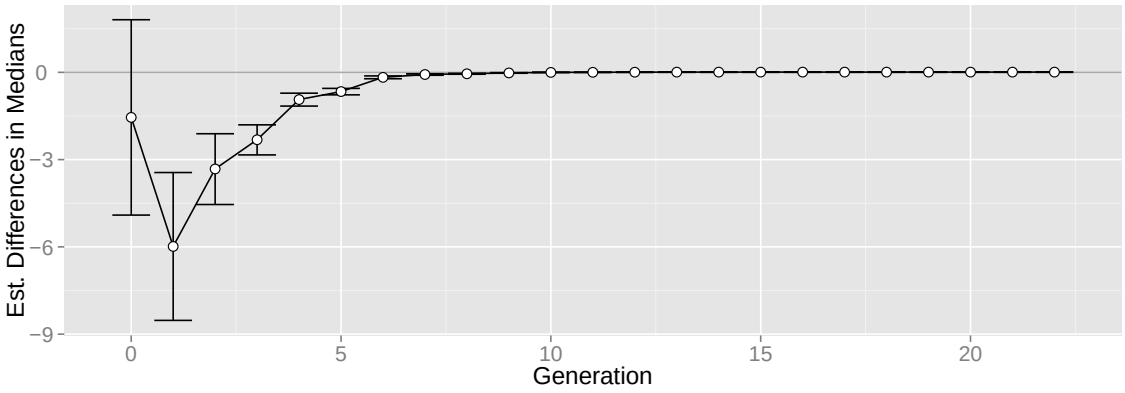


Figure 5.19: This shows the by-generation estimated differences in medians between the *gap* and *split* training set configurations for f_{rot_grie}.

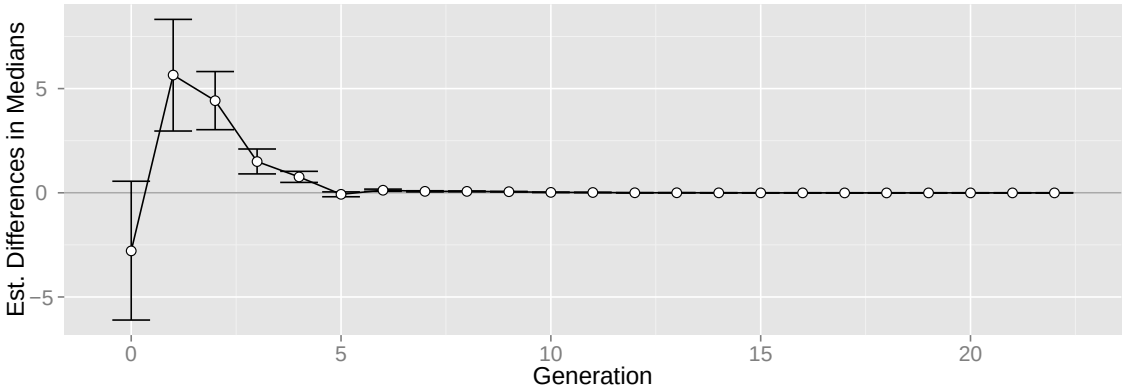


Figure 5.20: This shows by generation differences with 95% confidence intervals between the non-rotated and randomly rotated Griewangk runs use a *split* training set configuration. Values above zero favor f_{rot_grie} while those below favor f_{grie}. For the first several generations beyond the initial the randomly rotated Griewangk *split* runs converge more closely to the global optimum; but there is no difference once the runs have converged.

Since, once again, it may be difficult to discern the differences between the *gap* and *split* configuration runs, Fig. 5.19 shows the by-generation differences between them for f_{rot_grie}. As with the previous test functions, this shows that the *split* population trajectories are further from the global optimal than *gap* which remains true through to the final generation (Wilcoxon p = 0.03393).

For the *gap* runs we saw that there was very little difference between the rotated and non-rotated Griewangk runs, as shown before in Fig. 4.30 on page 82. However, when

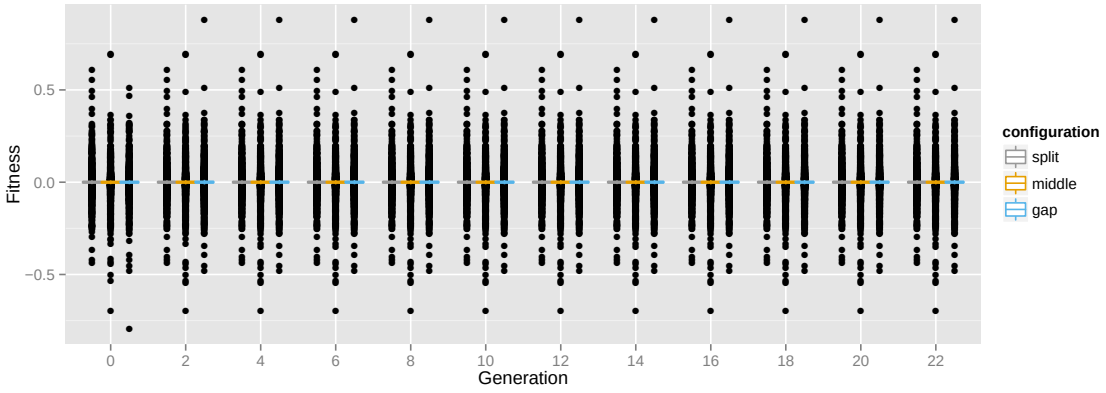


Figure 5.21: Box and whisker plot showing the population trajectories for f_{lang} for the three training set configurations of *split*, *gap*, and *middle*. Note that the inter-quartile distance is so small that the first and third ranges over plot on the median, which means most of the population fitnesses are confined to a very narrow band. All the generations beyond the first are clones of one another since the machine learner was unable to learn anything.

we use the *split*, we get slightly different behavior, which is depicted in 5.20. Here we see that for the first several generations the randomly rotated Griewangk runs converge more closely to the global optimum than their non-rotated counterparts. This is discussed in more detail in §5.2.2 on page 116. However, there is no discernible difference between the rotated and non-rotated Griewangk runs using the *split* through to the last generation (Wilcoxon p-value = 0.339).

Langerman

The Langerman function is a very challenging problem. It is multi-modal, not linearly separable, and has randomly placed minima. It also has large basins of attraction for inferior minima to all the better trap optimizers, which we have already witnessed in previous chapters. Indeed Fig. 5.21 shows identical population trajectories by training set configuration for f_{lang} to those we have seen for different rule interval sampling strategies, as found in Fig. 4.25 on page 75. That is, there is little progress made by from the initial population forward, which suggests a large basin of attraction for the fitness value 0 when we know that the global optimum fitness is below -1.

This is covered in more detail in §5.2.2 on page 112.

5.2.2 Discussion

In this section I discuss the results for the by-fitness-rank runs.

gap vs. *split* training set configurations for by-fitness-rank

With the exception of f_{sphe}, f_{step}, and f_{lang}, which I will address shortly, the by-generation run comparisons indicated that the *gap* training set configuration was a little greedier than the *split* particularly at the beginning of runs. That is, the by-generation estimated differences in medians between runs shown in Figs. 5.9 (Rastrigin), 5.12 (Rotated Rastrigin), 5.16 (Griewangk), and 5.19 (Rotated Griewangk) consistently show for all generations that the *gap* runs were closer to the global optimum than *split*; though, again, at the end of the runs this difference was very small.

However, I must caution that using the medians as a representative statistic but say little about associated distributions. Figs. 5.8 (Rastrigin), 5.11 (Rotated Rastrigin), 5.15 (Griewangk), and 5.18 (Rotated Griewangk) shows the associated population trajectories for those runs, and we can observe from them that there is a great deal of overlap between the *gap* and *split* respective distributions. Moreover, the differences between the run medians are generally small when compared to the overall fitness values. So I would warn against reading too much into these differences, though they still provide some insight into the underlying behavior.

Though there is still a great deal of over lap between the two sets of runs, the significant differences in their respective medians tell us that there is at least a little more greediness with the *gap* approach over *split*. And this would almost certainly be due to the reason why implementors have traditionally used a *gap* training set configuration: that a gap of "mediocre" parents separating the "best" and "worst" training sets minimizes to an extent the overlap between the two. Thus the ML gets a clearer perspective on the underlying fitness landscape and so is better able to explore it.

The experiments for the f_{sphe}, f_{step}, and f_{lang} functions merit further discussion because their results are different for the other fitness landscapes.

The results for f_{sphe} are actually similar to those for the Rastrigin and Griewangk functions in that, as we see in Fig. 5.3 on page 97, that the start of the runs do favor *gap* training set configuration over *split* as happened with the Rastrigin and Greiwangk runs. However, the difference is that after the population has converged near the global optima at the origin that it is the *split* that is able to converge closer yet, albeit for very small amounts. What is happening is that the population is so homogeneous that the larger size of the *split* gives the ML just enough additional information to converge a little further towards the optimum than the *gap* configuration.

For the f_{step} we observed seesawing between *gap* and *split* training set configurations as shown in Fig. 5.6 on page 99. This interleaving pattern of dominance between the two training configurations may be a reflection of which side of the implicit barrier at -210 the population is mostly located at a given generation. The *split* converges more quickly to the global optimum as the population approaches the barrier, then *gap* converges more quickly as the population pushes through, and then *split* is better for a few generations once the population is outside the boundary.

f_{lang} is further discussed in §5.2.2 on page 112. There are also some rotation related effects between *gap* and *split* which are further discussed in §5.2.2 on page 116.

Testing exploration enhancement effects of the *middle* training set configuration

Wallin and Ryan's research was motivated by determining the necessity of "mediocre" training sets in LEM-like systems. Naturally the "baseline" experiment was a *gap* configuration, which was compared to a set of experiments that eliminated this mediocre set by evenly dividing the set of parents into "best" and "worst" training instances; I refer to this approach in this work as the *split* training set configuration. However, the *split* had one third more individuals, or training instances, than the *gap* configuration, which was likely to affect what the ML learned. So they created a third set of experiments where they took the *split* configuration and trimmed the extrema from the positive and negative training sets such that the "best" and "worst" were the same sizes as the equivalent training sets in the *gap*

configuration thereby making a fairer comparison between the two [Wallin and Ryan, 2009].

They discovered that the *middle* configuration for many of their test problems converged on superior answers than their counterparts. They reasoned that this was likely because the highest performing parents were not visible to the ML, but would always survive to the subsequent generation because truncation survival selection was used and they were the fittest individuals. That is, with truncation survival selection the top N individuals in the current population of parents and their offspring would be selected to be the potential parents in the next generation. However, the ML would not "see" these fittest parents because, again, the very best and the very worst extrema were ignored when creating the training sets of positive and negative examples; so these fittest parents would make no contribution to the rules that were used to guide offspring creation. In the event that offspring were created with fitnesses higher than these very fittest parents, then those offspring would then be added to the set of the absolute fittest parents pushing down some of the previously absolutely fittest parents into the new "best" training set; i.e., these old fittest parents would now become visible to the ML and thus contribute to the learning process and, subsequently, to offspring creation. In a sense, these fittest parents are a hidden safety feature that allows the algorithm to potentially escape local optima.

However, this was an untested hypothesis, but one that could be tested here. That is, by suppressing other sources of selection pressure by having non-overlapping generations those absolute fittest parents that were invisible to the ML would be discarded every generation. If their hypothesis was correct, this would mean that in none of my test problems the *middle* configuration would converge to better optima than the *gap* or *split* configurations. And we can see in the previous set of results that was indeed the case, which lends credence to their argument.

Unfortunately there are some threats to validity. First, they used a binary representation instead of real-valued one, a different machine learner, and a different set of test problems. To confirm these results, their original set of experiments would need to be re-run with one small change: they, too, would need to employ non-overlapping generations. My intuition,

Table 5.1: This shows the results of a uniform sample of [0,10] for a step size of one for f_{lang}, dividing the fitnesses into five evenly valued bins, and then counting the number of occurrences of fitnesses within those ranges. This indicates that much of the search space is associated with the fitness range (-0.308,0.061] with comparatively few instances in the bin that contains the global optima, (-1.05,-0.678], which is an indication of the difficulty of the Langerman function.

Fitness Ranges	Count
(-1.05,-0.678]	13
(-0.678,-0.308]	410
(-0.308,0.061]	154651
(0.061,0.43]	5829
(0.43,0.8]	148

though, is that the results of such experiments should provide further evidence buttressing their original hypothesis.

Basins of attraction scale to training set sizes

In §5.2.1 we observed that all three training set strategies failed to make any progress on f_{lang}, which suggests that most of the initial population was seeded in inferior local optima with large basins of attraction. We get a sense for this basin of attraction when observing fitnesses for regularly spaced samples of the Langerman function. I sampled f_{lang} with the samples spaced an even distance of one apart in [0,10], which is the same domain I used for the experiments, to get an idea of the distribution of the fitness values; the results of this sampling are shown in Tbl. 5.1. In the table I arbitrarily split the fitness counts into five equally sized ranges where we can see that 96% of the fitnesses are in (-0.308,0.061]. By contrast, the global optimum is located in the first bin, which had just 13 samples out of the total of 161,151. Clearly almost the entire initial parents will be contained within (-0.308,0.061], and evidence from prior runs shows that subsequent generations cannot escape that region.

There are a number of remediation schemes that can be employed when an EA is unable to make significant progress. One could relax the selection pressure to allow for more exploration; but I have intentionally removed other sources of selection pressure to focus on the effects of the ML, so I do not consider that option here. Similarly I do not

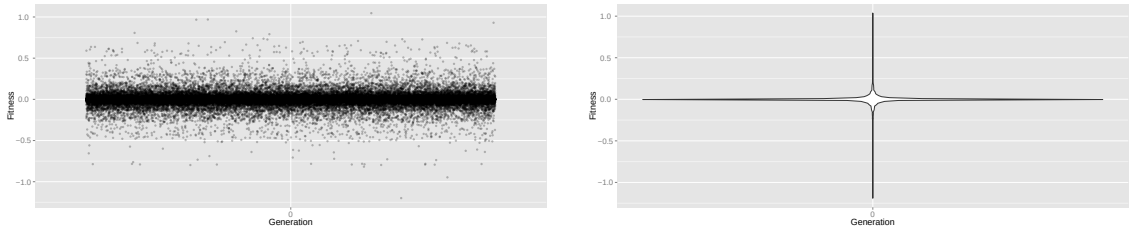
consider increasing mutation rates or adding in crossover to aid exploration since, again, I want to concentrate on the effects of the ML, which is the sole source of novelty for this work. However, another way to possibly improve the EA's performance is to increase the population size to allow for better search space exploration.

$$m \geq \frac{1}{\epsilon}\left(4\log_2\left(\frac{2}{\delta}\right) + 8VC(H)\log_2\left(\frac{13}{\epsilon}\right)\right) \tag{5.1}$$

The training set sizes I chose were based on PAC theory [Mitchell, 1997]; specifically, from Eq. 3.8 on page 37, the estimated training set lower bound for two concept problems with 0.05% error and 95% confidence was 60 instances, which I have used so far. With that in mind, let us look at a different PAC equation, Eq. 5.1, which is based on the hypothesis space and not the concept space. Eq. 5.1 gives the number of randomly drawn samples, m, needed to learn the target concept — in this case the fittest gene values — in the hypothesis space, H, with probability (1 - δ) before we ϵ-exhaust the version space. So, given an H of 5, an ϵ of 0.05 and a δ of 0.05, that means m is greater than or equal to $\ulcorner 6843.648 \urcorner$, or 6844, samples. Might the ML have better traction on f_{lang} with this population size?

Fig. 5.22 shows the results of increasing the population sizes for f_{lang}. The first subfigure, Fig. 5.22a, shows a scatter plot of all the populations from the first generation. I have compensated for over plotting by randomly "jittering" points horizontally; I also used alpha blending so that the points are partially translucent such that any points below a given point are partly visible — i.e., point density is in proportion to the darkness of a given point on the scatter plot. Still, there is a tremendous amount of over plotting making it difficult to discern the true population density. I use a violin plot to show the population density, which is depicted in Fig. 5.22b; there we can observe that most of the fitness values are tightly clustered around zero. Fig. 5.23 shows that the machine learner was unable to learn anything for the larger populations for f_{lang} for any of the runs.

We have already seen that with homogeneous populations characteristic of the end of runs that the positive and negative training sets are too similar for the machine learner to



(a) This is a scatter plot for the initial generation for f_{lang}. Jitter and alpha blending is used to overcome over plotting. Though most of the population is tightly clustered around fitness values of zero, there is some banding at other fitness values that suggests other local optima.

(b) This is a violin lot of the same population that shows the population distribution. The vertical spars show the range of values whereas the horizontal shows population density. From this we can observe that the fitness values are tightly clustered near zero.

Figure 5.22: These figures show the population of the first generation for 20 runs where the population size was increased from 60 to 6844 for f_{lang} for *gap* runs. Since the machine learner learned nothing for all the runs this population distribution is unchanged for all subsequent generations since the ML is the sole source of gene perturbation; so this effectively shows the result of randomly sampling the f_{lang} space with 136,880 points (6844 $\times$ 20). We can see that the majority of points is clustered near the origin, though the banding effects shown in 5.22a suggest other local optima.

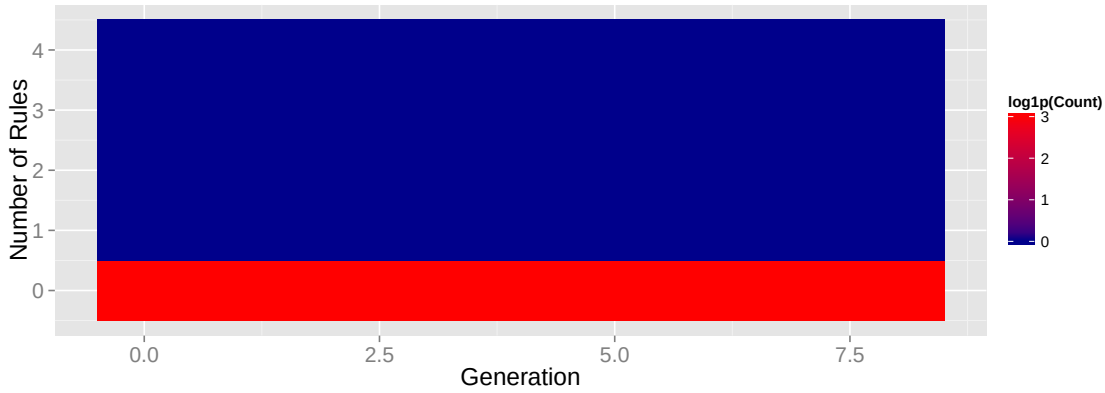


Figure 5.23: This is a heat map with a log adjusted scale that shows by generation how many rules were learned for f_{lang} for a larger parent population of 6844. We can see here that the machine learner was unable to learn any rules for all generations and runs.

infer anything useful. The same sort of thing is happening here: With f_{lang} most of any given initial set of parents will be in that one large minima with a fitness at or near zero; this means that there is no significant difference between the "best" and "worst" parents such that nothing can be learned. Even if one is lucky and manages to find one or two parents near the global optima in the initial set of parents, it may as well be noise by the

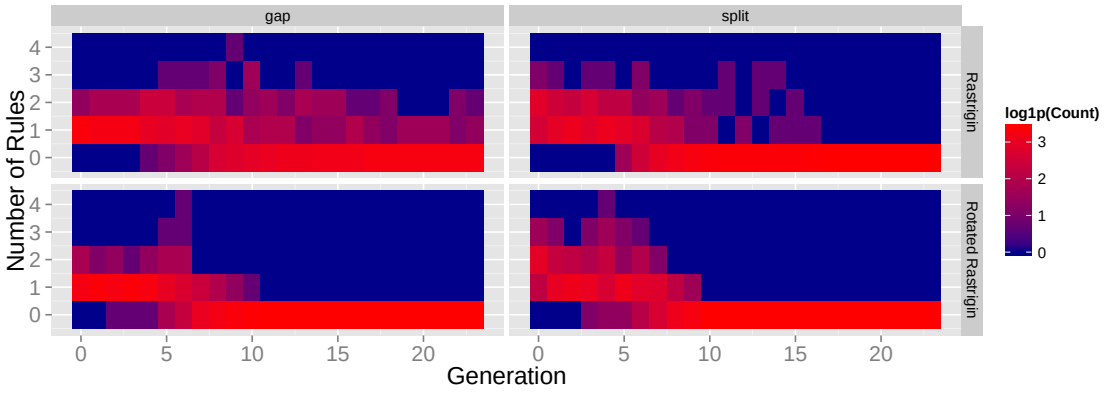


Figure 5.24: This shows the log transformed aggregate number of rules by generation for f_{rast} and f_{rot_rast} runs by training set configurations *gap* and *split*. The shows that the machine learner will generally stop learning rules sooner when the Rastrigin function is rotated.

sheer weight of numbers in the inferior local optima.

In this scenario increasing the population size had no performance impact. If one were to think of the hyper volumes of the local minima taking up a proportional amount of space, then any sampling, no matter the size, will reflect those proportions. Given the large size of the local minima that traps most of the populations and the comparatively small size of the global optima, then any sampling will reflect that. Indeed, this type of scenario was also observed for EDAs in that large basins of attraction would lure individuals for the Rosenbrock and a tailored test function [Yuan and Gallagher, 2005].

Resolving the intractability of f_{lang} may not have been necessary if a more legacy-like EA configuration was used that employed mutation and crossover and other sources of selection pressure such as would be articulated via parent and survival selection. That is, these operators may work well in tandem with the machine learner to give it the traction it needs to tackle f_{lang}, or other similar problems. This influence of legacy EA configurations and operators is discussed further in §7.2 on page 165.

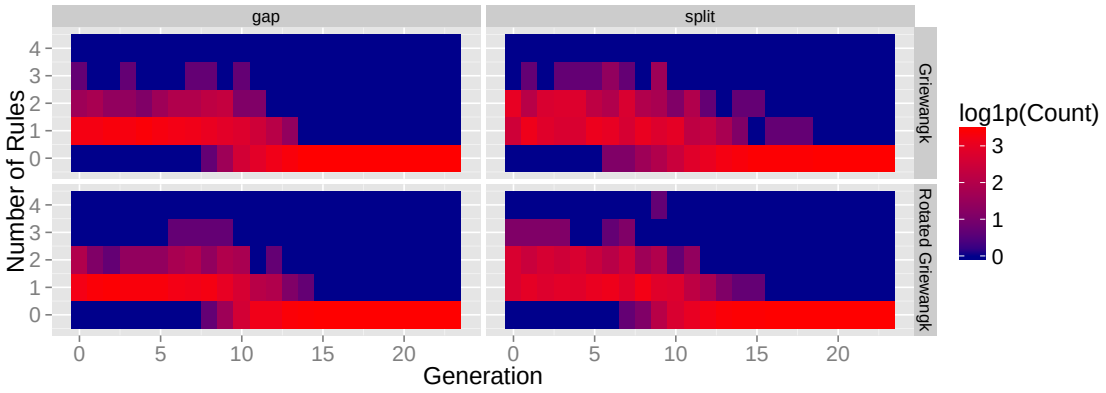


Figure 5.25: This shows the log transformed aggregate number of rules by generation for f_{grie} and f_{rot_grie} runs by training set configurations *gap* and *split*. The shows that randomly rotating the Griewangk has little impact on the number of rules the machine learner learns.

Effects of random landscape rotation

One significant observable difference between the non-rotated and rotated Rastrigin runs is that the latter has far more outliers, which are indicative of the machine learner not learning any rules fairly early in some runs. This is corroborated by the number of rules that the machine learner learns as shown in Fig. 5.24. That is, randomly rotating the landscape removes some of the axially aligned regularity that the ML was taking advantage of, consequently it learns fewer rules per generation.

Note that rotation for f_{rot_grie} eliminated difference between the *gap* and *split* runs for the tail end of the runs.

5.3 By Fitness Percentage

In the previous section we explored the effects of training sets sizes as selected by fitness rank of the parent population. An alternative strategy, again as described in some detail in section 2.1.1 on page 14, would be to instead select the individuals by *percentage* of the full fitness range. That is, the parents are sorted by fitness as before, but the "best" and "worst" individuals are selected based on percentage of the full range of fitness values. This means that the number of individuals in the "best" and "worst" training sets will vary from

Table 5.2: One tailed pairwise Bonferonni adjusted Wilcoxon rank sum p-values for fitnesses of the last generation for three different by-percentage training set configurations for f_{sphe}. The hypotheses are that the median fitnesses of the row values are less than the column. This corroborates the results from Figs. 5.26 and 5.27 by showing that the *middle* runs converged the furthest of the three from the global optimum, whereas the *split* was the closest.

	split	middle
middle	1.0	
gap	1.0	< 0.001

generation to generation; moreover, the fitness landscape will have a direct influence on the proportions of these training sets as we will soon see.

As in the previous section, we will look at three sets of runs for the test suite that cover the three training set strategies, but do so by fitness percentage instead. The *gap* training configuration will now take the top third and bottom-most third individuals as a *percentage* of overall fitness values for the "best" and "worst" training sets, respectively. For the *middle* configuration the "best" and "worst" instances are taken from the middle of the full range of fitness values with the extrema being excluded for consideration by the ML. And, finally, for the *split* training set configuration, we take the top half by percentage of fitness to be the "best" parents and the rest as "worst" for training the machine learner.

We will now look at the effects of the emergent selection pressure inherent to the ML when used to create offspring when training sets are selected by fitness percentages instead of by fitness rank. Note that for all the runs, unless otherwise specified, that the rule intervals were sampled with a Gaussian distribution of three σ and the *best* open rule interval closing strategy was used.

5.3.1 Results

Here I will first share the results by the various fitness landscapes described in §3.4 on page 34 using the by-fitness-percentage approach for training set configurations.

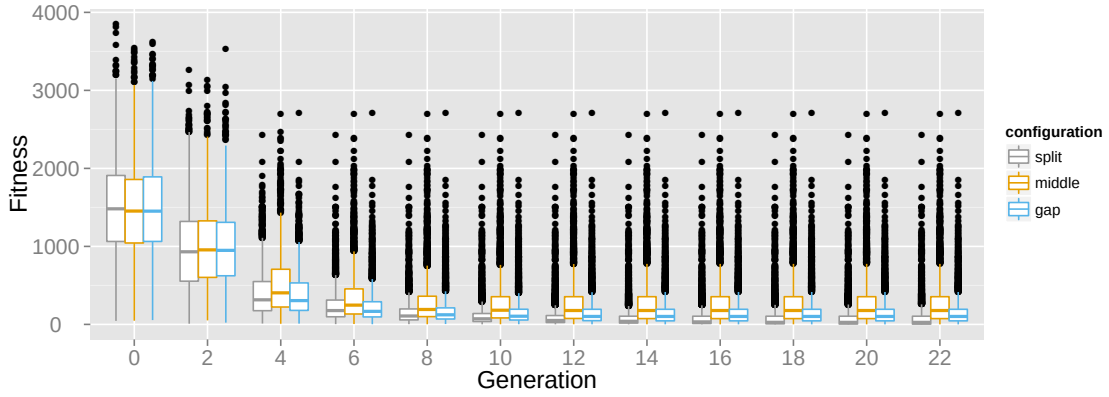


Figure 5.26: This shows the aggregate population trajectories for f_{sphe} for the training set configurations *gap*, *middle*, and *split*. A Gaussian distribution of three standard deviations was used to sample the rule intervals, and the *best* open rule interval strategy was used. This shows that, as before, the *split* and *gap* approach more closely to the global optimum than *middle*.

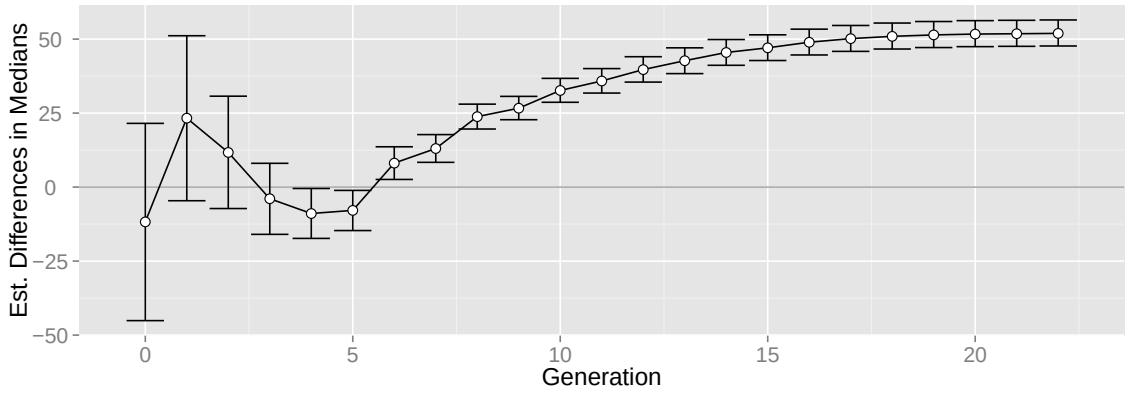
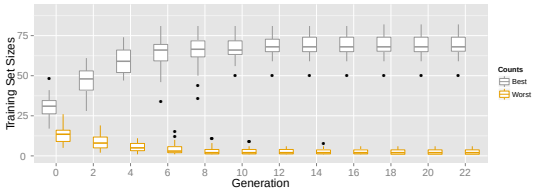


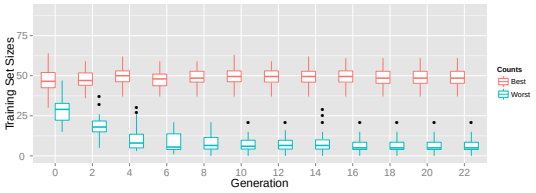
Figure 5.27: This shows the by-generation differences between the *split* and *gap* runs for f_{sphe}. Medians below zero favor *gap* those above *split*. The first four generations there is no significant difference between the two sets of runs. Then for two generations the *gap* training set configuration runs are a little closer to the global optimum before the *split* then overtakes *gap* to converge more closely to the optimum; this difference widens by generation throughout the run.

Spheroid

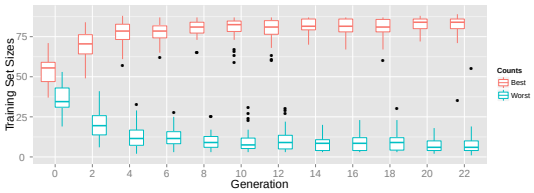
Fig. 5.26 shows the population trajectories for the *split*, *gap*, and *middle* training set configurations for the f_{sphe}. We can see that the *middle* continues the trend of converging further from the global optima than the *gap* and *split* runs; the *split* runs appears to converge a little more quickly than *gap*, though it is difficult to discern here. We get a



(a) This shows the number of "best" and "worst" individuals by generation for the *gap* training set configuration for f_{sphe} where we can see that the number of "best" steadily increases for the first ten generations while the "worst" decreases to almost zero.



(b) This shows the number of "best" and "worst" individuals by generation for the configuration *middle* for f_{sphe}. The number of "best" remain roughly the same from generation to generation, but the number of "worst" parents decreases for the first ten generations before stabilizing. However, the number of "worst" are still much smaller than "best" through all the f_{sphe} runs.



(c) This shows the number of "best" and "worst" individuals by generation for the *split* configuration for f_{sphe}. This is similar to Fig. 5.28a in that the "best" size steadily increases for the first several generations while the "worst" decreases.

Figure 5.28: This shows the number of "best" and "worst" parents by generation for the f_{sphe} runs for all training set configurations. For all three sets of runs the "worst" decrease in size for the first several generations before stabilizing; moreover they are significantly smaller than the corresponding set of "best" parents. For the *gap* and *split* runs the "best" increase while "worst" decrease; for the *middle* runs the number of "best" individuals remain roughly the same throughout the runs.

better perspective on their respective differences in Fig. 5.27, which shows that the first few generations that *gap* and *split* runs are similar, then *gap* runs converge a little more closely to the global optimum for the fifth and sixth generations before *split* then overtakes *gap* to then steadily widen the gap by generation until the end of the run. Tbl. 5.2 shows the Wilcoxon pairwise results for the last generation for all three configurations that the distance from the global optima is ranked *split*, *gap*, and *middle* in order of decreasing distance.

Fig. 5.28 shows the relative sizes of the "best" and "worst" training sets for all the f_{sphe}

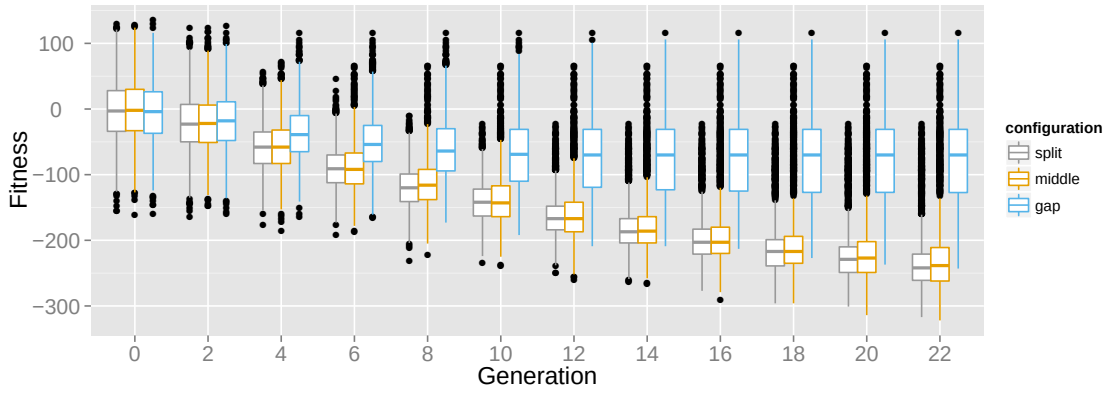


Figure 5.29: This shows the population trajectories for f_{step} for all three training set configurations. The *gap* training set configuration runs converge slowly for several generations before stagnating, which is unusual since it is normally the *middle* runs that take on that role. Instead, it is now the *middle* and *split* that have similar trajectories as they continue to converge towards the global optimum at negative infinity.

Table 5.3: One tailed pairwise Bonferonni adjusted Wilcoxon rank sum p-values for fitnesses of the last generation for three different by-percentage training set configurations for f_{step}. The hypotheses are that the median fitnesses of the row values are less than the column. This shows that for the last generation the rank of distance from the global optima is *split*, *middle*, and *gap* training set configurations.

	split	middle
middle	1.0	
gap	1.0	1.0

training set configurations. The first thing to note is that even though the percentage values for the "best" and "worst" are the same that the realized proportions of those training sets are far from similar. In particular for all the runs there are consistently greater number of parents in the "best" set than "worst". For the *gap* and *split* configuration runs the gap between the number of "best" and "worst" training set instances widens for several generations before stabilizing; this is partly true for the *middle* configuration in that the two similarly widen for several generation before stabilizing, but with the difference in that it is only the "worst" that shrink whereas the number of "best" instances remains more-or-less the same throughout the generations.

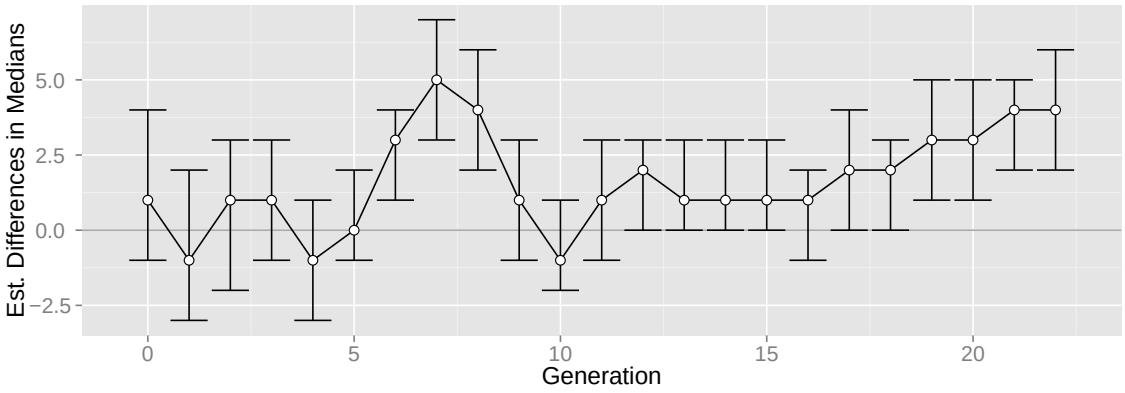
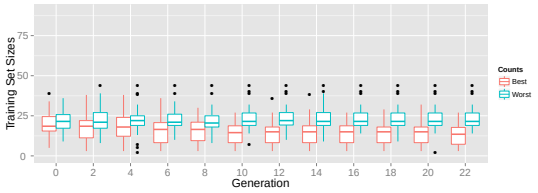


Figure 5.30: Differences by generation between *middle* and *split* runs for f_{step}. Values below zero favor middle; those above zero favor *split*. We can see that the only times the medians for the runs are statistically significantly different are seventh through ninth generations, 18th generation, and twenty forward where *split*'s medians are slightly closer to the global optimum.

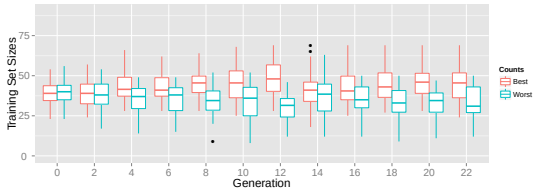
Step

Fig. 5.29 shows the f_{step} population trajectories for the *gap*, *middle*, and *split* training set configuration runs. Here we can observe that, unusually, it is now the *gap* configuration that behaves as the *middle* configuration for previous runs; now it is the *split* and *middle* that have trajectories that are difficult to differentiate as they steadily continue towards the global optimum at negative infinity. Fig. 5.30 shows the differences between the medians of the *middle* and *split* runs where we see that much of the time there is no difference, though there are occasions that the *split* training set configuration medians are slightly closer to the global optimum than the *middle* configurations. Tbl. 5.3 shows the Bonferonni adjusted Wilcoxon rank sum results between the three training set configurations, and corroborates what we have already seen: that the *split* is closest to the global optimum, followed by the *middle* runs, and then trailing furthest from the global optima are the *gap* configuration runs.

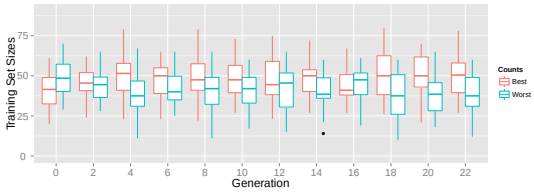
Fig. 5.31 shows the aggregate counts for the "best" and "worst" individuals for f_{step} by generation for the three training set configurations. Unlike the other test problems, here there is some overlap between the sizes of the "best" and "worst" training sets; though for



(a) This shows the number of "best" and "worst" individuals by generation for the *gap* training set configuration for f_{step}. Note that there are generally more "worst" individuals than "best" for all generations.



(b) This shows the number of "best" and "worst" individuals by generation for the configuration *middle* for f_{step}. After a few generations there are slightly more "best" than "worst" individuals in the ML training sets.



(c) This shows the number of "best" and "worst" individuals by generation for the *split* configuration for f_{step}. Again, there are slightly few more "best" than "worst" parents.

Figure 5.31: This shows the number of "best" and "worst" parents by generation for the f_{step} runs for all training set configurations. The *middle* and *split* have similar ratios of "worst" vs. "best" parents, though the *split* has higher counts of each, presumably because *middle* intentionally excludes extrema. However, the *gap* runs have far fewer parents in the "best" and "worst" training sets than either *gap* and *split*.

the *middle* and *split* configurations the number of "best" is still somewhat larger than the size of the "worst" training sets, for the *gap* it is the "worst" that is larger than the "best".

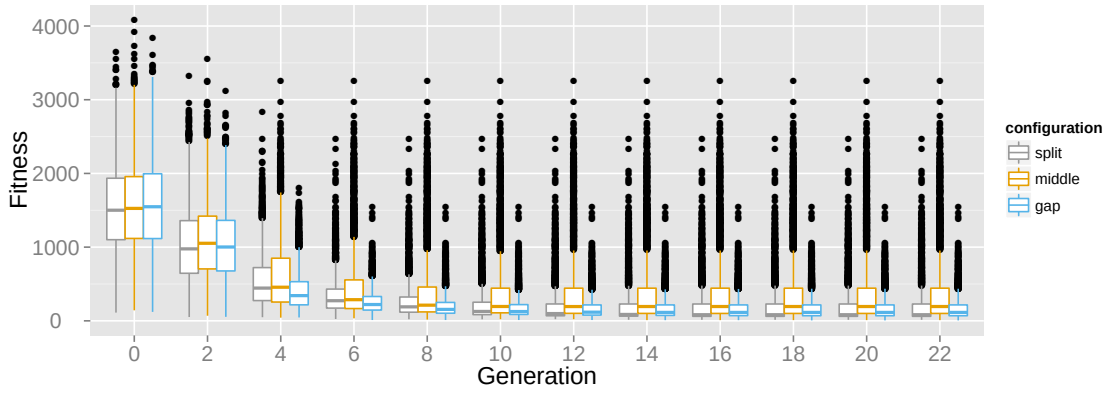


Figure 5.32: The population trajectories for the three training set configurations for f_{rast}. This shows that *gap* is slightly more aggressive in converging to the global optima than *split*; whereas *middle* configuration lags a little further behind those two and has a higher variance from the fourth generation forward.

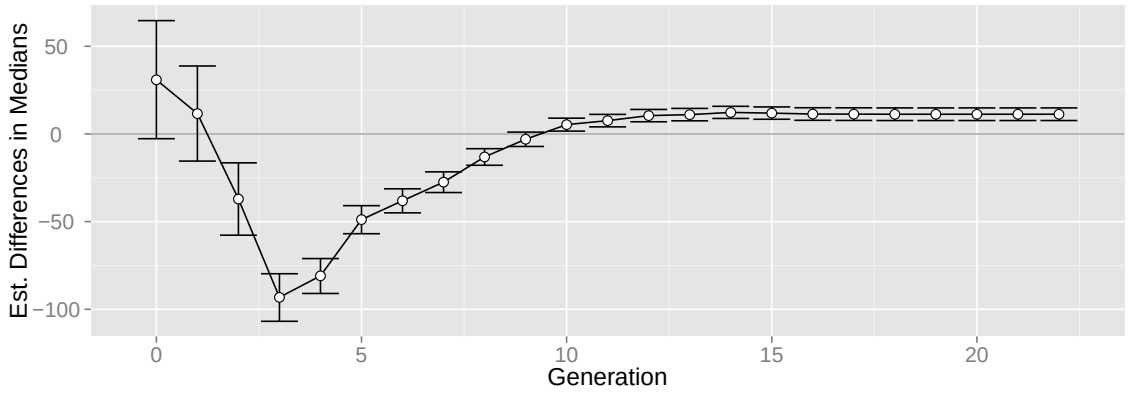


Figure 5.33: Differences by generation with 95% confidence intervals for f_{rast} between the *gap* and *split* runs. Values below zero favor *gap*, whereas those above zero favor *split*. We can see the *gap* converges more quickly for the first several generations to then switch to *split* converging more quickly.

Table 5.4: One tailed pairwise Bonferonni adjusted Wilcoxon rank sum p-values for fitnesses of the last generation for three different by-percentage training set configurations for f_{rast}. The hypotheses are that the median fitnesses of the row values are less than the column. The observed ordering of training set configuration by distance from the global optima is as the *split* as the closest, followed by the *gap*, and then with the *middle* being the furthest.

	split	middle
middle	1	
gap	1	< 0.001

Table 5.5: One tailed pairwise Bonferonni adjusted Wilcoxon rank sum p-values for fitnesses of the last generation for three different by-percentage training set configurations for f_{rot_rast}. The hypothesis are that the median fitnesses of the row values are less than the column. Once again this shows that the *split* configuration runs end closest to the global optimum of the three, followed by the *gap* and then the *middle* configuration runs.

	split	middle
middle	1	
gap	1	< 0.001

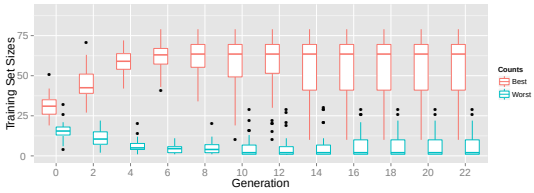
Rastrigin

Fig. 5.32 shows the population trajectories for the three training set configurations for f_{rast}. We can observe that the *gap* and *split* configuration runs are again similar whereas the *middle* lags behind both. Fig. 5.33 shows the estimated differences between the medians for *gap* and *split* between those runs by generation. Note that the *gap* configuration converges more quickly to then switch to the *split* converging more quickly after the population has converged near the optima. This is similar to what we have seen earlier with the f_{sphe} runs in Fig. 5.27 on page 118. This is further corroborated by the end-of-run statistics given in Tbl. 5.4 that shows the Bonferonni adjusted Wilcoxon tests for the three training set configuration, and indicates that the *split* is the closest to the global optima, followed by *gap*, and with the *middle* runs being the furthest.

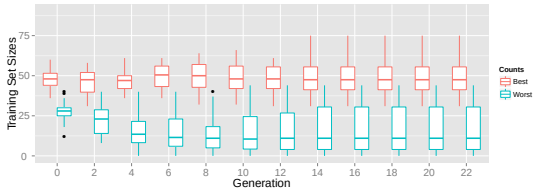
Fig. 5.34 shows the sizes of the "best" and "worst" training sets for f_{rast} broken down by training set configuration. There we can see the continuing trend, with the notable exception of f_{step} shown earlier, of the number of "best" instances being greater than the "worst"; the gulf between the sizes gradually increase until the runs converge. However, with the *middle* configuration this pattern holds true, though the number of "best" remain roughly the same over time and it is only the number of "worse" that decrease.

Rotated Rastrigin

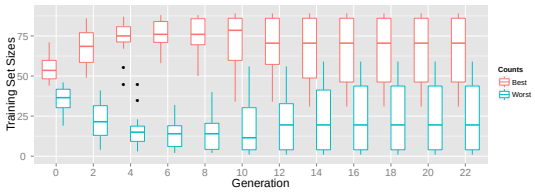
Fig. 5.35 shows the aggregate population trajectories for f_{rot_rast}, which shows once again that the *gap* and *split* similarly converge towards the global optimum, whereas the *middle* lags behind both. Fig. 5.36 shows the differences by generation between the *gap* and *split*



(a) This shows the number of "best" and "worst" individuals by generation for the *gap* training set configuration for f_{rast}. As with other runs, for the first several generations the number of "best" and "worst" training instances widens for the first several generations before stabilizing with there being more "best" than "worst."



(b) This shows the number of "best" and "worst" individuals by generation for the configuration *middle* for f_{rast}. We can see that, as with other *middle* runs, the number of "worst" decreases where as the number of "best" remain roughly the same.



(c) This shows the number of "best" and "worst" individuals by generation for the *split* configuration for f_{rast}. As with Fig. 5.34a there are more "best" than "worst", though there is some overlap after the runs converge around generation ten or twelve.

Figure 5.34: This shows the number of "best" and "worst" parents by generation for the f_{rast} runs for all training set configurations. We can see that the number of "best" is generally greater than "worst" for all training set configurations as happened with f_{sphe}. However we can see that for the *split* configuration there is some overlap after the runs have converged. The *middle* configuration continues its trend of having the "best" be larger than "worst" but with the "best" remaining the roughly the same while it is the number of "worst" instances that decrease over time.

runs. That plot shows that for the period where the runs are converging the most quickly it is the *gap* that is the most aggressive; however, after the runs have settled there is no significant difference between the two. By contrast, Tbl. 5.5 shows the Bonferonni corrected Wilcoxon rank sum tests comparing the three and shows that the by now established pattern that at the end of run the *split* is the closest to the global optimum, followed by the *gap*, and then the *middle*. It is likely that this end-of-run analysis dictates that there is a difference between the *gap* and *split* configuration runs, which differs from the story told in Fig. 5.36 is due to the Bonferonni p-value adjustment.

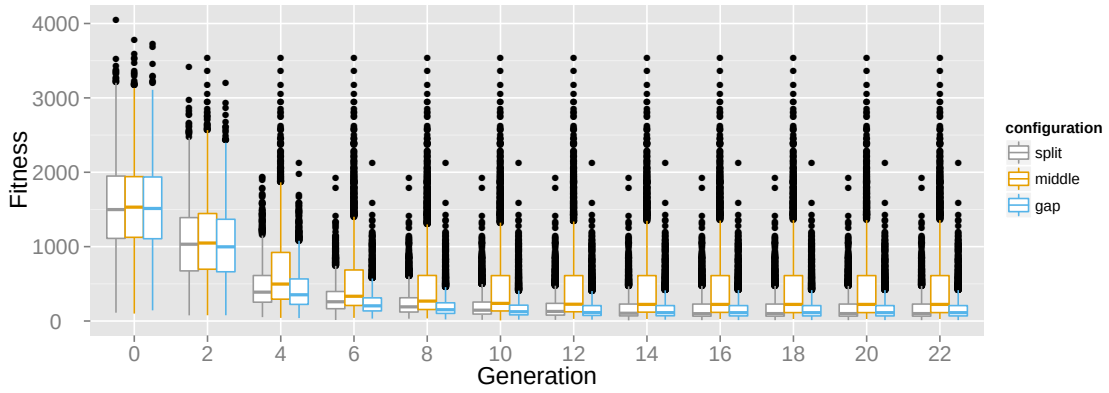


Figure 5.35: This shows the collective population trajectories for f_{rot_rast} for the training set configurations *gap*, *middle*, and *split*; once again the *gap* and *split* configurations move steadily towards the global optimum while the *middle* lags behind both.

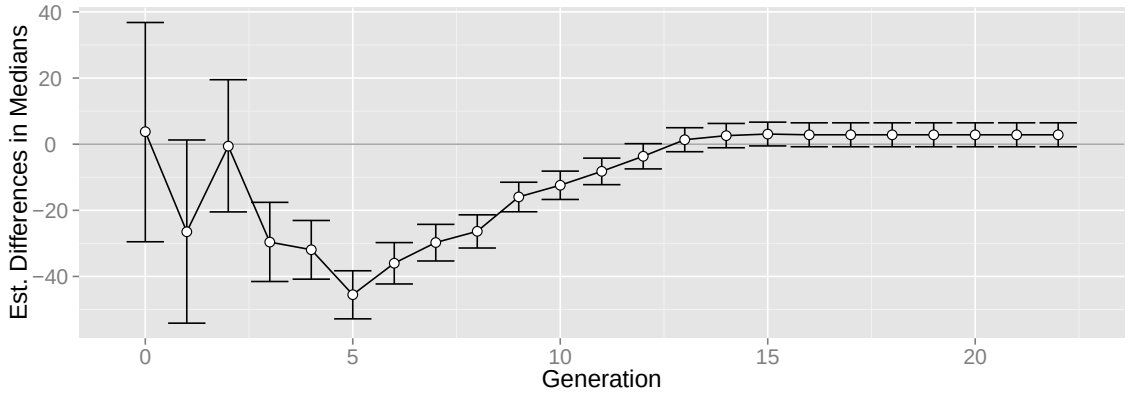
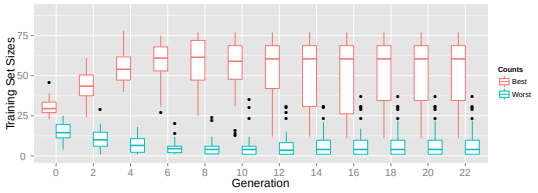


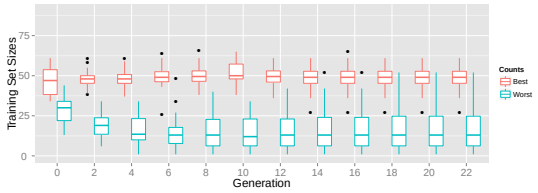
Figure 5.36: This shows the differences in medians with 95% confidence intervals between the *gap* and *split* training configuration runs for f_{rot_rast}. Values below zero favor the *gap* runs and the medians above zero favor *split*. The only significant differences are between generations 4 and 12, inclusive, and show that the *gap* configuration is closer to the global optimum.

Fig. 5.37 shows the relative sizes of the "best" and "worst" training sets by training set configuration. We see that once again, as with the unrotated f_{rast}, that there are more "best" than "worst" training instances, though once more the *middle* has the "worst" decreasing until the system stabilizes whereas the "best" remain roughly the same through all generations for all runs. Also, the associated "best" and "worst" count distributions run close to one another after the runs have converged for the *split* configuration.

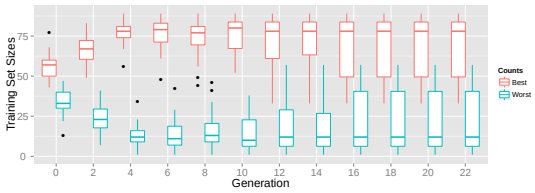
Fig. 5.38 shows the estimated differences between the non-rotated and randomly rotated



(a) This shows the number of "best" and "worst" individuals by generation for the *gap* training set configuration for f_{rot_rast}. Once again there are more "best" than "worst" individuals for all generations with the difference gradually increasing from the first generation until stabilizing after several generations.



(b) This shows the number of "best" and "worst" individuals by generation for the configuration *middle* for f_{rot_rast}. As with previous *middle* configuration runs, the "best" outnumber the "worst", but only the "worst" decrease in size while the "best" maintain a fairly consistent count through the run duration.



(c) This shows the number of "best" and "worst" individuals by generation for the *split* configuration for f_{rot_rast}. And with the *gap* configuration runs, the "best" outnumbers the "worst" throughout all runs. This gap widens from the first generation and steadily increases for several generations before stabilizing.

Figure 5.37: This shows the number of "best" and "worst" parents by generation for the f_{rot_rast} runs for all training set configurations. We see the the number of "best" is always greater than the "worst"; generally this difference increases from the first generation to stabilize after several generations. The only difference, as with other fitness functions, is that in the *middle* configuration, the gap widens only because the "worst" decrease in count whereas the "best" maintain roughly the same size throughout.

versions of the Rastrigin test function using the by-percentage fitness training set configuration approach. I used the *split* training set configuration for this comparison since for both sets of Rastrigin runs that configuration converged slightly closer to the global optimum. We can see that at the start of the run the non-rotated Rastrigin is closer to the global optimum, but then for several generations it is the rotated Rastrigin that surges ahead before finishing with the non-rotated Rastrigin being slightly closer to the global optimum. The Wilcoxon rank sum statistics for the end of runs has the non-rotated Rastrigin closer to the global optima than the rotated ($p < 0.001$).

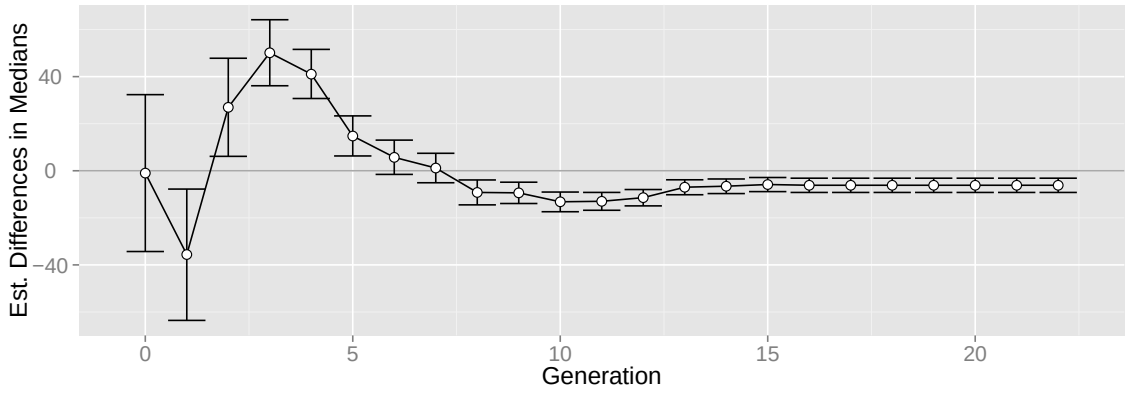


Figure 5.38: This shows the differences between f_{rast} and f_{rot_rast} with 95% confidence intervals. Median values below zero favor f_{rast}, whereas those above zero favor f_{rot_rast}. For the first generation the non-rotated runs are closer to the global optimum, then for several generations the rotated, then once the system has converged the non-rotated Rastrigin runs are slightly closer to the global optimum.

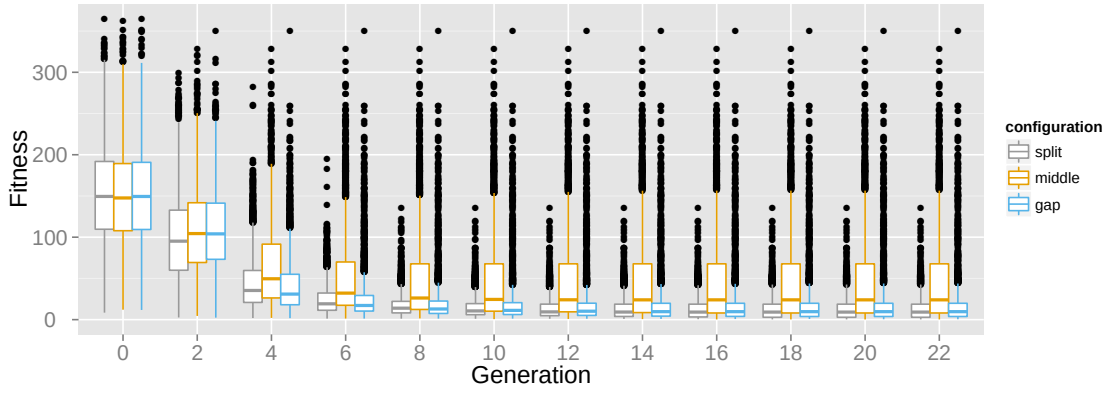


Figure 5.39: This shows the population trajectories for f_{grie} for the three training set configurations *gap*, *middle*, and *split*. We can see that *gap* and *split* training set configurations readily converge towards the global optimum whereas the *middle* comparatively lags behind both of them.

Table 5.6: One tailed pairwise Bonferonni adjusted Wilcoxon rank sum p-values for fitnesses of the last generation for three different by-percentage training set configurations for f_{grie}. The hypotheses are that the median fitnesses of the row values are less than the column. This continues to establish the pattern that for the end-of-runs, the *split* is closest to the global optimum, followed by the *gap*, and then the *middle* training set configurations.

	split	middle
middle	1	
gap	1	$p < 0.001$

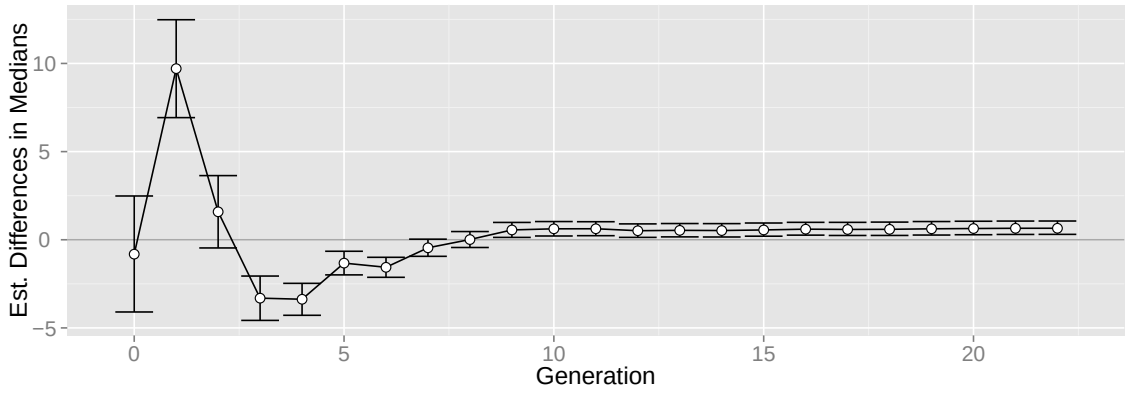
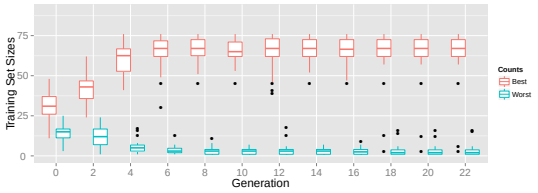


Figure 5.40: This is a comparison between the estimated medians between the f_{grie} *gap* configuration and *split* configurations. Values below zero favor *gap*, those above, *split*. This shows that for the second generation *split* converges more quickly. However for the fourth through seventh generations the *gap* converges a little more aggressively, to then swap places again with *split* once the system has converged.

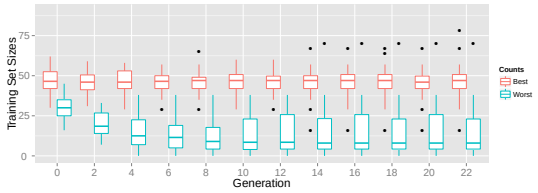
Griewangk

Fig. 5.39 shows the population trajectories for f_{grie} broken down by training set configuration where we see a continuation of the established pattern of *gap* and *split* readily converging towards the optima leaving *middle* behind. Fig. 5.40 all the better shows the differences between the *gap* and *split* training set configuration runs for f_{grie} that were difficult to discern in the previous plot. There we can observe a seesawing effect where the *split* more aggressively converges to then have the *gap* take the lead for a few generations; once the system has converged the *split* configuration is able to move slightly closer to the global optimum. This is corroborated by Bonferonni adjusted Wilcoxon rank sum statistics for the last generation, as shown in Tbl. 5.6; here, again, the *split* ends the runs closest to the global optimum, followed by *gap*, and then *middle*.

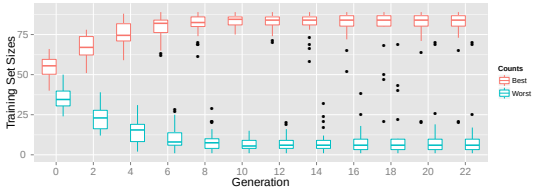
Fig. 5.41 shows the relative counts of the "best" and "worst" training sets by configuration, which mostly closely resembles training set counts we have seen for the f_{sphe}. That is, once again there are more "best" than "worse" instances with gap increasing from the first generation until it stabilizes onces the system has more-or-less converged. Once more the *middle* configuration also has such a gap, but with the "best" remaining the same



(a) This shows the number of "best" and "worst" individuals by generation for the *gap* training set configuration for f_{grie}.



(b) This shows the number of "best" and "worst" individuals by generation for the configuration *middle* for f_{grie}.



(c) This shows the number of "best" and "worst" individuals by generation for the *split* configuration for f_{grie}.

Figure 5.41: This shows the number of "best" and "worst" parents by generation for the f_{grie} runs for all training set configurations. The number of the "best" training set instances continues to outnumber the "worst" for all the runs. Of note is that once again the *middle* configuration runs have a flat number of "best" for all generations.

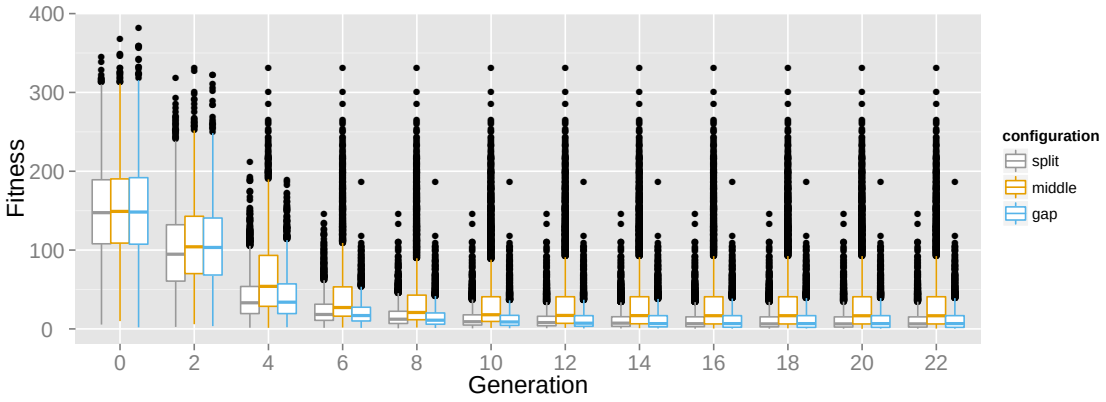


Figure 5.42: This shows the population trajectories for f_{rot_grie} by training set configuration. Once again, the *gap* and *split* configurations steadily converge towards to the optima while the *middle* configuration lags behind both.

throughout the run.

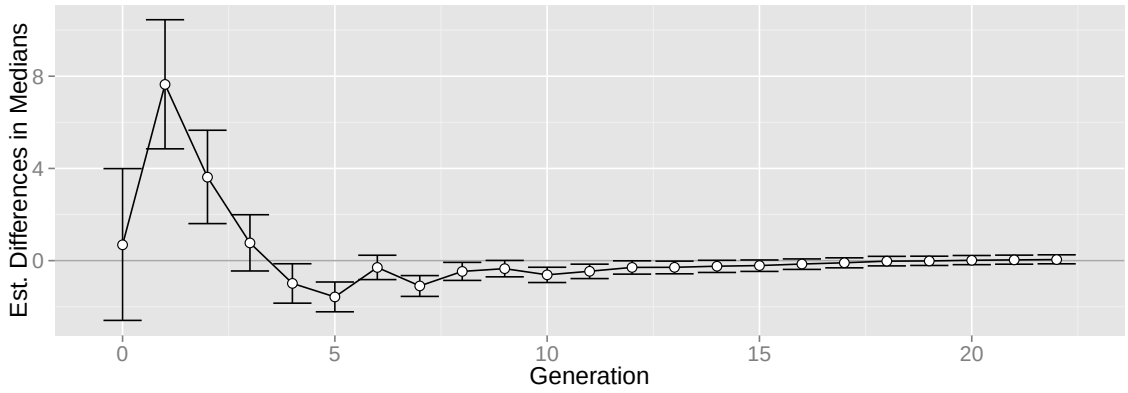


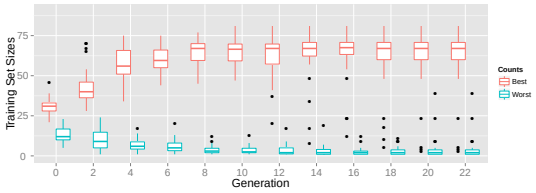
Figure 5.43: This shows the estimated differences by generation for f_{rot_grie} between the *gap* and *split* runs. Values below zero favor *gap* configuration whereas those above favor *split*. So for the first couple generations the *split* converges more quickly to then switch to the *gap* configuration converging more quickly towards the global optima; however, this is only true for a few generations until there is no significant difference between them after the runs converge.

Table 5.7: One tailed pairwise Bonferonni adjusted Wilcoxon rank sum p-values for fitnesses of the last generation for three different by-percentage training set configurations for f_{rot_grie}. The hypothesis are that the median fitnesses of the row values are less than the column. This continues the now established trend that the *split* configuration runs end closest to the global optimum, followed by the *gap*, and with the *middle* configuration the furthest.

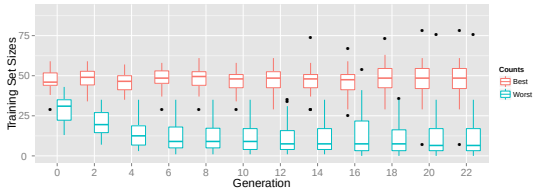
	split	middle
middle	1	
gap	1	$p < 0.001$

Rotated Griewangk

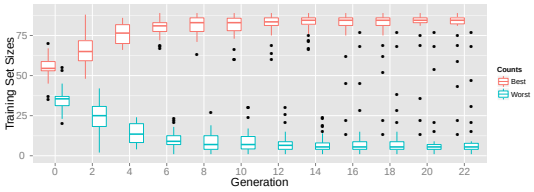
Fig. 5.42 shows the population trajectories for *gap*, *middle*, and *split* for the f_{rot_grie} runs. There we can see the continuation of the by now established pattern: that the *gap* and *split* configuration steadily converge towards the global optima with the *middle* configuration lagging behind both. Fig. 5.43 shows the estimated differences between the *gap* and *split* runs for f_{rot_grie}. Of note is that the *split* converges more quickly than *gap* for the first couple generations to then have *gap* converge faster, but with a rate that gradually diminishes. Tbl. 5.7 shows the Bonferonni adjusted pairwise Wilcoxon statistics for the last generation, and shows once again that the *split* converges more closely to the global optimum, followed by the *gap*, and with the *middle* the furthest. Though Fig. 5.43 shows now difference between the *split* and *gap* runs, they do not take into consideration the Bonferonni p-value



(a) This shows the number of "best" and "worst" individuals by generation for the *gap* training set configuration for f_{rot_grie}. The "best" outnumber the "worst" throughout all runs with a gap that initially widens for several generations before stabilizing.



(b) This shows the number of "best" and "worst" individuals by generation for the configuration *middle* for f_{rot_grie}. Once again the "best" outnumber the "worst"; however, as with previous *middle* runs, the number of "best" remain roughly the same through the runs whereas the "worst" gradually decrease in number.



(c) This shows the number of "best" and "worst" individuals by generation for the *split* configuration for f_{rot_grie}. This is similar to Fig. 5.44a in that the "best" outnumber the "worst", but the gap between their respective sizes is larger.

Figure 5.44: This shows the number of "best" and "worst" parents by generation for the f_{rot_grie} runs for all training set configurations. This shows that count of "best" training instances is consistently larger than the corresponding "worse" for all three configurations. Generally the gap between these two sets widens for several generations before stabilizing; in the case of the *middle* the number of "best" remains more-or-less constant throughout the runs.

adjustment.

Fig. 5.44 shows the relatives sizes of the "best" and "worst" training sets by training set configuration for f_{rot_grie} and continues a by now well established pattern that we have previously observed in all but the f_{step} test function. That is, the number of "best" training instances are greater than the "worst"; the gap gradually increases from the start of the runs to stabilize after several generations; in the *middle* configuration "best" remain roughly the same throughout all the runs leaving the "worst" to decrease in size before itself stabilizing.

Fig. 5.45 shows the by-generation estimated differences in medians between the randomly rotated and non-rotated versions of the Griewangk test function. There is no significant difference between the two for the first few generations, but after that the runs for

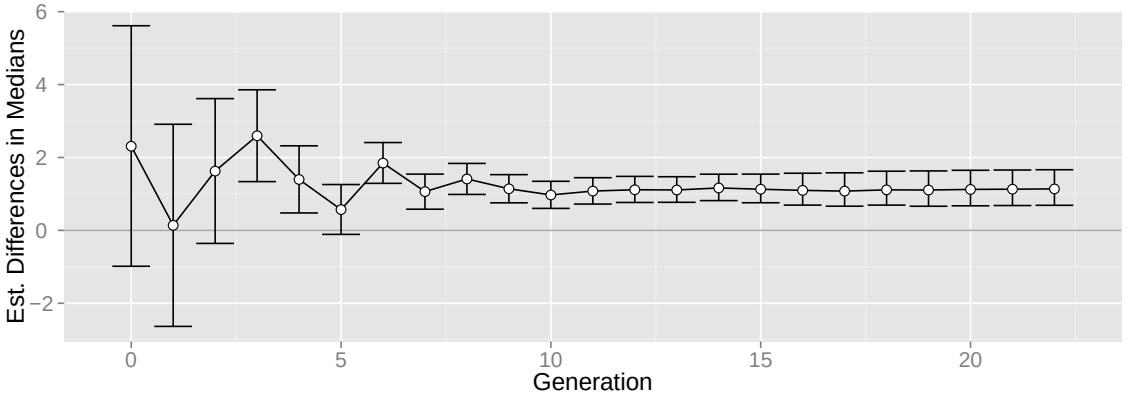


Figure 5.45: This shows the estimated differences between medians for the non-rotated and rotated Griewangk test function runs. Lower values favor f_{grie} where as those above zero favor f_{rot_grie}. Though there is no significant difference for the first three generations between the two versions of Griewangk, there is a small difference in favor of the f_{rot_grie} runs.

rotated are slightly closer to the global median; this is corroborated by comparing the last generation which shows that the rotated runs are significantly closer to the global optimum (Wilcox $p < 0.001$). This is surprising since the non-rotated version should be better since presumably the algorithm can gain better traction on the axially aligned optima. However, the difference is very small — the difference is consistently around one after the algorithms settle after ten generations.

Langerman

Fig. 5.46 tells a by now familiar story for the Langerman test function. This plot shows the population trajectories for the *gap*, *middle*, and *split* training set configurations for f_{lang}, which shows that changing to training sets built by percentage of fitness instead of by rank has no impact on SLEM's performance with regards to the Langerman function.

Fig. 5.47 shows the relative sizes of the "best" and "worst" training set sizes for f_{lang} for all three training set configurations. Here we can see that the previously established patterns for the relative training set sizes does not hold for the Langerman function. The first thing we note is that the variances for the "best" and "worst" are quite large compared to the previous test problems, which contributes to a significant overlap; the one notable exception

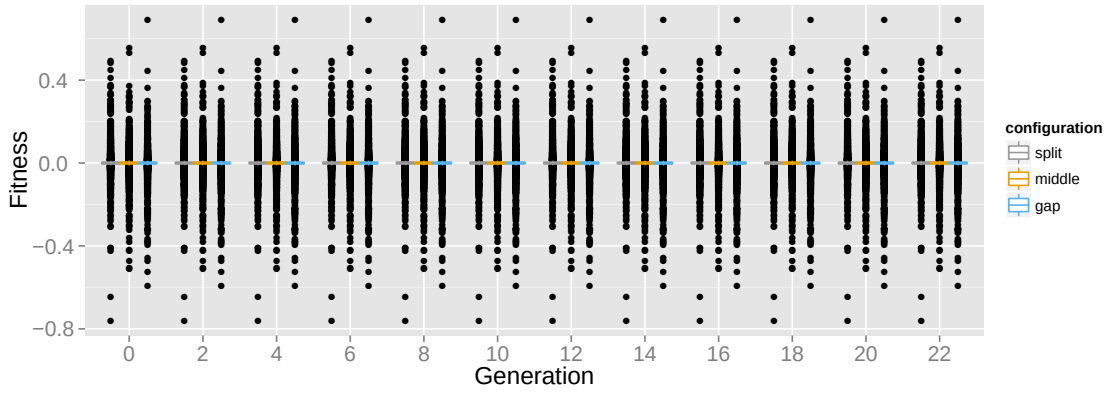


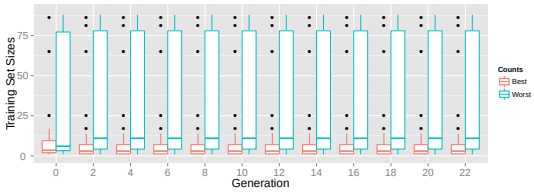
Figure 5.46: Shows the population trajectories for f_{lang} for the three training set configurations *gap*, *middle*, and *split*. This shows that creating the "best' and "worst" training sets by percentage of fitness values does not change performance as compared to the by-rank fitness approach.

is with the *gap* configuration where the variances for the "best" are comparatively small to the "worst". For the previous runs the "best" training set was typically much greater than the "worst", which is not entirely true here. For the *gap* and *middle* runs the "worst" are somewhat higher though this claim is somewhat weakened by the overlap between the two training set sizes. The *split* training set sizes appear to somewhat be the inverse of the *middle*'s; though both sets have large variances and overlap, it is the "best" that is larger than the "worst" for the *split*.

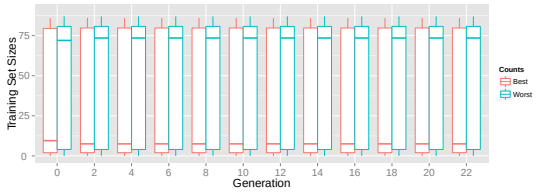
Fig. 5.48 shows the aggregate number of by generation for f_{lang} for all three training set runs. There we can observe unlike before that there were some rules learned, albeit only for a handful of runs and only for the first generation.

5.3.2 Discussion

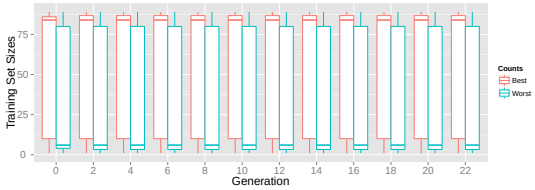
Here I discuss the results of the runs where the training sets were constructed based on the percentage of fitnesses instead of by fitness rank. Note that the discussion will focus on just these results and will not compare the by-rank and by-percentage training set approaches. These two training set approaches are compared in §5.4.



(a) This shows the number of "best" and "worst" individuals by generation for the *gap* training set configuration for f_{lang}. This shows that there was more "worst" than "best" training set instances; moreover the variance of the "worst" was much higher than for the "best".



(b) This shows the number of "best" and "worst" individuals by generation for the configuration *middle* for f_{lang}. There are more of the "worst" than "best", but both sets have a larger variance and significant overlap.



(c) This shows the number of "best" and "worst" individuals by generation for the *split* configuration for f_{lang}. This is essentially the inverse of Fig. 5.47b in that now it is the "best" that is larger than the "worst", but there is a similar large variances for both training sets as well as overlap.

Figure 5.47: This shows the number of "best" and "worst" parents by generation for the f_{lang} runs for all training set configurations. The "worst" outnumbers the "best" for the *gap* and *middle* configurations, whereas it is the "best" that outnumbers the "worst" for the *split*. However, there variances for these counts are quite large and have a great deal of overlap, except for the "best" in the *gap* configuration.

Sizes of training sets vary by runs and test problem

The first sets of runs in this chapter used the by-rank approach for creating the "best" and "worst" training sets, which meant that their sizes remained static throughout all the runs. By contrast, we have observed that for all the runs where the by-fitness-percentage approach was used that the training set sizes were dynamic — they would change from generation to generation as well as being influenced by test problem.

This comes as no surprise as we would expect that as the population changes so would the size of the "best" and "worst". Even the initial population, which represents a uniform sampling of the problem space, would reflect the underlying topology by relative size of the

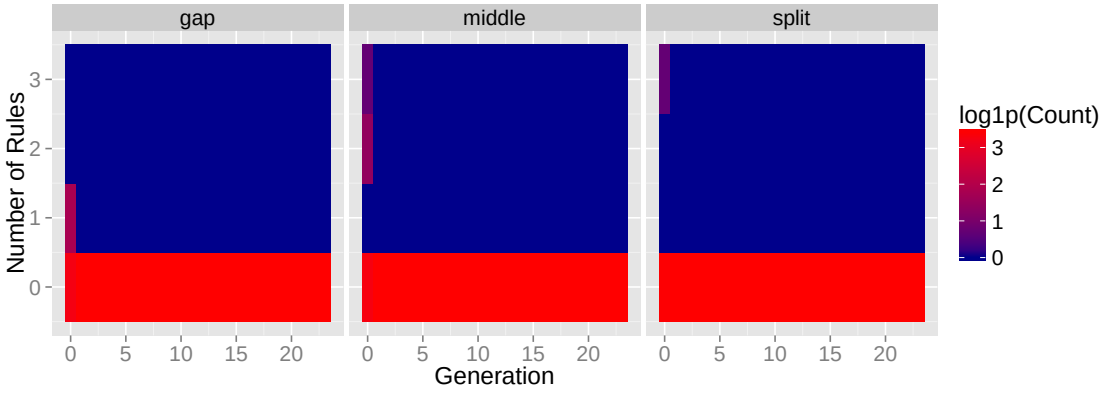


Figure 5.48: This shows the aggregate count of rules learned by generation by training set configuration for f_{lang} for the by-percentage approach for creating training sets. The color scaled has been log transformed for visual clarity. This shows that the first generation is the only generation that the ML learns anything. The *gap* runs did not learn more than a single rule for the first generation; the *middle* learned two or three rules for the first generation, and the *split* learned three rules in some of its runs.

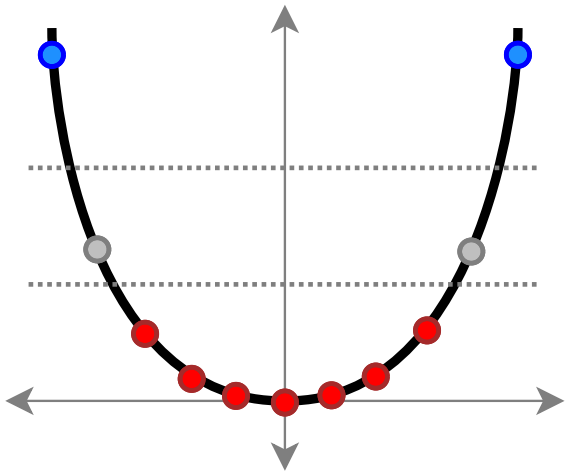


Figure 5.49: This represents a one dimensional f_{sphe} sampled with regularly spaced points, which shows the influence that problem topology can have on the number of instances in each training class. The dashed lines represent the boundaries for the "best" and "worst" training sets using a by-fitness-percentage approach. The top third, represented as blue points, has two individuals whereas the bottom third, represented in red, has seven; the "mediocre" set, which is ignored by the ML, is represented by two gray points

training set.

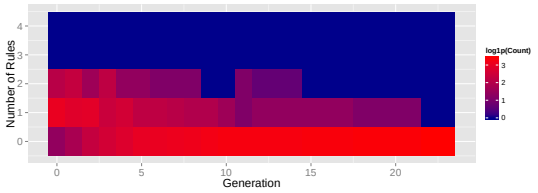
Consider f_{sphe} which consistently has more "best" than "worst" throughout all its runs. Again, the initial population reflects a uniform sampling of the problem domain, but even

then there will generally be more individuals in regions of relatively higher fitness within the large basin of attraction of the global optima. This is depicted in Fig. 5.49 which shows a one dimensional f_{sphe} evenly sampled with ten points. Dividing the fitness values by thirds, we observe that the top third, which represents the "worst" and is shown in blue, has just two points compared to the seven red points in the bottom third representing the "best". We can envision as the population converges down the gradient under the guidance of the machine learner that the gap between the "best" and "worst" sizes would widen; and, indeed, that is what is observed for all but the f_{step} and f_{lang} functions, which I will cover later.

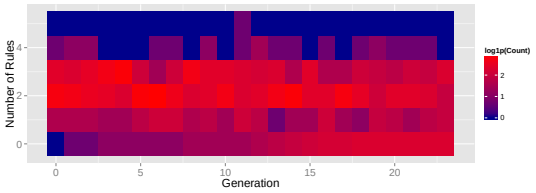
Except for f_{step} and f_{lang}, the *gap* and *split* training set configurations exhibit similar patterns; i.e., the number of "best" is always greater than the "worst", the difference grows from the initial generation for several generations until reaching a steady state when the population converges. However, in the case of the *middle* configuration, the pattern is similar with one marked difference: the "best" roughly remains at a steady state throughout the runs, and the widening gap between the "best" and "worst" for those runs is only due to the "worst" decreasing in size as the run progresses. Why does this happen? Given the lopsided proportion of the "best" as compared to the "worst" by fitness percentage, removing the extrema will remove more "best" individuals than the "worst"; so, in a sense, the *middle* configuration is similar to the *split* configuration, but with far fewer "best" and only a few less "worst".

The *middle* training set configuration consistently converges more slowly than *gap* and *split* except for the f_{step} runs. As shown in Fig. 5.29 on page 120 the *middle* configuration runs have essentially exchanged places with the *gap* configuration.

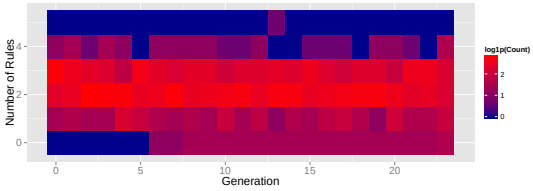
Fig. 5.31 on page 122 gives a possible explanation for why the *gap* training set configuration apparently has swapped with the *middle*. This figure shows the aggregate counts for the "best" and "worst" individuals for f_{step} by generation for the three training set configurations. What stands out is that the *gap* runs have far fewer individuals in their respective training sets compared to the other sets of runs. This may have an impact on



(a) This is the aggregate count of rules using the *gap* training set configuration. We can see that the ML learns at most two rules per generation.



(b) This is the aggregate count of rules using the *middle* training set configuration. Here the ML learns a rich number of rules throughout all the runs.



(c) This is the aggregate count of rules using the *split* training set configuration. We can see that the ML consistently learns a large number rules per generation throughout all the runs.

Figure 5.50: This shows the aggregate count of the number of rules by generation for f_{step} for the by-fitness-percentage approach. We can see that the ML is able to sustain learning a large number of rules through all the runs for the *middle* and *split* configuration, but learns far fewer rules for the *gap* configuration.

the ML performance as presumably it will not have a better perspective on the fitness landscape with fewer individuals. Indeed this explanation is supported by Fig. 5.50 that shows the aggregate counts of rule by generation for f_{step} for all three sets of training set configurations. Here we can see that number of rules learned during the *gap* configuration runs is comparatively anemic to the rules learned in *middle* and *split*.

gap vs. *split* training set configuration

What of the differences on the inherent selection pressure from using the *gap* or *split* training set configurations? Let us work from the end of the runs forward to analyze the differences.

At the end of the runs the populations are typically homogeneous as they should have converged near an optima. Given that *gap* configuration was intended to minimize overlap between the "best" and "worst" training sets for the by-fitness-rank approach, the *gap* configuration may not be as necessary since the by-fitness-percentage approach should do

a better job of minimizing training instance overlap. So at the end of the runs the *split* configuration would overtake *gap* since then larger training set sizes would give enough additional information to the ML to allow it to possibly converge more closely to the optima.

Indeed this is the case — either there is no difference between the *gap* and *split* configuration runs, as happened in f_{rot_rast} and f_{rot_grie} cases, or the *split* converged more closely to the optima than *gap* in the case of f_{sphe}, f_{step}, f_{rast}, and f_{grie}. (f_{lang} is not considered here as there was no significant differences between any of the runs.)

What of the beginning of the runs? There the story is a little murky. The first generation there is no difference as, naturally, that generation reflects a uniform sampling of the problem space. After the initial generation, the behavior varies by fitness function. For f_{sphe}, the first significant differences between *gap* and *split* do not occur for several generations, and even then the *gap* configuration is only moderately more aggressive than *split* before the latter overtakes after the population has converged closely to the optimum.We see a repeat of this pattern in the non-rotated and rotated Rastrigin functions. However, the f_{grie} is a little different in that it is the *split* that converges more quickly before switching with the *gap* configuration; at the end of the run there is no difference between the two for non-rotated Griewangk, and the *split* configuration converges a little more closely to the optimum for the rotated version. The Griewangk is a little more complex than the Rastrigin function; so the larger training set courtesy of the *split* configuration is necessary at the start to find an optima, then it switches to the *gap* while exploiting that optima, to then switch again to *split* as with the f_{sphe} in the non-rotated Griewangk, or to no difference between the two schemes in the rotated Griewangk.

Essentially the *gap* has higher fidelity rules because the "mediocre" set provides a buffer between the "best" and "worst" that minimizes overlap. However, overlap is inevitable once the population has converged, in which case it then becomes more important to have larger training sets.

A word of caution is that in the cases of f_{step} and both rotated and unrotated Griewangk functions is that the differences between the *gap* and *split* configuration runs is small as

compared to the full fitness range. And, in all cases there is a great deal of population overlap between the *gap* and *split* runs. Moreover, the dynamic between these two configurations may change in the presence of higher selection articulated either via more aggressive parent selection and/or survival selection.

By-percentage fitness training sets and f_{lang}

In §5.2.2 on page 112 we became more aware of the large basin of attraction for local optima in the Langerman function. And, here we see evidence that using a by-percentage of fitness values approach to creating training sets does not yield any performance difference as seen in Fig. 5.46 on page 134. Once again, most of the individuals are centered around fitness values of zero.

Except for f_{step}, for all other runs besides f_{lang} there is a fair gap between the "best" and "worst" individuals. For f_{lang} there is a not only an overlap between the two training sets, but the variances for them are also much larger than for the other fitness functions. Moreover the training set composition is very sensitive to the training set configuration. For example, we previously noted that more of the best individuals are removed when going from *split* configuration to *middle*, and we see something similar here. However, there is still a significant overlap between the "best" and "worst" sizes, though their relationship of which is more has flipped when going from *split* to *middle*. Additionally in the *gap* there are more "worst" than "best", and there is a lot more of the "worst" than the "best", which is different from all other runs except for f_{step}. This suggests that f_{lang} is a very complex fitness landscape where changes in test set configuration can yield very different ML-related behaviors.

Of note is that, unlike for the by-fitness-rank runs, the by-fitness-percentage runs the ML was at least able to learn something in the first generation. Unfortunately this was not sufficient for the population to escape the very large basin of attraction of local optima.

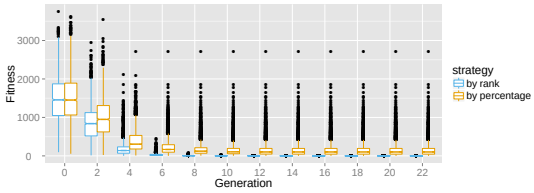
Effects of random landscape rotation

As stated before, some algorithms gain in performance with problems that have axially aligned optima as found in both the Rastrigin and Griewangk functions, an advantage that can be removed via rotation. Certainly we see this for first generation and the latter part of the Rastrigin runs in that SLEM is able to converge a little more closely to the global optima, as shown in Fig. 5.38 on page 128. However, for the portions of the runs where the population converges the most quickly it is the randomly rotated Rastrigin that does so. Interestingly the same thing occurs for the Griewangk, although that occurs for most of the run — i.e., with the exception of a few generations, the randomly rotated Griewangk runs converge a little closer to the global optimum. Note, though, that the differences are small, but nonetheless are statistically significant.

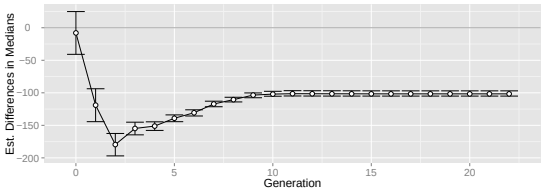
The random rotations may be presenting fitness landscapes by happenstance upon which the machine learner gains more traction. It could very well be the case that other rotations may actually impair the machine learner's performance. Or, it could be that the rotation consistently transforms the problem space into one from which it is easier for the ML to learn even it is for just a few generations, as in the case of the Rastrigin, or for most of the generations as was the case for the Griewangk.

5.4 Combined Effects of Fitness Proportion Approaches and Configuration Strategies

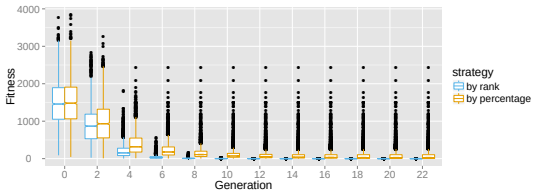
In the previous sections I examined three different training set configurations, *gap*, *middle*, and *split*. Though these observations were made first by fitness-rank and then by fitness-percentage training set approaches, there are still some general statements that can be made about those three configurations. That is, generally, the selection pressure increases from *middle*, to *split*, to *gap*; or $middle < split < gap$. However, practitioners will also want to know of the relative effects on selection pressure between the by-rank and by-percentage training set approaches, which we will look at next.



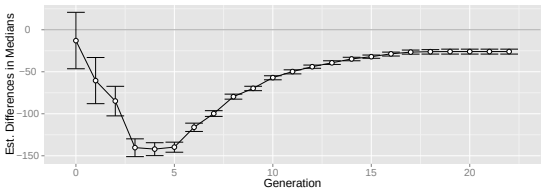
(a) This depicts the aggregate population trajectories for f_{sphe} for the by-fitness-rank and by-fitness-percentage runs using the *gap* strategy where we can see the by-fitness-rank runs converge more closely to the global optimum.



(b) This shows the estimated differences between the rank and percentage fitness approaches for creating training sets for f_{sphe} using the *gap* training set configuration. Values below zero favor by fitness rank runs, those above zero by fitness percentage. This corroborates Fig. 5.51a by showing that the by-fitness-rank runs have higher emergent selection pressure.



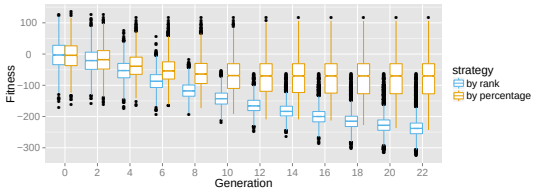
(c) This depicts the aggregate population trajectories for f_{sphe} for the by-fitness-rank and by-fitness-percentage runs using the *split* strategy, and similarly shows the by-fitness-rank runs converge more aggressively towards the global optimum.



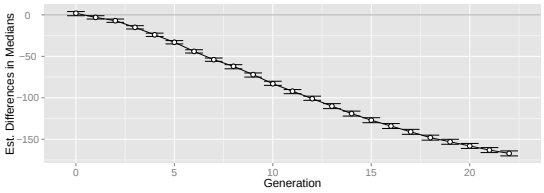
(d) This shows the estimated difference between the rank and percentage fitness approaches for creating training sets for f_{sphe} using the *split* training set configuration. Values below zero favor by fitness rank runs, those above zero by fitness percentage. This shows the differences between the by-fitness-rank and by-fitness-percentage runs are statistically significantly different.

Figure 5.51: This compares the by-rank and by-percentage of fitness runs for f_{sphe} using the *gap* and *split* training set configurations, which shows that the by-rank runs converged more quickly to the global optimum throughout all the runs.

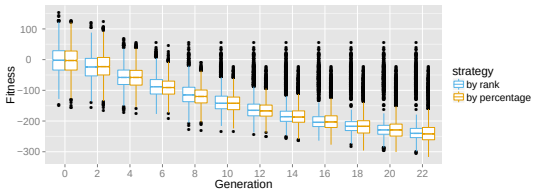
Fig. 5.51 shows the results between the by-fitness rank and by-fitness percentage runs using the *gap* and *split* training set configurations for f_{sphe}. There we can see that, regardless of the training set configuration, the by-fitness-rank runs converged more quickly towards the global optimum than the by-fitness-percentage runs. For the last generation the by-rank runs are significantly closer to the global optimum than the by-percentage runs (Wilcoxon rank sum p $< 0.001$for both sets of comparisons). The Rastrigin and Griewangk functions tell the same story — that the by-fitness-rank approach has higher selection pressure than the by-fitness-percentage. For brevity, the corresponding plots for these functions are found in Appendix B.



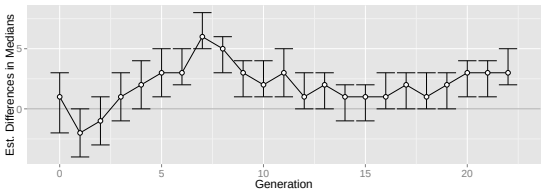
(a) This depicts the aggregate population trajectories for f_{step} for the by-fitness-rank and by-fitness-percentage runs using the *gap* strategy. The by-rank is able to continue converging towards the global optimum at $-\infty$ whereas the by-percentage runs converge towards an implicit asymptote typical of f_{step} runs.



(b) This shows the corresponding differences between the rank and percentage fitness approaches for Fig. 5.52a. Values below zero favor by fitness rank runs, those above zero by fitness percentage. This corroborates what is clearly shown in the previous figure, that the by-fitness-rank is able to continue converging without an implicit asymptote, whereas the *gap* configuration in the by-fitness-percentage runs converges to the asymptote and makes no further progress.



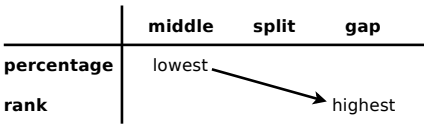
(c) This depicts the aggregate population trajectories for f_{step} for the by-fitness-rank and by-fitness-percentage runs using the *split* strategy. This shows that both the by-fitness-rank and by-fitness-percentage trajectories are similar.



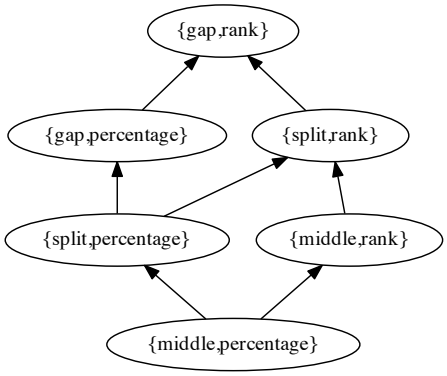
(d) This shows the differences between the rank and percentage fitness approaches for creating training sets for f_{step} using the *split* training set configuration. Values below zero favor by fitness rank runs, those above zero by fitness percentage. The trajectories are statistically similar at the beginning and at generations 13, 15-17, and 19. For the other generations the by-fitness-percentage runs converged slightly closer to the global optimum.

Figure 5.52: This compares the by-rank and by-percentage of fitness runs for f_{step} using the *gap* and *split* training set configurations, which shows that the by-rank runs converged more quickly to the global optimum for the *gap* configuration throughout all the runs, and that the by-percentage runs sometimes was sporadically slightly closer to the global optimum for some generations.

We might expect that the opposite would have occurred. That is, because the by-fitness-percentage approach has training sets that, in a sense, adapt to the underlying fitness landscape that the learned rules would have higher fidelity, which would, in turn, lead to better offspring. The f_{step}, the results of which are shown in Fig. 5.52, was the only fitness function where this was true — that is the f_{step} *split* runs that used by-fitness-percentage training set approaches converged more quickly than the by-fitness-rank approach, and



(a) This shows that the selection pressure increases from by-percentage to by-rank training set approaches; similarly the selection pressure increases from *middle* to *split* to *gap* training set configurations. The least selection pressure uses the by-percentage and *middle* training set configurations, and the highest selection pressure uses by-rank and *gap*.



(b) This depicts the poset describing the relationship between training set configurations. One can traverse the graph from the node with the least selection pressure, {middle, percentage}, to the end node with the highest selection pressure, {gap, by-rank} with the selection pressure increasing as one moves along the graph in the direction of the arrows.

Figure 5.53: This depicts the relationship of selection pressure between different training set configurations. The selection pressure is weakest using a by-fitness-percentage approach and *middle* training set configuration, and is strongest using a by-fitness-rank approach and the *gap* strategy. The relationship between these different configurations can be represented as a partially ordered set, as shown in Fig. 5.53b.

that is the only set of runs where that happened (last Generation Wilcoxon comparison $p < 0.001$). This exception gives insight into why the selection pressure for the by rank approach was higher than by percentage.

What is different about the f_{step} runs? We note that the corresponding training set sizes for the by-fitness-percentage approach for f_{step} were roughly the same, as shown in Fig. 5.31 on page 122, whereas for the spheroid, Rastrigin, and Griewangk functions the training set sizes were heavily skewed in favor of the "best" training sets with correspondingly anemic "worst" training sets. (As depicted in Figs. 5.28, 5.34, 5.37, 5.41, 5.44 on pages 119, 125, 127, 130, 132, respectively.) So it was this skew that likely impeded the progress for almost all the by-fitness-percentage runs.

In any case, with the aforementioned exception of f_{step}, in general it is the by-rank

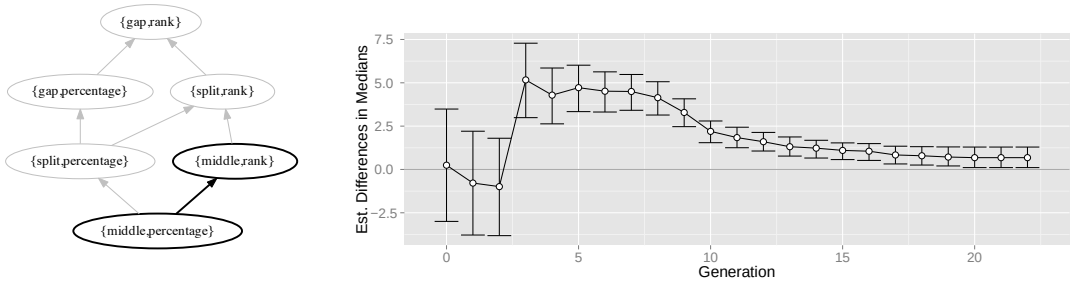


Figure 5.54: This shows the 95% confidence intervals for the by generation differences between medians for the {middle, percentage} → {middle, rank} runs, which are highlighted in the Hasse diagram on the left, using the f_{rot_grie} test function. Values above zero favor the {middle, rank} runs with regards to selection pressure; so this shows that the {middle, rank} training set configurations had higher selection pressure than {middle, percentage}.

approach that has higher selection pressure than the by-percentage, or by-percentage $<$ by-rank.

Now that we have an independent understanding of the effects on selection pressure of the three training set configurations and two proportion approaches, we next need to combine the two to get an overall sense of the impact of these different training set design decisions has on selection pressure. Fig. 5.53 shows a diagram similar to the one in §4.4 on page 85. That is, the first subfigure, Fig. 5.53a, shows a table that defines a gradient of selection pressure that increases from the upper left to the lower right depending on what training set configuration design choices are made, and based on the results discussed earlier. From that table we can define a set of partial orderings of steadily increasing selection pressure, which is depicted in the Hasse diagram in Fig. 5.53b. That diagram shows that we can start with the design decision corresponding to the weakest selection pressure, {middle, percentage}, and then steadily increase the selection pressure by changing either the training set configuration or proportion approach until we arrive at the maximum selection pressure criteria, {gap, by-rank}.

We have previously seen empirical support for all the partial orderings described in Fig. 5.53b — that is, except for the ordering {middle, percentage} → {middle, rank}. For the same reasons given in §4.4 on page 85 we can use the rotated Griewangk function as

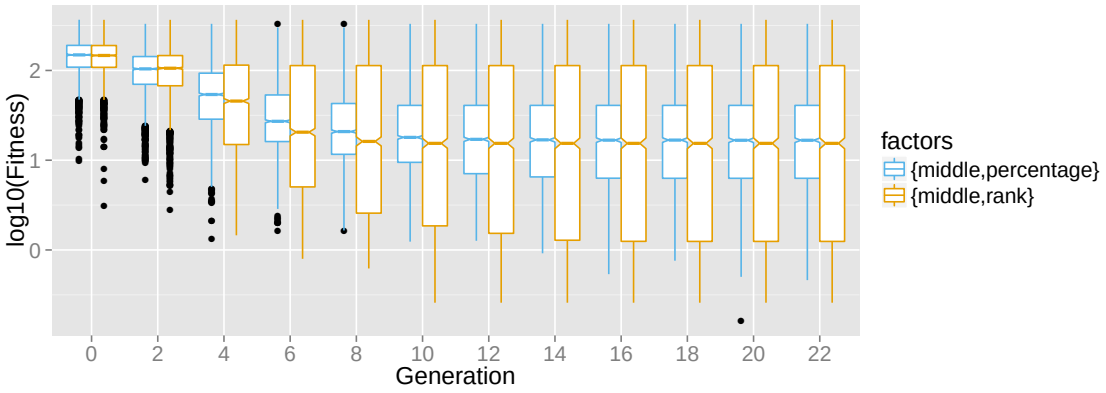


Figure 5.55: This shows the log transformed population trajectories for {middle, percentage} and {middle, rank} with log adjusted fitnesses using f_{rot_grie}, and which shows that the {middle, rank} runs converge more closely to the global optimum than {middle, percentage}.

representative of real world problems practitioners may encounter. Fig. 5.54 shows the by generation comparison between the {middle, percentage} $\rightarrow$ {middle, rank} runs for f_{rot_grie}, and shows that the selection pressure does appear to increase as predicted by the poset. (The last generation of the {middle,rank} runs are closer to the global optima, Wilcoxon rank sum test $p < 0.001$.) However, there is a caveat. The population median is used a representative statistic for comparison between the runs, and though this shows that the population medians for the {middle, rank} runs are closer to the global optimum than the {middle, percentage} runs, the population distributions overlap as shown in Fig. 5.55. So we must be cautious before relying solely on comparing population medians to draw conclusions.

If anything that the population variance for the {middle, percentage} is smaller is interesting. This suggests that the ML is, as expected, able to learn rules that allow for more consistent targeting of optima; however, as related earlier, the skewed ratios between the "best" and "worst" still hamper progress as measured by comparing the medians.

In any case, practitioners now have some guidance with regards to the influence training set configurations have on machine learner induced selection pressure. By using the Hasse diagram shown in Fig. 5.53 to navigate by one can increase or decrease selection pressure

by iteratively changing the training set configuration.

Similar to the combined effects for rule interval sampling, in future work we could go further than defining partial orderings between training set configurations, but determine their relative strengths as well as gaining deeper insight into their respective behavior.

5.5 Conclusions

In this chapter I covered the influence on SLEM's emergent selection pressure via training set configurations where the "best" and "worst" parents were selected by rank or by fitness percentage on a variety of fitness functions. Within these two approaches I examined three different training set configurations. The first training set configuration was the traditional *gap* configuration where the "best" and "worst" parents used as training examples are separated by a set of "mediocre" parents that are ignored for purposes of teaching the ML; as is typical for this configuration, I evenly split the population into thirds for each of these subsets of the parent populations. The second training set configuration was where the positive and negative examples were evenly *split* between all parents. And, in the third configuration, inspired from previous research on training set configurations, I modified the *split* configuration by removing enough of the extrema parents to make the "best" and "worst" training sets the same size as those found in the *gap* configuration.

I will next go over the conclusions first for the by-fitness-rank results, then the by-fitness-percentage, and then finally look at general concepts that combine both.

5.5.1 Training Sets By Fitness Rank

We learned that, generally, the *gap* converged a little more quickly than the *split* training set configuration. However, the differences were fairly small and there was a great deal of overlap between population distributions; so practitioners should be cautious about reading too much into these differences especially given that SLEM is not a complete EA since legacy selection and birth operators were absent. In any case, the gap of "mediocre" parents gave the ML a somewhat clearer perspective on the underlying fitness landscape, a perspective

that ultimately yielded slightly better quality offspring.

Of note, though, is that the *gap* configuration for f_{sphe} initially was a little more aggressive converging towards the global optimal as with other test problems, but once the population had converged it was the *split* that managed to converge closer to the optimum a very small, but still statistically significant, distance. This was due to the population being homogeneous enough that the larger training set size of *split* was enough to push it a small way further towards the global optimum.

Of particular interest was the difference between the *gap* and *split* training set configurations for f_{step}. There the *split* and *gap* alternated dominance with regards to convergence velocity towards to global optima due to the influence of the implicit bounds dictated by the initial set of parents as described in §4.3.2 on page 75.

We also observed that for all the test problems the *middle* training set configuration consistently converged further from the global optima than *gap* and *split*. The *middle* configuration did not exhibit the "tortoise vs. hare" effects Walling and Ryan observed whereby the *middle* runs would at the end escape local optima to superior solutions [Wallin and Ryan, 2009]. This was due to my use of non-overlapping generations to eliminate survivor selection bias, which removed the very best parents that would otherwise likely persist from generation to generation and where they would possibly serve as an escape route from local optima.

We also learned via the Langerman test problem that increasing training set size had no impact on performance regardless the training set configuration since proportion of basins of attraction will scale with population size. This is particularly a problem if local optima have comparative large basins of attraction as compared to the global optima.

Randomly rotating the Rastrigin function did appear to somewhat hamper its effectiveness at exploration as witnessed by Fig. 5.14 on page 104. Surprisingly random rotation appears to help SLEM on the Greiwangk function when using the *split* configuration.

5.5.2 Training Sets By Fitness Percentage

As might be expected when creating training sets based on percentage of fitness values — as opposed to by fitness rank — is that they will vary in size. Not only will the proportions of actual training set sizes differ from the user specified proportions, but the training set sizes will change over the course of a run. Moreover the way the sizes change over time depends on the problem and on the training set configuration. Even the first generation, which reflects a uniform distribution of the problem space, will have relative training set sizes of "best" and "worst" training instances that reflect the problem defined topology.

For the most part there were more individuals in the "best" training set than in the "worst", which was chiefly a reflection of landscape topology. This ratio was reflected in the f_{sphe}, f_{rast}, and f_{grie} functions, which all have similar "bowl-like" characteristics; whereas where this ratio of "best" vs. "worst" sizes was not true, namely in f_{step} and f_{lang}, had dissimilar topologies that were distinctly not bowl-like. The f_{step} is essentially an infinite stepped sloped plane, whereas the f_{lang} is an extremely complex, jumbled terrain with a very large basin of attraction for local minima. In the bowl-like topologies the gap between the "best" and "worst" is at its smallest at the start of runs, but gradually widens until the it stabilizes around the time the population has converged.

Why is the ratio of "best" and "worst" training set sizes important to practitioners? Because practitioners are interested in ensuring that there is adequate exploration, and if the ratio of one training set is too skewed then that might suggest that some tuning of the "best" vs. "worst" percentages is necessary. For example, where there are far more "best" than "worst" individuals for training the machine learner may be a form of greediness; and in those scenarios if the fitness landscape is particularly complex, the practitioner may wish to reorient the percentages in favor of more "worst" individuals to allow for better exploration.

If we accept this notion of significantly more "best" over "worst" individuals as a form of greediness, then using the *middle* configuration is an interesting and unique approach of attenuating that greediness. We already had a hint that this was so with the original work

done by Wallin and Ryan in that the *middle* was able to converge to better optima over alternative training set configurations on certain problems [Wallin and Ryan, 2009]. That is, if we are faced with a complex optimization problem we want to reduce greediness to increase exploration to prevent the system from prematurely converging on inferior solutions, and their use of the *middle* configuration exhibited that very characteristic. I had already suggested directly reducing the percentages of the "best" to reduce greediness, but with the *middle* configuration the "best" *percentages* can be the same, but will still have the effect of reducing the count of "best" parents used for training.

So I have discussed the *middle* training set configuration — what of the other two configurations, *gap* and *split*? In previous results we observed that for the spheroid the emergent selection pressure was in proportion to the training set size [Coletti, 2012], and to an extent that is still true. For the end of the runs the *split* is able to converge a little more closely to the global optima than the *gap* configuration runs. This is because once the population is too homogeneous, then there is more advantage to having larger training sets over having a "mediocre" gap that is no longer necessary because the "best" and "worst" are too similar. However, this is not always the case. For the rotated fitness functions, f_{rot_rast} and f_{rot_grie}, there were no significant differences at the end of runs between the *gap* and *split* configurations, which is indicative of the added complexity introduced via rotation.

Of interest is what happens at the beginning of the runs with regards to the *gap* and *split*. For the f_{sphe} and both Rastrigin functions the *gap* configuration is a little more aggressive than the *split* configuration runs until the convergence rate slows down once the "best" and "worst" populations become too homogeneous; whereas the *split* runs have overlapping positive and negative examples which, in turn, lead to lower quality rules which is reflected in the offspring. The Greiwangk functions are like the Rastrigin, but that run pattern is preceded with the *split* configuration being a little more aggressive.

Generally, the *gap* will converge more quickly if there is not only minimal overlap between the "best" and "worst", but also that this disjoint distribution is meaningful; i.e.,

there may be scenarios whereby the "best" and "worst" do not have much overlap, but because of the complexity of the fitness landscape, the ML still cannot learn anything meaningful. Generally the *split* can possibly converge a little further than *gap* training set configuration in scenarios where the larger training set sizes can compensate for homogeneous populations typical of runs that have converged close to an optima.

However, I must again caution that even in cases where *split* is demonstrably greedier than *gap* that sometimes the differences are still small, and that the two populations have a great deal of overlap. So the practitioner should not be too quick to conclude that *split* is going to have significantly higher selection pressure than the *gap* configuration.

The by-fitness-percentage approach made little difference with regards to SLEM's performance on the Langerman function — the previously observed large basin of attraction still drew in the population. The only difference was that the ML did manage to sometimes learn rules in the first generation, but the generated offspring from those rules were still within the strong basin of attraction of the local optima.

With regards to rotating the Rastrigin and Greiwangk functions, we note a few things. First SLEM is able to marginally approach the global optimum a little more closely with the non-rotated Rastrigin function, which suggests that the ML is gaining some leverage from the original rectilinear arrangement of optima. Second that the convergence rate is higher at the beginning of runs for the rotated functions, particularly for the rotated Rastrigin. Third, the rotated Griewangk runs are able to marginally approach the global optimum throughout the entire runs, though admittedly by a small amount. It could be that increase in selection pressure due to rotation may be due to happenstance; i.e., that the random rotation arbitrarily transformed the fitness landscapes in such as way to give more leverage to the machine learner, PART.

5.5.3 Rank vs. Percentage

We learned that the by-fitness-rank training set approach had relatively higher selection pressure than the by-fitness-percentage approach for all the test functions except for f_{step}.

(f_{lang} was not considered since SLEM was unable to make progress beyond the first generation regardless the training set approach.) That the f_{step} was the sole exception to this pattern offers a clue as to why.

That is, the ratio of "best" to "worst" was significantly skewed in the "best" training set's favor for all but the f_{step} function, which consequently damaged the fidelity of any learned rules in the non-f_{step} runs; this, in turn, lead to the by-fitness-percentage approach generating lower quality offspring. However, this suggests that the by-fitness-percentage approach selection pressure could be increased by adjusting the "best" and "worst" percentages such that their respective counts are more in parity.

5.5.4 Combined Effects of Rank vs. Percentage and Training Set Configuration

Finally, I showed the overall influence of using by-fitness-rank or by-fitness-percentage as well as the training set configurations of *middle*, *split*, and *gap* on the machine learner induced selection pressure. The weakest selection pressure is for the training set attributes {middle,by-fitness-percentage} and the strongest by {gap,by-fitness-rank}. One can gradually increase the selection pressure by either moving from by-fitness-percentage to by-fitness-rank as well as traversing from *middle* to *split* to *gap* as depicted by the partial orderings shown in Fig. 5.53 on page 144.

In the next chapter I discuss the combined effects of both rule interval sampling and training set configurations on selection pressure.

Chapter 6: Combined Effects of Rule Interval Sampling and Training Set Configurations

The previous two chapters explored the effects on the emergent selection pressure derived from using learned rules of two types of design decisions: those involving approaches to sampling rule intervals and those using different training set configurations. In this chapter I will examine at a high level design decisions unifying those two approaches.

6.1 Introduction

So far we have learned that design decisions regarding sampling rule intervals and assembling training sets in certain ways has an effect on selection pressure. We learned in Ch. 4 that selection pressure increased when going from clamping unbounded rule intervals from the most extreme gene value to only the most extreme value from the "best" parents. We also learned that, in general, the selection pressure increased when going from a uniform, to histogram-based, to Gaussian distributions to sample rule intervals. Moreover, we discovered that these principles could be combined into a partially ordered set of design configurations that practitioners could use to navigate design decisions when assembling these kinds of EA/ML hybrids. Furthermore, we learned in Ch. 5 that, overall, selection pressure increased when moving from using training sets alloted by fitness percentage to those alloted by fitness rank; similarly selection pressure generally increased when moving from the *middle* to *split* and then to *gap* training set configurations. And, as with the rule interval sampling approaches, training set configuration guidance with regards to selection pressure could be represented as a partially ordered set.

However useful this knowledge is, though, practitioners will have to not only decide on optimal means of sampling rule intervals, but also how to set up training sets given to the

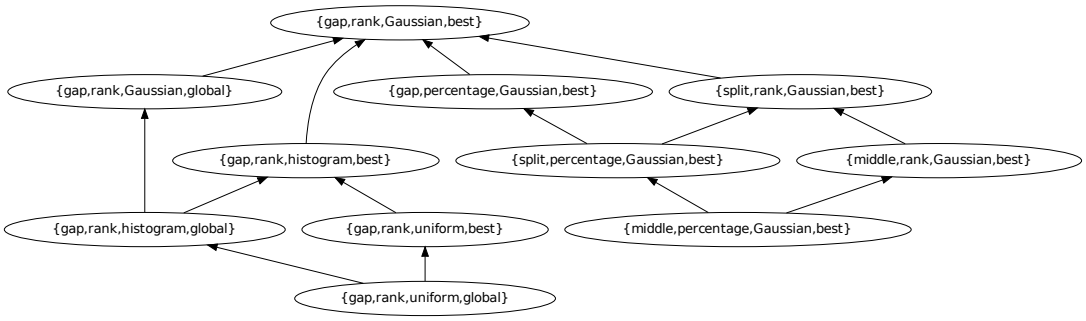


Figure 6.1: This shows the union between the two posets that define the relative selection pressure strengths for various SLEM design configurations from Ch. 4 and Ch. 5. The common configuration between the two Hasse diagrams is {gap,rank,Gaussian,best}.

machine learner. These design decisions are not made independent of one another. The focus of the next section is to combine these two areas to provide practitioners with a basic road map to guide them when implementing these kinds of hybrid systems.

6.2 Combined Effects of Rule Interval Sampling and Training Set Configurations

The rule interval sampling designs and training set configuration posets representing the relative selection pressure intensities can be merged into a single partially ordered set. All the rule interval runs in Ch. 4 used the *gap* training set configuration with a by-fitness-rank setting; and all the training set configuration runs in Ch. 5 used a Gaussian distribution with a scale factor of three standard deviations — also, unbounded rule intervals were filled with appropriate extrema from the "best" parents. So the next step is to visualize a poset combining the partially ordered sets of design decisions from those chapters being mindful of the design relationships for which we have no supporting evidence yet.

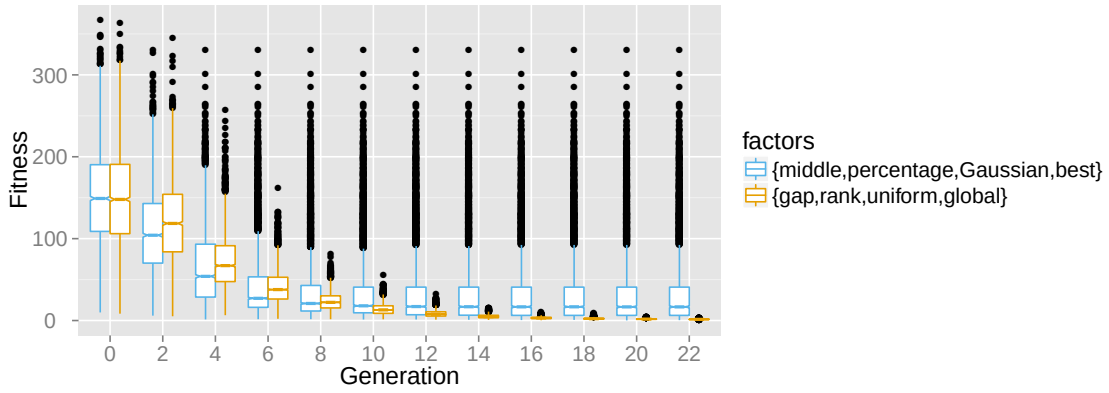
Let us consider blending the attributes from Ch. 4 and Ch. 5 into one partially ordered set of selection pressure related design decisions. From Ch. 4 we would consider the attributes for rule interval sampling distribution and unbounded rule interval closing; we

would similarly consider the attributes for training set configuration and proportion approach from Ch. 5. So the tuples for the new poset would be of the form {configuration, proportion, distribution, unbounded}. Fig. 6.1 depicts the union of the posets as shown in Figs. 4.33b and 5.53b on pages 86 and 144, respectively. From this new poset we can observe that the configuration with the highest selection pressure corresponds to the attribute tuple {gap, rank, Gaussian, best}, which is the configuration that is in common between the two chapters.

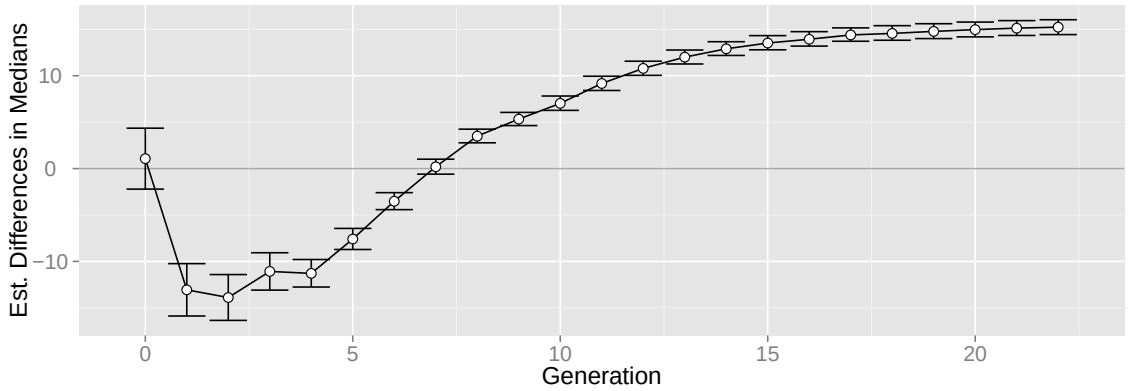
Unfortunately, this is unsatisfying from a practitioner's perspective in that the weakest attribute tuple is unknown — all design decisions in the current diagram must start from the strongest known configuration from which the practitioner must then choose from four possibilities if they desire to reduce selection pressure. It may be, for some, more convenient to start from the configuration with the weakest selection pressure and gradually increase the pressure through a variety of sensitivity tests for their problem of interest.

We can identify candidates for configuration with the weakest selection pressure by traversing within this poset from the node with the strongest selection pressure, {gap, rank, Gaussian, best}, down the graph to arrive at the nodes with the weakest selection pressure, namely {middle, percentage, Gaussian, best} and {gap, rank, uniform, global}. Unfortunately these two nodes define an antichain in that there is no means of directly comparing them within the current partially ordered set.

However, we can do a comparison between these two sets of runs to empirically determine which of the two has the least selection pressure. Fig. 6.2 shows the population trajectories between these two sets of runs for the f_{rot_grie} test function — as established in the preceding two chapters, I chose the rotated Griewangk function as most representative of the types of problems that might interest practitioners. The figure shows that for the first several generations the {middle, percentage, Gaussian, best} runs converge more closely to the global optimum, but after the ninth generation not only does the {gap, rank, uniform, global} do so, but also continues to widen the gap between the two runs. For the last generation, the {gap, rank, uniform, global} ends being significantly closest to the global



(a) This shows the respective population trajectories for runs using {middle, percentage, Gaussian, best} and {gap, rank, uniform, global} for f_{rot_grie}. The {middle, percentage, Gaussian, best} runs initially converge more quickly than {gap, rank, uniform, global}, but then the latter then converges more quickly from generation nine forward.



(b) This shows the corresponding by-generation differences between the {gap, rank, uniform, global} and {middle, percentage, Gaussian, best} f_{rot_grie} runs with 95% confidence intervals for the estimation of the difference between the respective population medians. Values below zero favor {middle, percentage, Gaussian, best} and those above {gap, rank, uniform, global}. Here we can see that for the first several generations {middle, percentage, Gaussian, best} does converge more quickly; but after the ninth generation not only do the {gap, rank, uniform, global} runs then converge more closely to the global optimum, but continue to distance themselves from the {middle, percentage, Gaussian, best} runs through the last generation.

Figure 6.2: These figures compare runs using {middle, percentage, Gaussian, best} and {gap, rank, uniform, global} configurations for f_{rot_grie}, and shows that for the first several generations {middle, percentage, Gaussian, best} runs converge more closely to the global optima to then switch to {gap, rank, uniform, global} being the more aggressive.

optimum (Wilcoxon rank sum test $p < 0.001$).

One converges more quickly at the start, but is then overtaken by the other configuration for the rest of the run, which is classic "tortoise vs. the hare" behavior between a greedy

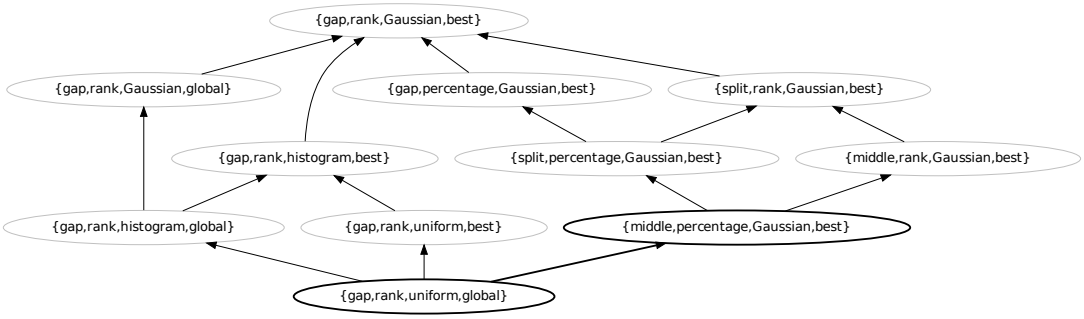


Figure 6.3: This shows the partially ordered set of design configurations updated to reflect that we have empirically determined that the {middle, percentage, Gaussian, best} configuration has higher selection pressure than {gap, rank, uniform, global}.

approach and one that is less greedy. That is, the greedy approach, in this case the one with the highest selection pressure, quickly finds a choice local optima and converges on that, whereas the less greedy approach is free to explore longer to increase the likelihood of finding an even better optima to then exploit. Given this new information, we can now update the merged poset to reflect this new information, which is shown in Fig. 6.3.

For some practitioners the former is important in that they may be solving a problem with a very limited birth budget due to inherent fitness function complexity, so would prefer configurations that converged quickly within several generations; still other practitioners may be willing to wait for better answers, and so would not be constrained by small birth budgets. So the former would likely prefer the {gap, rank, uniform, global} configuration, whereas the latter {middle, percentage, Gaussian, best} should they wish to start with the weakest selection pressure configurations. They could then implement a series of sensitivity tests by gradually increase selection pressure by traversing along the graph in Fig. 6.3 until finding a "sweet spot" appropriate for a given problem.

However, this advice comes with a caveat: all experiments done in this work intentionally suppressed legacy sources of selection pressure to study the selection pressure originating from applying learned rules when creating offspring. Therefore these configurations reflect that selection pressure and so the partial ordering between configurations may change once

other sources of selection pressure are restored. Further work is needed to study the impact of legacy parent and survival selection operators on these types of EA/ML hybrid systems. Nonetheless, the results shared here fill an enormous vacuum with regards to identifying design options and posing some basic guidance for practitioners wishing to implement these systems.

6.3 Summary

In this chapter I combined the results from the preceding two chapters — the first, Ch. 4, addressed the impact on rule interval sampling approaches on ML-based selection pressure; and the second, Ch. 5, focused on how different training set configurations affected this selection pressure. Each of these chapters defined a partially ordered set of design decisions, design decisions differentiated by relative selection pressure. In this chapter I merged these posets into a single poset where I observed there was a single common design configuration between the two original partially ordered sets, and that this design configuration had the highest selection pressure.

However, this poset did not have a single design configuration denoting the weakest selection pressure. To empirically find such a configuration I compared runs corresponding to the two design configurations with the least selection pressure that were antichains in the initial version of the merged partially ordered set. From the results I was then able to determine which of the two design configurations exhibited higher selection pressure, and was then able to update the merged poset such that there was now a single design configuration corresponding to the weakest selection pressure.

In the next chapter, I summarize the overall contributions of my work here as well as outline future work.

Chapter 7: Contributions and Future Work

In this work I have focused on the effects on the unique emergent selection pressure that manifests when an EA's offspring are created using rules learned from the best and least fit parents. To this end, I created a simplified version of Learnable Evolution Model (LEM) to expedite analysis and serve as a means for future research; I also developed two visualization methods to make analysis of results easier. I examined the impact that uniform, Gaussian, and a histogram-based rule interval sampling methods have on the aforementioned selection pressure. I also observed that occasionally the machine learner would learn only a single interval bound, and so I also analyzed three different approaches for resolving those unbounded intervals. And, finally, I looked at three training set configurations by fitness rank and by percentage of overall fitness values.

This chapter summarizes my contributions derived from that research as well as describing areas of related future exploration.

7.1 Contributions

Here I summarize the contributions I made with this research. These contributions include a new algorithm, novel data visualization techniques, as well as the outcomes of the analysis of the influence of rule interval sampling methods and training set configurations on the system's selection pressure.

7.1.1 Simplified Learnable Evolution Model

The Simplified Learnable Evolution Model (SLEM) was my vehicle for exploration. Again, it was necessary to pare down Learnable Evolution Model (LEM) to something that was analytically tractable, and that is what SLEM provided. However, as I will relate in §7.2,

there is much work left to do to better understand these types of EA/ML hybrids, and SLEM remains the optimal means of exploring those issues. Moreover, SLEM may be a useful addition to a practitioner's EA problem solving toolkit as it has demonstrated problem solving potential in its own right.

7.1.2 Novel data visualization techniques

I created two data visualization techniques to serve as analytical aids. The first technique involves comparing a pair of EA runs using a series of Wilcoxon statistical tests where I show the Wilcoxon rank sum test 95% confidence intervals for each generation shaded in proportion to the corresponding p-value. This visualization compares behavior throughout the entirely of runs showing phenomena that might otherwise be missed if more traditional end-of-run and convergence-time behaviors were the only probe points. This technique should be readily applicable to other EA pairwise run comparisons. The second data visualization technique uses heat maps to portray frequency data on a by generation basis. This was used to depict the number of rules learned as well as the number of genes referred to by any learned rules for each generation. This technique could be applied to other EA by-generation frequency data.

7.1.3 Resolving unbounded rule intervals

One of the first problems a practitioner may encounter when assembling one of these EA/ML hybrids is that the ML may only learn one bound for a given rule interval. Given that we need an upper and lower bound to sample within an interval when creating offspring, we required a way of filling in the missing interval bound. I explored three ways of supplying missing rule interval bounds. The first approach, which I called the *init* strategy, would fill in the missing bound value from the appropriate initialization values; i.e., a missing upper bound would get the largest bound value, and missing lower bounds the smallest. The *global* approach would replace missing bounds with the largest or smallest gene value from entire population; the *best* strategy would do the same, but only from the fittest parents.

I found that the *init* approach had a disruptive effect on the population trajectories. As a population began to converge, inevitably one of the rule interval bounds would be "reset" to the initialization bounds. This meant, in turn, that offspring would then more closely resemble the initial parent population, thus moving the population away, instead of towards, an optima.

In general the *best* strategy would converge more quickly to the global optima than the *global* strategy, which may be expected given that in that approach unbounded rule intervals are filled from only the best individuals. However, this was not the case for the step function test problem. There, it was the *global* that converged more quickly to the global optima. This was due to the system never leaving the exploration phase of search — the larger areas sampled by the *global* strategy would make it likelier that lower "steps" in the step function would be discovered.

The *best* strategy did not fare any better or worse than *global* for the Langerman function because the initial population was mostly in a very large basin of attraction for local optima from which the population never escaped. I will talk more about the Langerman function shortly.

7.1.4 Initial population defines implicit bounds

I showed that the initial population defines an implicit bounds from which subsequent generations would be unable to escape. That is, if the global optima is outside the bounds of the initial population, then the trajectories will asymptotically approach — but not exceed — the limits to which that population was initialized. However, it is possible to escape that bounds if the rule intervals are sampled within a Gaussian distribution. Even then that is no guarantee that population trajectories will be able to explore beyond that initial bounds. I showed that a narrow Gaussian distribution would not only fail to penetrate this implicit boundary, but demonstrated a more conservative implicit bounds than that dictated by the initial population.

This is important for practitioners to know since it is sometimes the case we are unaware

of where the global optima is located, so the initialization boundaries are a guess as to its whereabouts. We would hope that an evolutionary algorithm is free to explore beyond those boundaries in pursuit of better answers, but I have shown here that unless precautions are made that LEM-like systems will fail to do so. This is particularly relevant given that legacy LEM implementations use a form of global crossover to fill in offspring gene values for genes that are not cited by rules. EA theory tells us that crossover operators explore within the space dictated by the current population, not outside. So this traditional LEM configuration will be unable to help the population move beyond the implicit bounds defined by the initial population.

7.1.5 Rule interval sampling distributions had impact on emergent selection pressure

I examined the influence of sampling rule intervals using three different distributions: that is I explored uniform, Gaussian, and histogram-based probability distributions. The experiments using the uniform distributions exhibited the least amount of selection pressure. I found that the histogram-based distributions could have their greediness modulated by controlling the amount of a uniform distribution blended in — essentially the more of a contribution the uniform distribution made the less selection pressure. The Gaussian distributions that sampled rule intervals within three standard deviations had the highest apparent selection pressure.

The one notable exception with regards to using Gaussian distributions of three standard deviations was with the step function. Again, for that one scenario the Gaussian distributions of two standard deviations were able to converge the most quickly, but narrowing the distribution greatly reduced the selection pressure.

7.1.6 Influence of training set configuration

I covered the impact on SLEM's emergent selection pressure from machine learner training set related design decisions. Specifically I examined two broad training set approaches: that

of taking the very best and least parents by rank, and the best and least by percentage of overall parent population fitness. Within these two approaches I further examined three training set configurations: one configuration denoted as the *gap* configuration which was comprised of the top one third as the "best" and the bottom one third as the "worst"; another where the top and bottom-most parents were evenly divided, known as the *split* configuration; and yet another where the population was evenly split, but with one third of the extrema ignored by the ML known as the *middle* configuration.

We learned that, in general, in the by-fitness-percentage training set approach that the "best" and "worst" training set sizes were heavily skewed in the "best" set's favor. This, in turn, meant that the by-fitness-rank approach had higher selection pressure than the by-fitness-percentage since this skewed ratio impaired the ML's learning effectiveness. The one notable exception with regards to the by-fitness-rank having higher selection pressure was with the f_{step} function, but there we observed that there was more-or-less parity between the "best" and "worst" training set sizes.

We also observed a general pattern between the *gap* and *split* configuration runs: that at the end of the runs the *gap* tended to converge more closely to the global optima for the by-fitness-rank approach, whereas it was the *split* configuration that did so in the by-fitness-percentage approach. Why does this happen? The by-fitness-percentage approach better reflects the underlying problem distribution, thus mitigating the need for manually dividing the "best" and "worst" training sets with an intermediate "mediocre" set; so, in that case, having more trainings instances would be better, which is what the *split* configuration does over the *gap* configuration. Conversely, the by-fitness-rank approach does a poorer job of matching the underlying topology, so it is there that it is more important to intervene to ensure minimal overlap between the "best" and "worst" to all the better approach the global optima.

As to the behavior of the runs prior to convergence, the *gap* configuration sometimes exhibited higher selection pressure. This happens when the population is heterogeneous enough such that the positive and negative training sets are sufficiently disjoint that the

ML can be more effective. However, sometimes the *split* configuration had higher selection pressure at the very beginning, as in the case of the Greiwangk functions; so just because the parent population is heterogeneous, and thus the positive and negative examples fairly non-overlapping, is no guarantee that the *gap* configuration is the more greedy. In those scenarios the additional training instances afforded the *split* may give the ML enough traction to move the population along to a particular optima while the *gap* configuration is still sorting itself out.

7.1.7 Validation of novel exploration approach

The *middle* configuration was originally designed by Wallin and Ryan to test the necessity of a "mediocre" set of individuals [Wallin and Ryan, 2009] — i.e., the set of parents dividing the "best" and "worst" in the *gap* configuration. What they found was that the *middle* was able to converge to better solutions than the *gap* or *split*. They speculated that the very best parents that were just outside the "best" training set given to the ML served as a sort of insurance policy to escape local optima. I have provided corroborating evidence they were correct. That is, my *middle* configuration runs were unable to escape such local optima because I discarded all parents with each generation, including the very best parents in the extrema just outside the "best" training set used by the ML.

7.1.8 Large basins of attraction for local optima trap populations

We also learned that the Langerman function posed a particularly difficult challenge. Regardless of which training set configuration that was used, the Langerman function had an enormous basin of attraction for local optima well away from the global optimum. We observed that increasing the training set size merely scaled the sampling of these local optima from which the machine learner gained no benefit.

7.1.9 Mixed perspective on randomly rotated landscapes

The Rastrigin and Greiwangk functions have axially aligned minima that prove advantageous to some algorithms, an advantage that can be removed by rotating the fitness functions. When we did this for the Rastrigin functions, we observed a slight drop in convergence pressure, which hints that perhaps SLEM gains some small advantage from the regularly spaced, non-rotated optima. However, sometimes the convergence pressure increased when the Griewangk was rotated for both the by-fitness-rank and by-fitness-percentage runs, particularly the latter. This may have been due to happenstance as the ML gained some advantage through the randomly rotated Griewangk topology.

7.1.10 Combined Effects of Selection Pressure

I also provided high level guidance with regards to the impact on machine learner enduced selection pressure via a partial ordering of design decisions. Practitioners can use this poset to navigate design choices for rule interval sampling, closing unbounded rule intervals, and training set configurations to modulate selection pressure.

7.2 Future Work

In this section I enumerate possible future directions for research related to this work.

7.2.1 Persistent knowledge

In the implementation I used here any learned rules were discarded each generation and learned anew for the next. However, it might be beneficial to allow rules to persist for more than one generation. Indeed in prior work we learned that having rules linger for more than one generation had the effect of reducing the emergent selection pressure associated with the ML [Coletti, 2012]. Unfortunately that research focused on a single, simple fitness landscape, the spheroid, so this research could be extended to other fitness landscapes to enhance the generality of what we learned.

One issue with keeping rules longer than one generation is that the practitioner has a design decision to make regarding choosing rules when creating offspring. Do we use a naïve uniform selection where rules are chosen from the same pooled collection regardless of rule age or quality? One biased in some way by rule age? Or one biased by some aspect of rule quality? In prior work a linear rank approach was taken when biasing rule selection by age [Coletti, 2012], though tournament-style or other legacy selection operator could presumably also be used. Cultural Algorithms also have rules that persist; CAs employ a credit assignment system whereby rule quality is associated with the fitnesses of offspring in which they had an influence [Reynolds, 1999]. A similar system could be explored for LEM-like systems.

As mentioned in §4.2.1 on page 51, a previous LEM implementation, LEM/C4.5, had a form of persistence in that regions of higher fitness were stored as intervals associated with each gene [Coletti, 2002, Coletti, 2009, Coletti, 2012]. These genes would be inherited from parents, and updated from applied rules. The effect on selection pressure of this implementation has not been fully explored.

Rules are not the only things that can persist from generation to generation – individuals, too, can similarly persist. In this work, I used non-overlapping generations as a means of suppressing a source of selection pressure. However, the behavior of the system would certainly change if individuals were allowed to survive longer than a single generation. Moreover, we can get further inspiration from CAs and borrow the notion of "exemplars"; i.e., individuals that serve as examples in a kind of "hall of fame". This is a form of elitism where a subset of the best of the population persists irrespective of what survivor selection is used. Moreover the original LEM specification allowed for saving individuals in special ways to enhance the ML's perspective on the fitness landscape [Michalski, 2000].

7.2.2 Accounting for negative examples

Instead of exploiting knowledge of areas of higher fitness to improve offspring quality, we can perhaps use the converse and similarly exploit knowledge of areas of lower fitness. This

is not a novel idea in evolutionary computation. Prior work has been done in walling off low quality areas of search space during runs [Ravisé and Sebag, 1996, Sebag et al., 1996]. Moreover, the original LEM specification allows for variants that similarly compartmentalize areas from the "worst" individuals called *avoid-past-failures*; this entails a generate-and-test method whereby runs are restarted after no new progress is being made with new populations created entirely by individuals that do not match rules corresponding to the lowest performers [Michalski, 2000]. Though this is similar to the approach some memetic algorithms employ — that of generating and testing to create viable individuals as a form of local search [Krasnogor and Smith, 2005] — this is different in that an entire population is created at once using rules corresponding to the "worst" parents from the previous run.

The use of negative examples can also apply at the level of genes. For example, in the histogram-based rule interval sample implementation the bins can be accumulations of counts of gene values just from the "best" parents; alternatively bin counts could also be decremented by counts of corresponding "worst" parent gene values. Similarly, the unbounded rule interval approach that fills in missing interval values from the "best" parents could be finely tuned to accommodate possible overlapping values from the "worst". That is, there may not be a clear linearly separable set of "best" and "worst" gene values from which to choose extrema to fill in missing rule interval bounds; the approach I used in §4.2 naïvely assumes that such values are linearly separated, when that might not be the case. Therefore a more fine-grained approach that takes into consideration this mix of "best" and "worst" gene values might yield higher convergence rates.

7.2.3 Influence of ML induction bias

I use one machine learner, PART, in this work. In related work, LEM has been implemented using C4.5 [Coletti, 2002] and AQ [Cervone et al., 2000b, Wojtusiak, 2008]. The choice of ML when implementing a LEM-like system naturally introduces an associated induction bias in the overall system behavior. Moreover, most machine learners have tailorable run-time behavior that may further influence how the system functions. For example, most

decision-tree-based machine learners have parameters to tune tree pruning, which could have an impact on performance if used in a LEM-like setting. Exploring the influence of different induction biases from different machine learners and their respective run-time settings would be valuable to practitioners.

7.2.4 Alternative test problems

I selected a test suite that was representative of aspects of real world problems. These included the spheroid function, which is commonly used as a "pure" problem to observe algorithm convergence behavior; the step function for learning how the system deals with optima outside bounds defined by initial populations; the Rastrigin and Griewangk functions as common multi-modal real-valued optimization problems that can be made more challenging via random rotation; and the Langerman function with an enormous basin of attraction for local optima. However, these test problems did not include those with noise, constraints, or varying dimensions, which should be further explored.

7.2.5 Influence of legacy operators

The focus of my research involved removing some of the traditional evolutionary algorithm operators for my experiments. That is, to better focus on selection pressure effects derived from the machine learner, I used deterministic parent selection and non-overlapping generations thus minimizing two sources of selection pressure. I also did not include any form of crossover; and the sole source of novel genetic material was from the ML and not from a legacy mutation operator, such as Evolutionary Strategy style Gaussian mutation. Given that the ML and the EA components form a circular feedback system — i.e., the EA provides training material for the ML, which in turn generates offspring, when in turn provides new training material — it follows that adding these components will influence this feedback mechanism. It may very well be that legacy selection pressure could also change the aforementioned partially ordered set of design decisions regarding selection pressure. Presumably practitioners would desire better understanding of how these components affect

this mechanism.

7.2.6 Further exploration of training set configurations

Again, in this work I examined how training sets set up by fitness rank and by percentage influence affects how the system converges, and within those I looked at three different configurations. The results show that training set design decisions can have a critical effect on how a population converges.

Of note was that the by-fitness-percentage approach had skewed population ratios, a ratio that was problem dependent. For most problems using this approach the number of "best" training instances was much higher than that of the "worst"; what might the effect be should the percentages be altered such that the ratios are more in parity? Would this speed up or slow down convergence velocities?

With machine learner training sets sometimes more is better. However, it may very well be the case that the computational burden for fitness evaluation is such that the training sets have to be small; e.g., if an involved simulation must be run for several minutes to determine an individual's fitness, then that means not only smaller population sizes, but a limited birth budget, too, which in turn means smaller training sets. This can lead to poorer quality rules and consequently offspring that are not as viable.

However, it has been suggested that the the "best" and "worst" training sets be created from combined parent and offspring populations [Luke, 2008]. That is, for the initial population create as many individuals as the number of desired parents and offspring; e.g., for a (50+50) scenario randomly create 100 individuals instead of 50 and pick the "best" and "worst" from that 100; and then for every subsequent generation similarly learn from combining and sorting the parents and offspring into a single collection and choosing the "best" and "worst".

Also, as noted earlier in §7.2.1, individuals could be separately saved from older generations that would be otherwise lost to survival selection and also used to increase training set size.

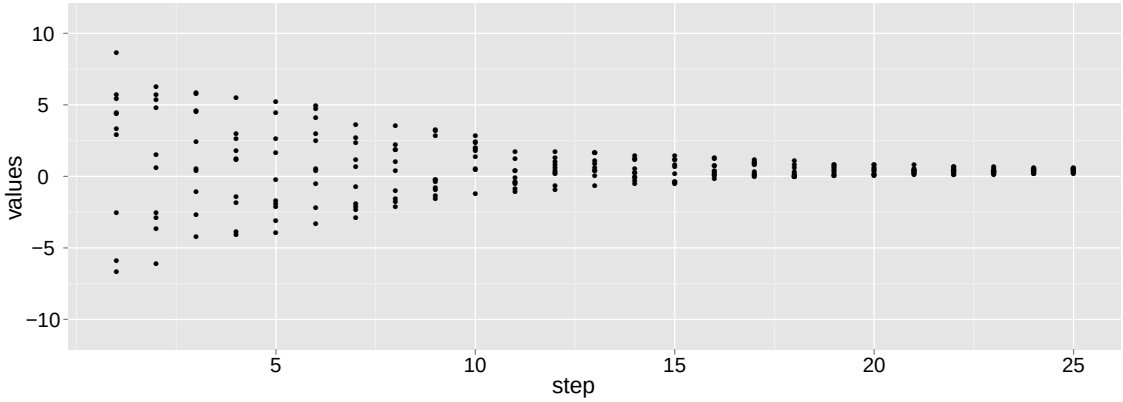


Figure 7.1: Results of starting out with ten random points in U(-10,10), then in each subsequent step sampling in the interval dictated by ranges from the minimum and maximum values from the previous step's interval.

7.2.7 Polynomial mutation

Some practitioners may wish to ensure that samples are strictly within rule boundaries, but may also wish to have some fine tuned control over the bias within intervals. Polynomial mutation enforces sampling strictly within given intervals, and also allows for adjusting the distribution's skew [Deb and Agrawal, 1999]. At this time no one has used polynomial mutation in this way for these types of EA/ML hybrids.

7.2.8 Influence of sample bias

It may be the case that iteratively sampling with a uniform distribution within an interval will gradually converge over time on its own accord and could partly explain the effect we're seeing. Fig. 7.1 depicts this effect. The experiments starts with a set of ten points sampled from U(-10,10). For the next step another ten points are similarly sampled from a uniform distribution. However, the interval is set to the minimum and maximum values from the first step. This process repeats where each subsequent step is comprised of ten points sampled from a uniform distribution with the new bounds set from the range of values from the previous step. Or, more formally:

$$x_{0,0}..x_{0,n-1} \sim U(-10, 10) \tag{7.1}$$

$$x_{i,0}..x_{i,n-1} \sim U(\min(x_{i-1,0}..x_{i-1,n-1}), \max(x_{i-1,0}..x_{i-1,n-1})) \tag{7.2}$$

Where the x's are discrete samples drawn from the current step's uniform distribution, and the current step's distribution's bounds dictated by the range of values in the previous step.

The notion here is that the intervals monotonically shrink with each step. That is, since it is almost impossible for a point to be generated on an interval boundary, and even less so for both boundaries, then the minimum and maximum points for a given sample will be within the current sample boundaries. Which means that the next interval will almost always be smaller. Since the procedure of clamping open bounds to the most extreme gene values has similar characteristics, then this process may be a contributing factor to the convergence rates using this approach.

7.2.9 Values for uncited genes

My research has shown that generally rules will refer to a subset of the genes within a genome. So, what to do with genes that are not referred to by rules when creating offspring?

There are differing strategies for addressing those genes that are not cited by any rules. Some LEM implementations would employ a global crossover operator such that genes from the fitter population would be arbitrarily assigned to genes not cited by a rule [Cervone et al., 2002]. Other implementations would fall back on a legacy EA mutation operator [Coletti, 2002]. In this work, uncited genes were merely cloned from the parent. In any case, different approaches for dealing with uncited rules merits further scrutiny.

We can draw some important lessons from the step function results regarding uncited genes. That is, in the step function the global optima was outside the implicit bounds dictated by the initial population. Though using a Gaussian distribution to sample rule intervals was successful in allowing exploration beyond those bounds, it was only partly so. Alternatively one could apply a mutation operator to the uncited genes as a means of nudging the population towards the global optima. Though a common practice with legacy LEM implementations, using a crossover operator would not succeed in moving the

population beyond this implicit bounds since we know that crossover operators also operate within an implicit bounds of the current population. That is, their novelty will always be within the bounding box of the current population, and not beyond it. However, this definitively needs to be put to the test within the LEM framework to firmly establish that this is the case.

7.3 Summary

In this chapter I related the contributions I have made in this research. These contributions include a simplified version of Learnable Evolution Model (LEM) known as Simplified LEM (SLEM) as well as two EA related data visualization techniques. I analyzed three approaches to resolve unbounded rule intervals. I also observed that the initial population defines an implicit bounds for SLEM, and one in which a Gaussian rule interval sampling approach is only partly successful in breaching. I also explored the impact that different rule intervals sampling strategies and training set configurations had on the selection pressure caused by using rules learned from a ML to create offspring in an EA. Additionally, I gave evidence to support that training sets that ignore parent population extrema can serve as a form of insurance policy to escape local optima at the end of runs. Furthermore, I determined that SLEM populations were unable to escape a very large basin attraction for local optima. And, finally, I observed that the rectilinear-oriented perspective that SLEM uses to model the fitness landscape is sometimes harmful and sometimes beneficial when problems with rectilinearly arranged optima are randomly rotated.

The future work I related in this chapter is not an exhaustive list, but nonetheless establish starting points for further research. First, I discuss how keeping rules or individuals longer than a generation might decrease selection pressure. Then I suggested looking into the use of negative examples to guide population trajectories. I also proposed that the influence of ML inductive bias on the performance be studied. I also shared that it would be useful to exercise SLEM on a larger variety of test problems. Additionally, I noted that it was important to better understand the role in legacy EA operators in LEM-like

system behavior. I also speculated that polynomial mutation, which has traditionally been used in multiobjective optimization EAs, might prove useful in LEM implementations. Furthermore, I described how sample bias may also be an independent contributor separate from the ML to the selection pressure that we have observed in these systems. And, finally, I noted that how approaches to address genes not covered by rules when creating offspring bears further scrutiny.

Appendix A: Rule Interval Sampling Plots

For completeness I have placed plots for the population trajectories, selection intensities, and number of learned rules for all the test functions in this section to make it easier to find relevant plots and to do side-by-side comparisons. However, some plots can be found elsewhere in this chapter; and, where that happens, I ensure that the section and page number is given to make a given plot easier to find.

A.1 Spheroid

Note that the figure for the population trajectories and selection intensities for the uniform runs is found in §4.2.2 on page 54.

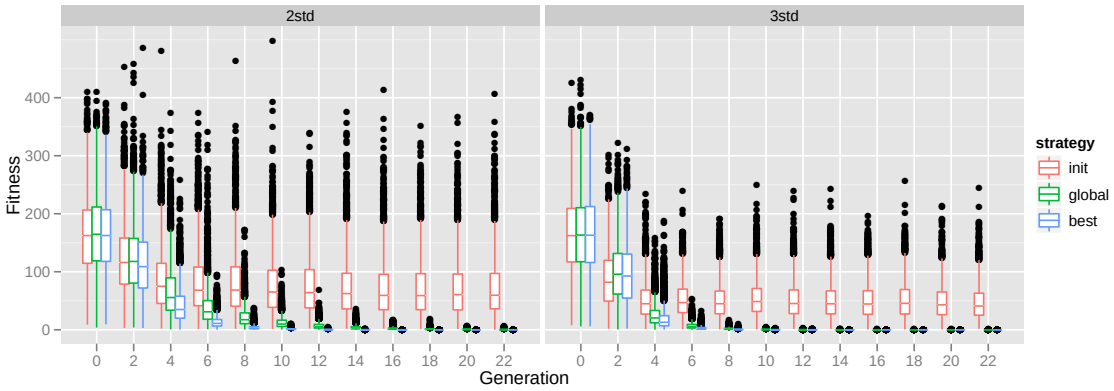


Figure A.1: Population trajectories for f_{sphe} for three different open rule interval clamping strategies using Gaussian-based rule interval sampling policies for two and three standard deviations.

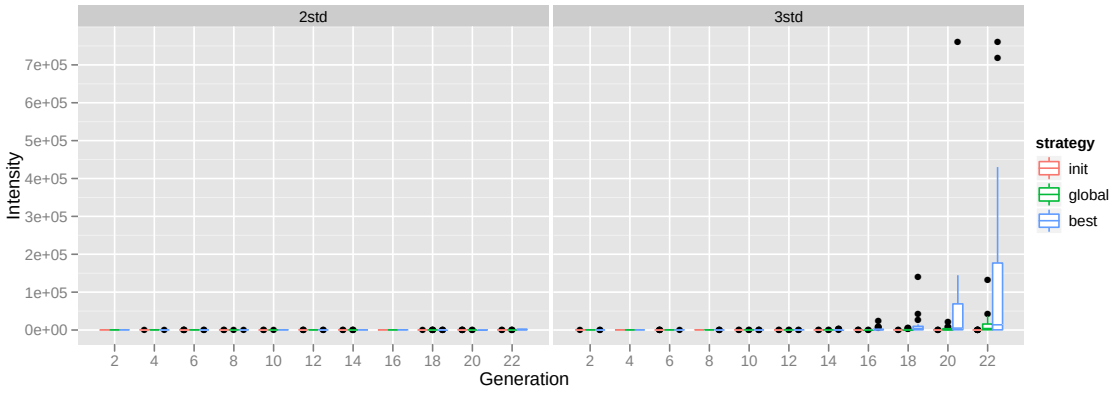


Figure A.2: Selection intensities for f_{sphe} for three different open rule interval clamping strategies using Gaussian-based rule interval sampling policies for two and three standard deviations.

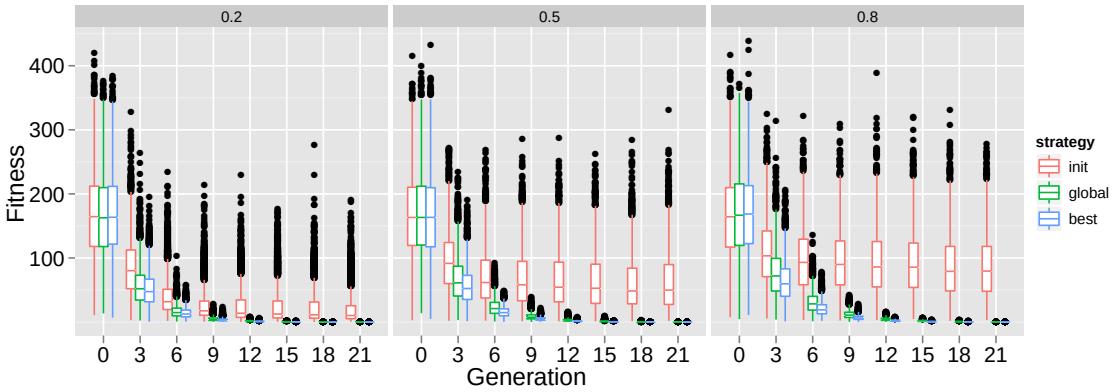


Figure A.3: Population trajectories for f_{sphe} for three different open rule interval clamping strategies using a histogram-based rule interval sampling approach using three budgets for the influence of a uniform distribution.

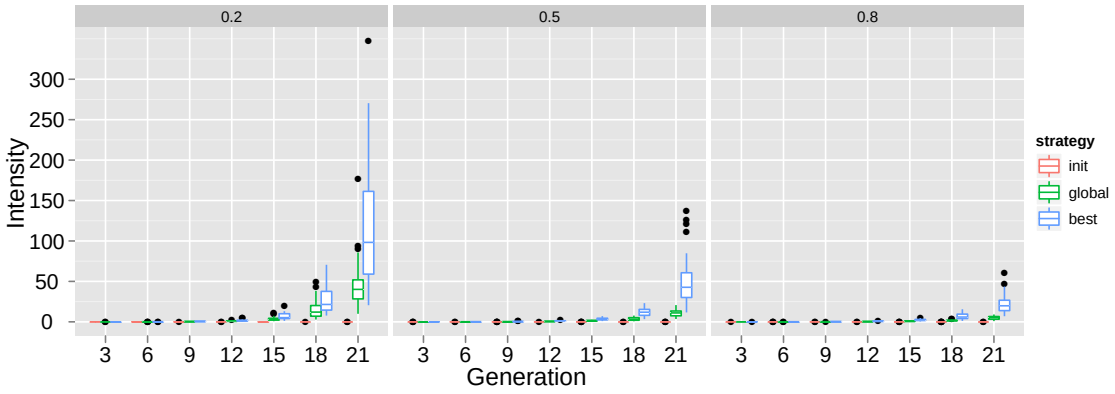


Figure A.4: Selection intensities for f_{sphe} for three different open rule interval clamping strategies using a histogram-based rule interval sampling approach.

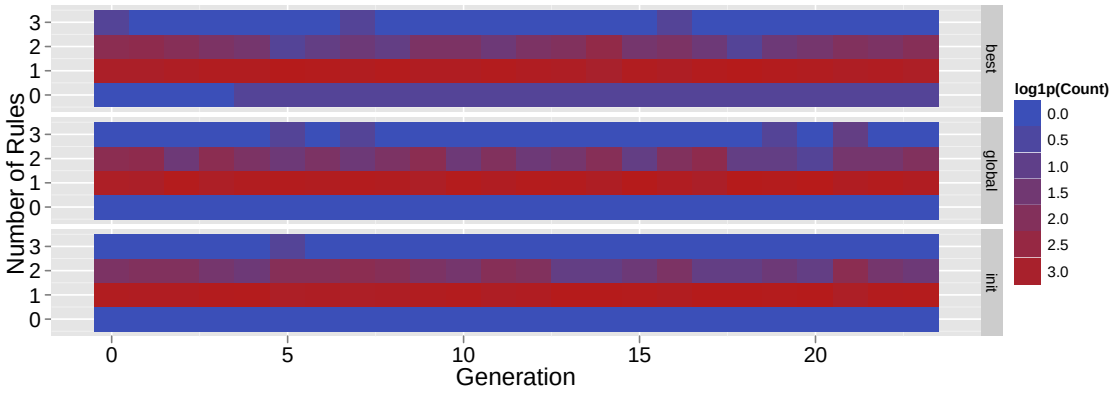


Figure A.5: Heat map of aggregate count of number of rules by generation for f_{sphe} for three different open rule interval clamping strategies using a uniform rule interval sampling policy.

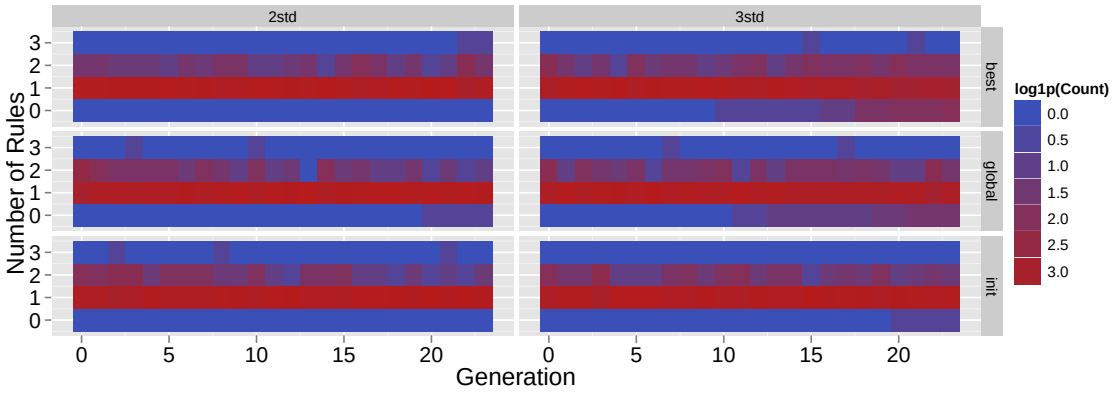


Figure A.6: Heat map of aggregate count of number of rules by generation for f_{sphe} for three different open rule interval clamping strategies using a Gaussian distribution to sample rule intervals within two and three standard deviations.

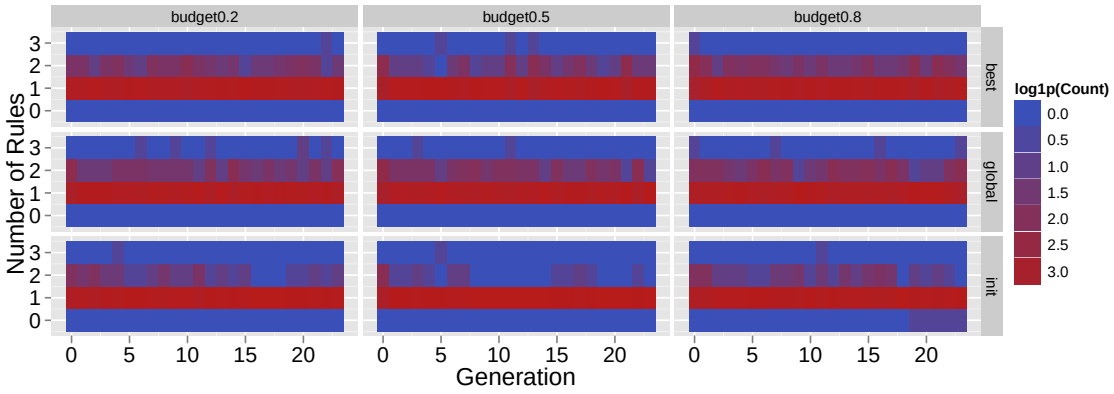


Figure A.7: Heat map of aggregate count of number of rules by generation for f_{sphe} for three different open rule interval clamping strategies using an histogram-based approach to sample rule intervals for the influence of a uniform distribution.

A.2 Step

Not that the plot for population trajectories for f_{step} where a uniform and Gaussian distribution was used to sample rule intervals is found in §4.3.2 on page 75.

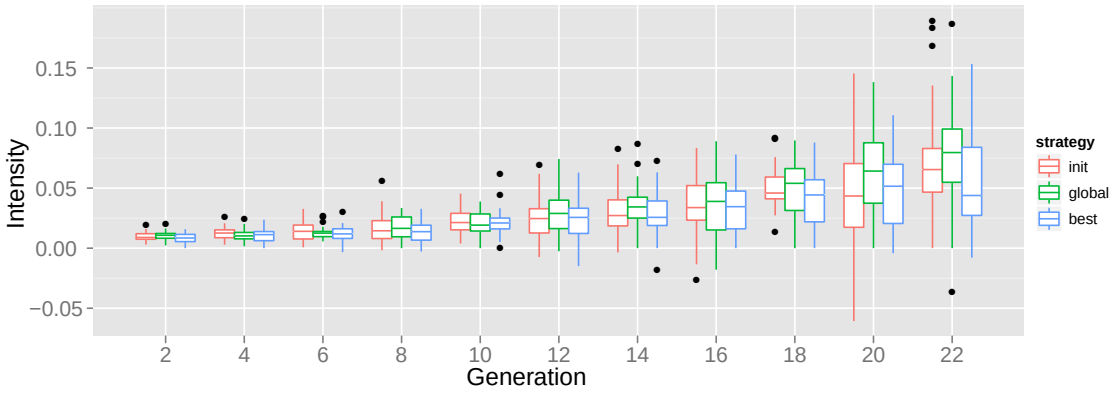


Figure A.8: Selection intensities for $f_{step}()$ for three different open rule interval clamping strategies using a uniform distribution to sample rule intervals.

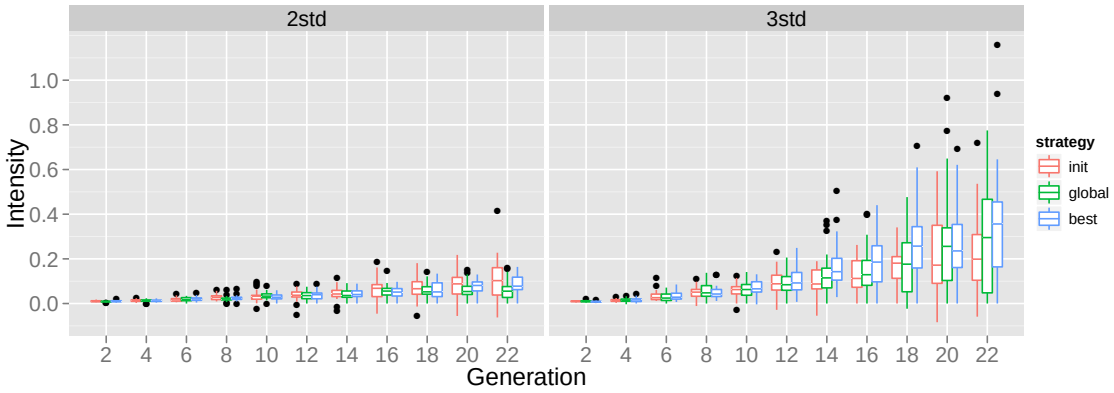


Figure A.9: Selection intensities for $f_{step}()$ for Gaussian distributions of two and three standard deviations for sampling rule intervals. As with Fig. 4.27, the plot is further broken down by closed rule interval strategies of 'init', 'global', and 'best'.

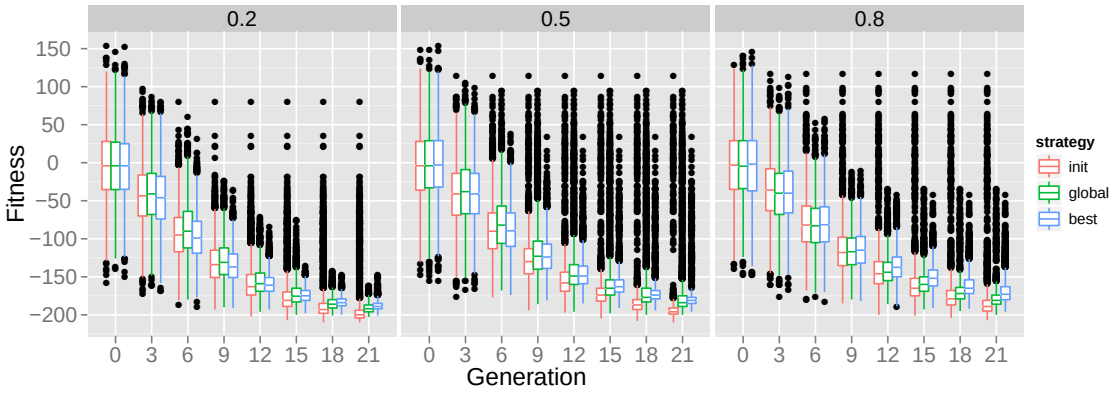


Figure A.10: Population trajectories for f_{step} using a histogram-based rule interval sampling approach for histogram uniform budgets of 0.2, 0.5, and 0.8. The figure is further broken down by open rule interval closing strategies 'init', 'global', and 'best.

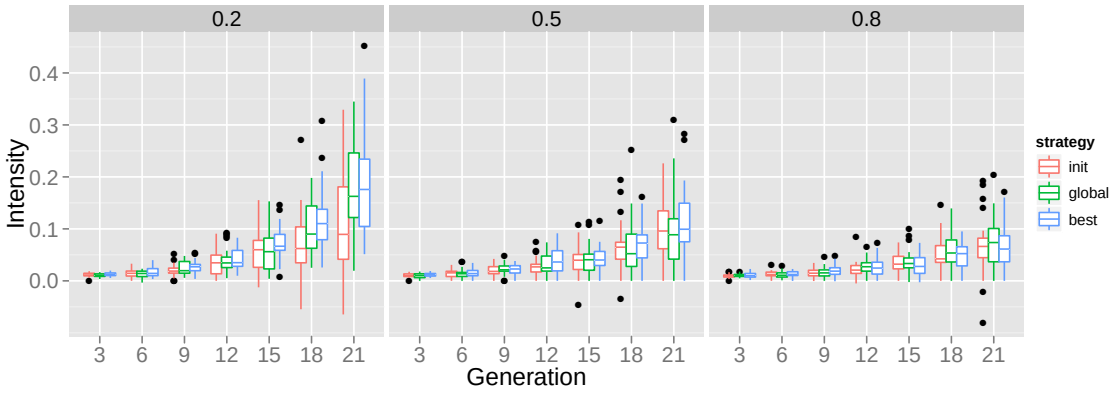


Figure A.11: Selection intensities for f_{step} using an histogram-based rule interval sampling approach for uniform budgets of 0.2, 0.5, and 0.8. The figure is further broken down by 'init', 'global', and 'best' open rule interval closing strategies.

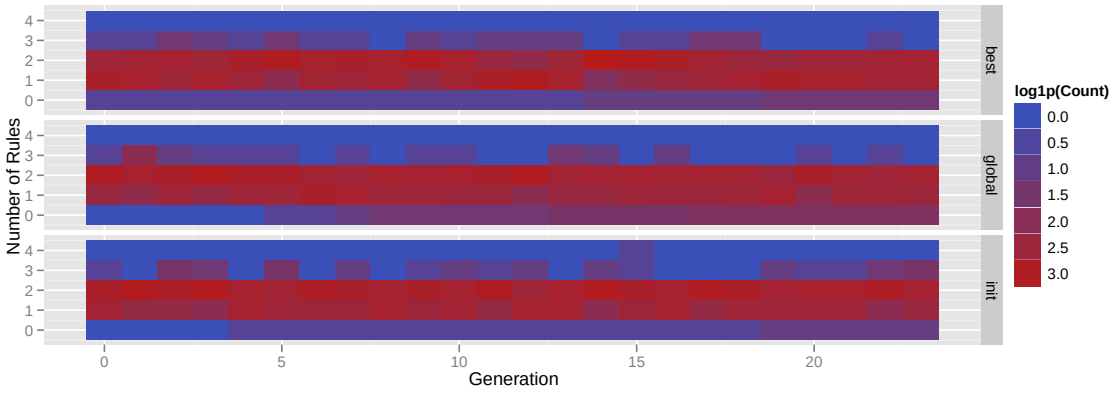


Figure A.12: Heat maps of the aggregate number of rules for f_{step} using a uniform distribution.

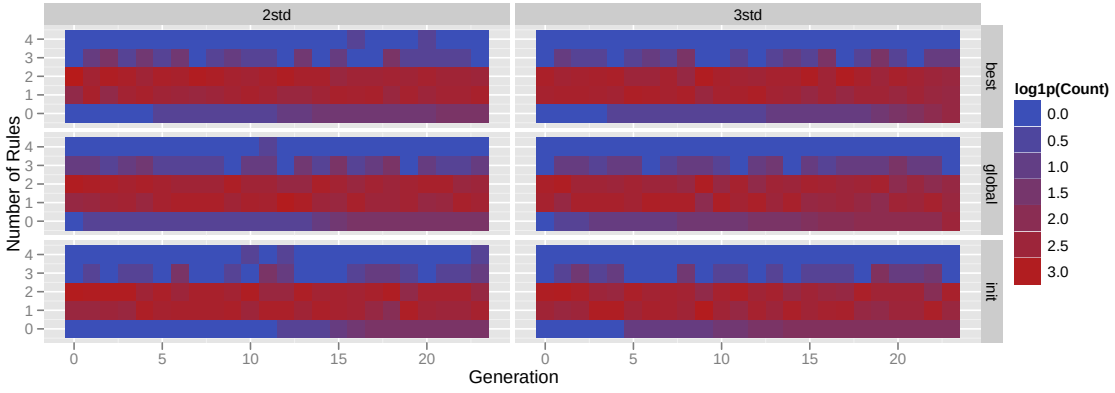


Figure A.13: Heat maps for the aggregate number of rules for f_{step} using a Gaussian distribution using two and three standard deviations within rule intervals.

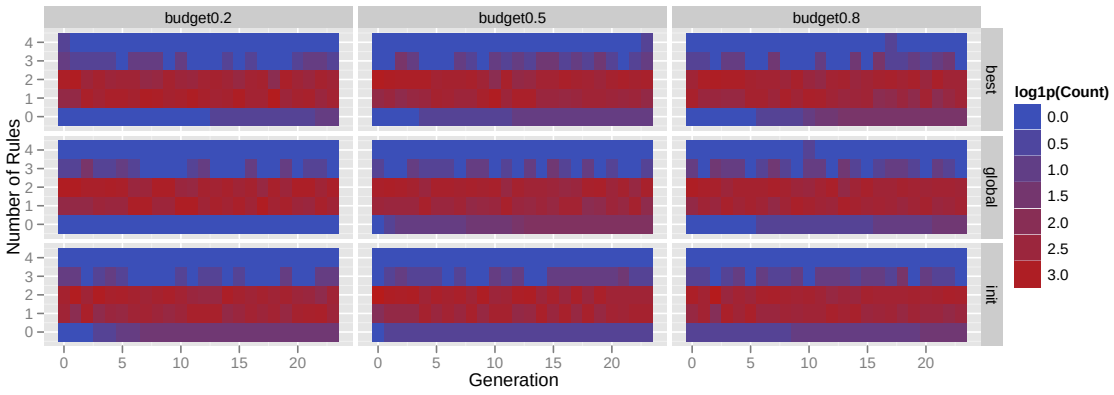


Figure A.14: Heat maps for the aggregate number of rules for f_{step} using a histogram-based distribution with three budget levels for the influence of a uniform distribution.

A.3 Rastrigin

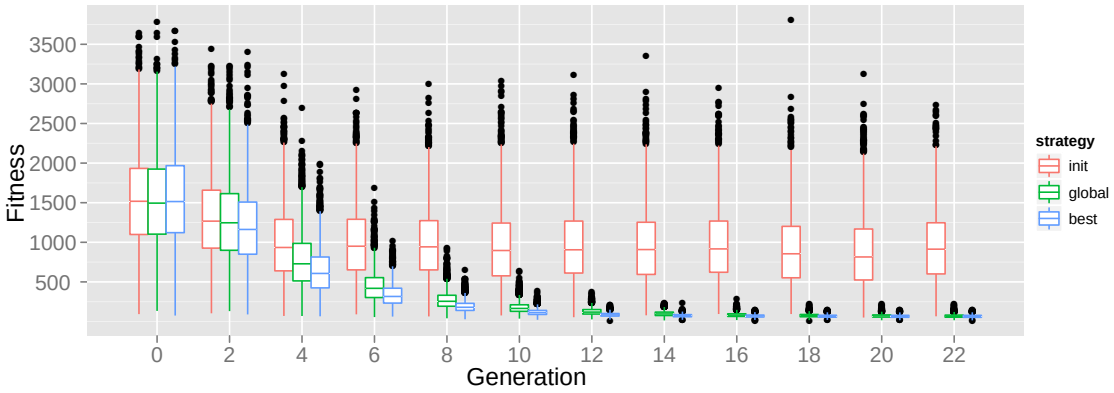


Figure A.15: Population trajectories $f_{rast}()$ for the 'init', 'global', and 'best' open rule interval clamping strategies using a sampling of rule intervals from a uniform distribution.

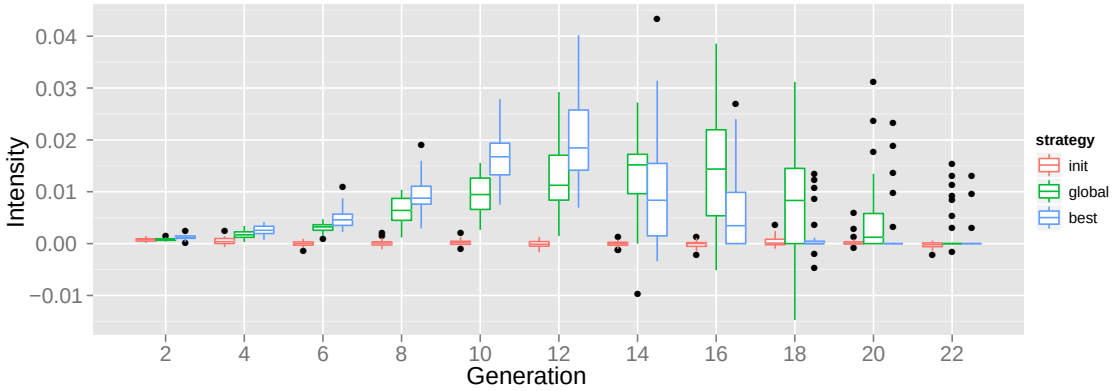


Figure A.16: Selection intensities for $f_{rast}()$ by generation between 'best' and 'global' open rule interval clamping strategies using a sampling of rule intervals fro a uniform distribution.

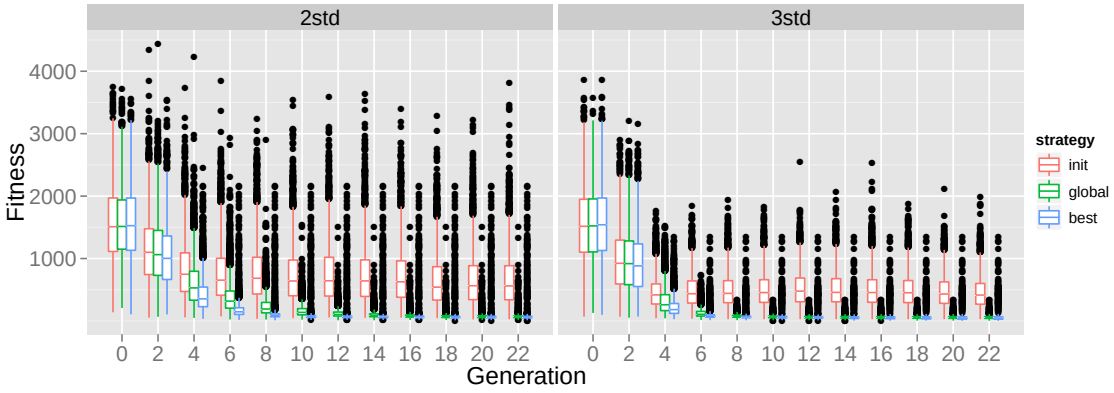


Figure A.17: Population trajectories for $f_{rast}()$ using a Gaussian sampling of rule intervals. This plots are broken down by the two and three standard deviation runs, and further subdivided by open rule interval closing strategies.

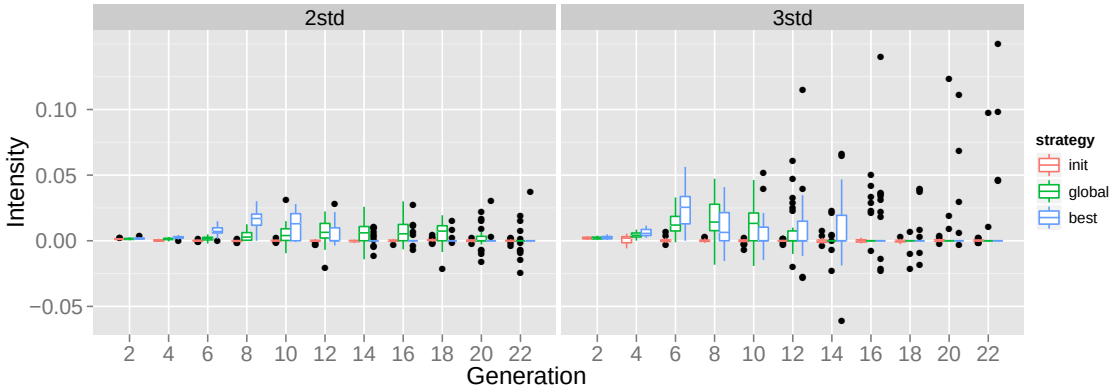


Figure A.18: Selection intensities for $f_{rast}()$ using a Gaussian sampling of rule intervals. This plots are broken down by the two and three standard deviation runs, and further subdivided by the open rule interval closing strategies.

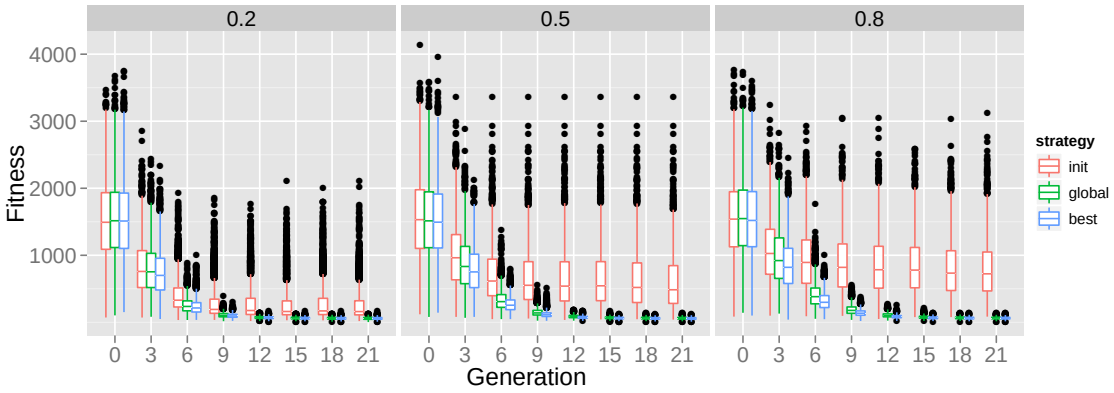


Figure A.19: Population trajectories for $f_{rast}()$ using a histogram-based sampling of rule intervals. This plots are broken down by the budgets for the influence of a uniform distribution, and further subdivided by the open rule interval closing strategies.

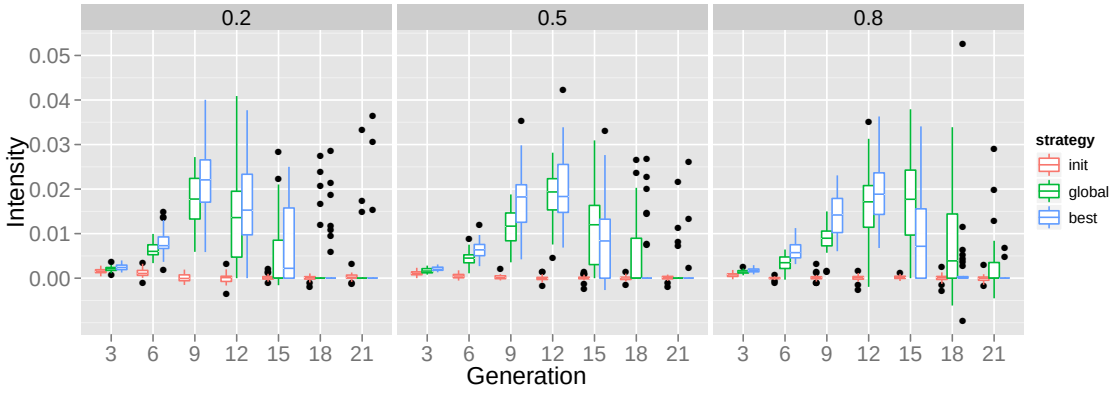


Figure A.20: Selection intensities for $f_{rast}()$ using a histogram-based sampling of rule intervals. This plots are broken down by budgets for the influence of a uniform distribution, and further subdivided by the open rule interval closing strategies.

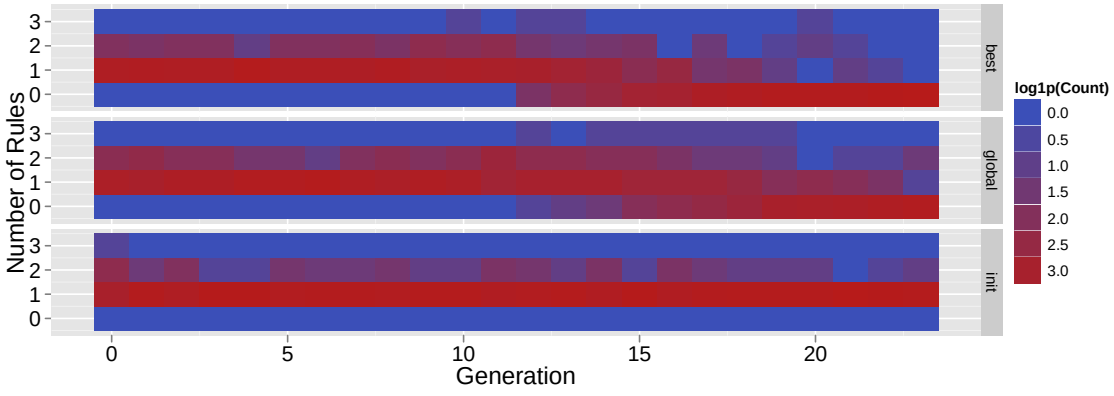


Figure A.21: Heat maps for the aggregate number of rules for f_{rast} using a uniform distribution.

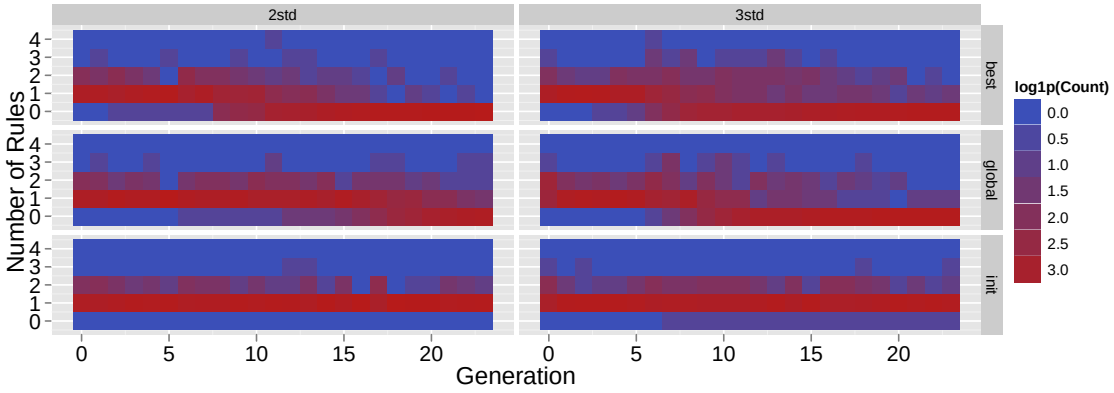


Figure A.22: Heat maps for the aggregate number of rules for f_{rast} using a Gaussian distribution.

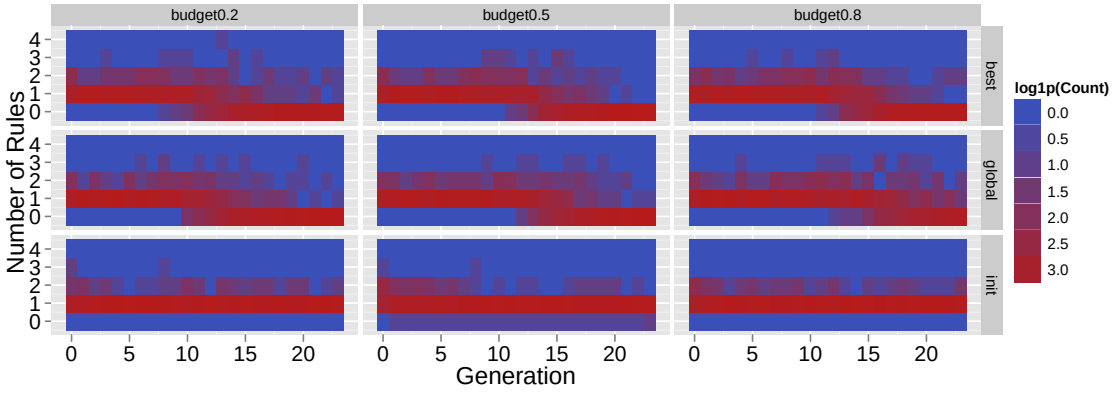


Figure A.23: Heat maps for the aggregate number of rules for f_{rast} using a histogram-based distribution broken down by budget dedicated to a uniform distribution.

A.4 Rotated Rastrigin

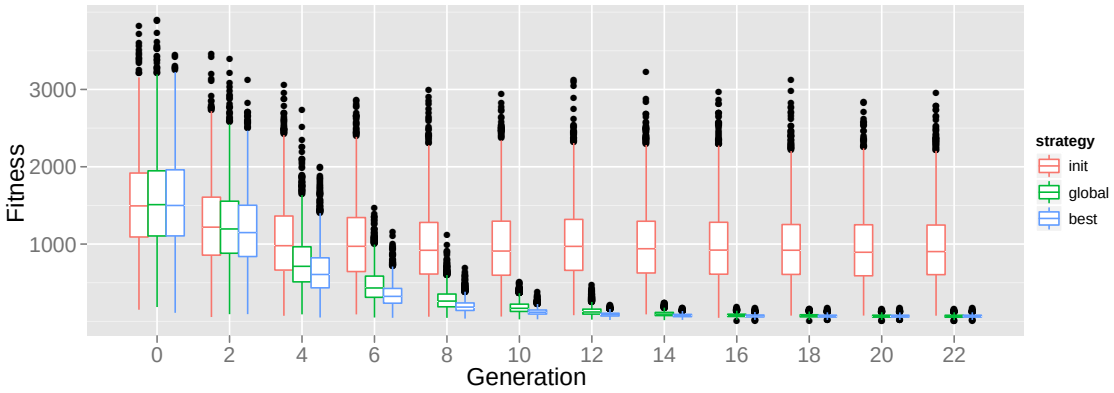


Figure A.24: Population trajectories for f_{rot_rast} using a uniform distribution.

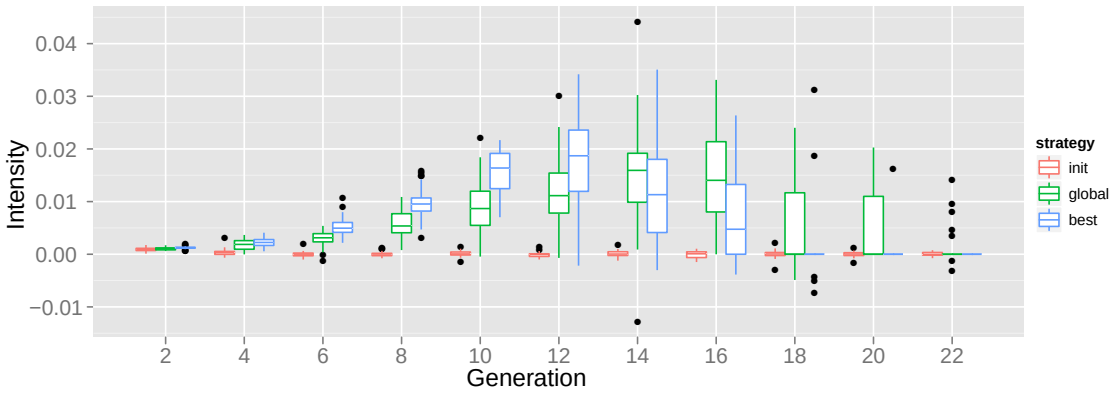


Figure A.25: Selection intensities for f_{rot_rast} using a uniform distribution.

The population trajectories for the f_{rot_rast} histogram-based population trajectories can be found in §4.2.3 on page 57.

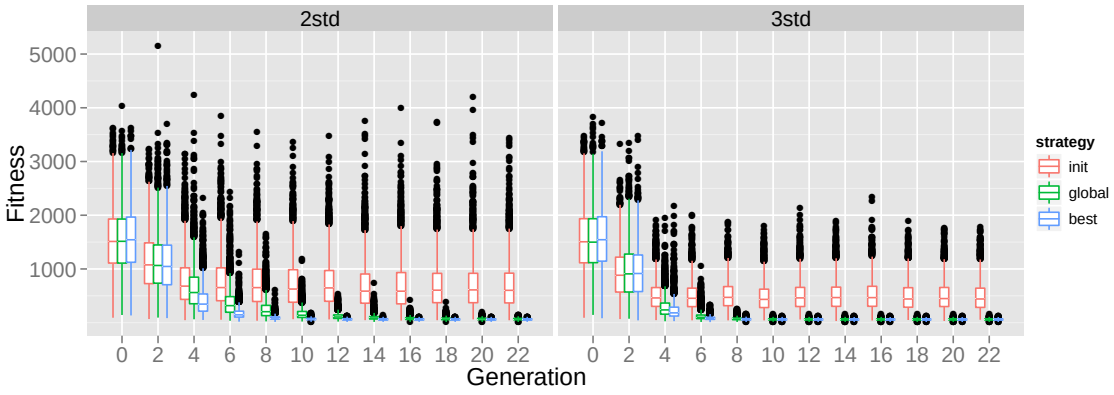


Figure A.26: Population trajectories for f_{rot_rast} using a Gaussian distribution of two and three standard deviations to sample rule intervals.

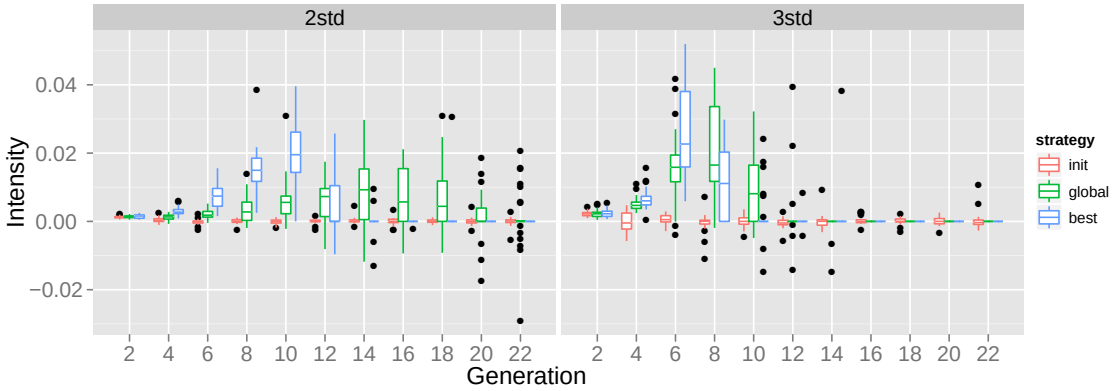


Figure A.27: Selection intensities for f_{rot_rast} using Gaussian distributions with two and three standard deviation to sample rule intervals.

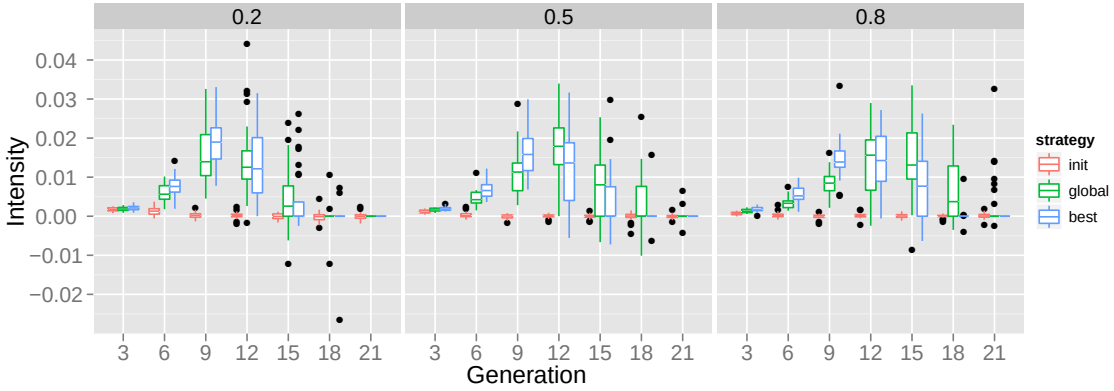


Figure A.28: Selection intensities for f_{rot_rast} using the histogram-based distribution to sample rule intervals further broken down by budget dedicated to a uniform distribution.

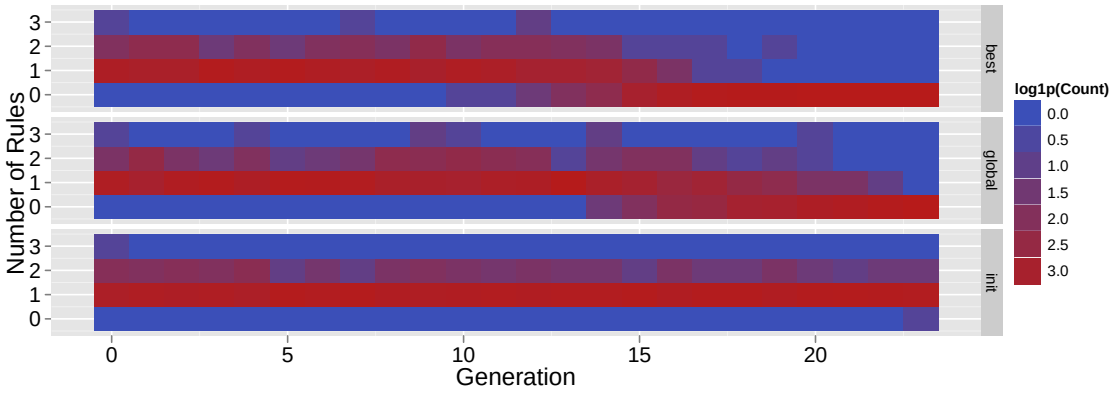


Figure A.29: Heat maps for the aggregate number of rules for f_{rot_rast} using a uniform distribution to sample rule intervals.

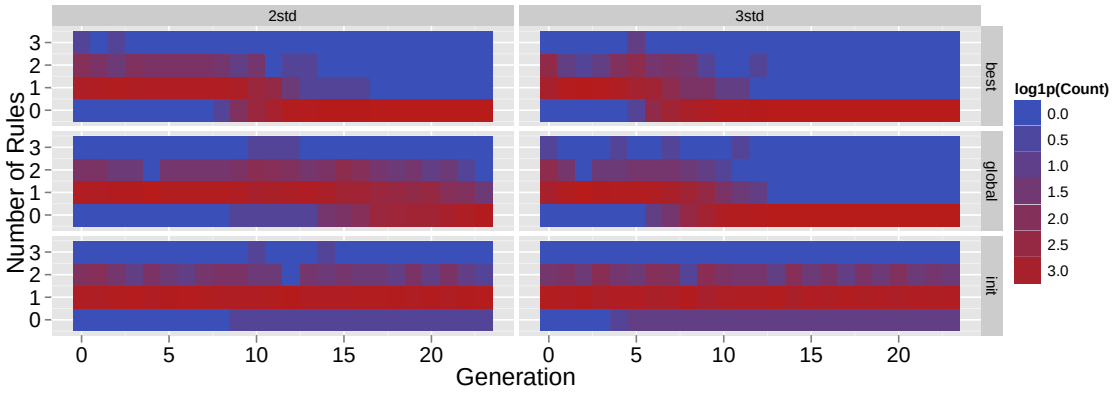


Figure A.30: Heat maps for the aggregate number of rules for f_{rot_rast} using Gaussian distributions of two and three standard deviations to sample rule intervals.

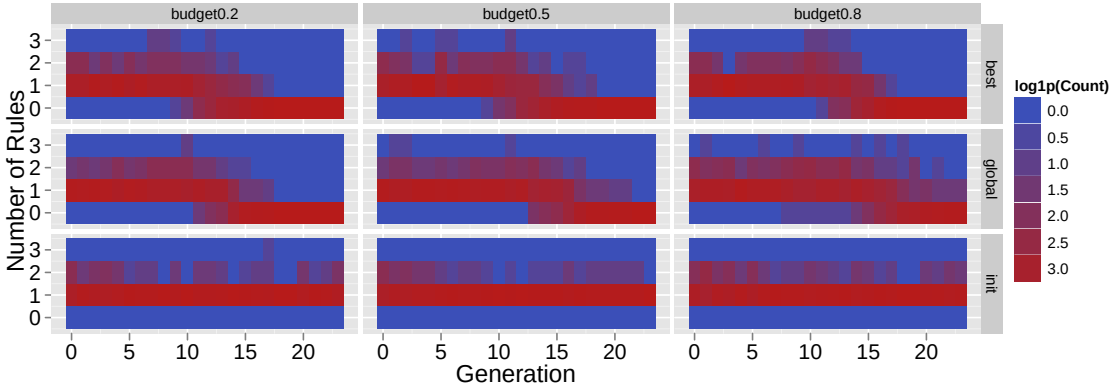


Figure A.31: Heat maps for the aggregate number of rules for f_{rot_rast} using a histogram-based distribution to sample rule intervals broken down by budget dedicated to a uniform distribution.

A.5 Griewangk

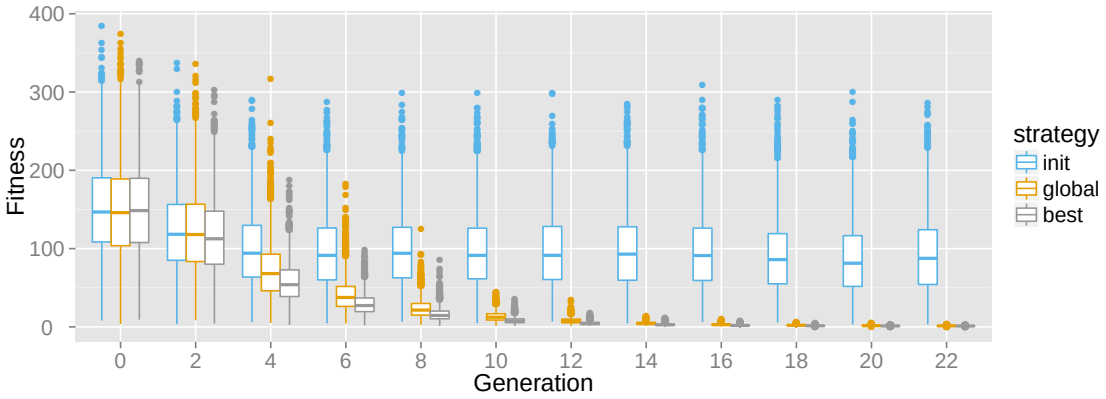


Figure A.32: Population trajectories for f_{grie} using a uniform distribution to sample rule intervals for the three open rule interval strategies *init*, *global*, and *best*. Once again the *init* strategy stops converging relatively early as compared to the othe two strategies; *global* and *best* have similar trajectories, though *best* converges more closely to the global optimum.

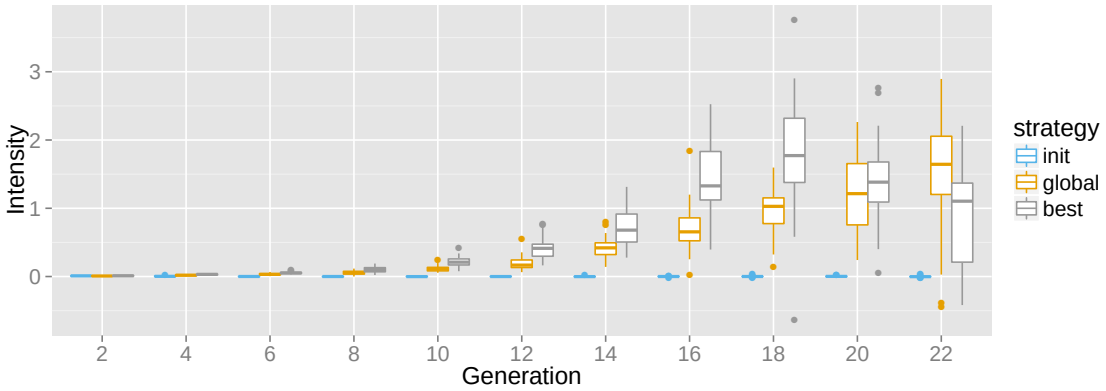


Figure A.33: Selection intensities for f_{grie} using a uniform distribution to sample rule intervals. The selection intensities for *init* open rule closing strategy remain flat throughout the runs. The *global* strategy selection intensities steadily increase as the runs progresss whereas the *best* strategies also increase, though peak at the 18th generation.

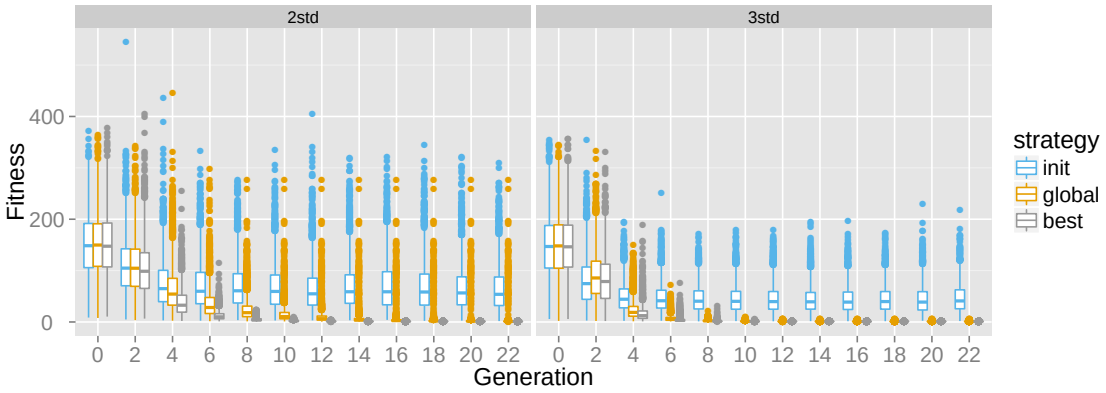


Figure A.34: Population trajectories for f_{grie} using Gaussian distributions of two and three standard deviations to sample rule intervals. We can observe that the *init* strategy continues to lag behind *global* and *best* strategies. Again, the *global* and *best* strategies have similar trajectories, with the *best* being the most aggressive.

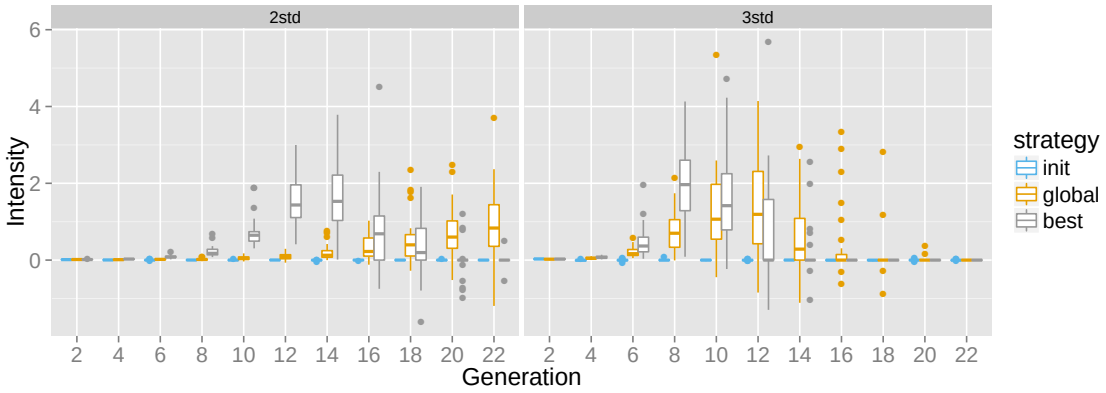


Figure A.35: Selection intensities for f_{grie} using Gaussian distributions of two and three standard deviations to sample rule intervals. Once more the *init* selection intensities are more-or-less flat relative to the *global* and *best* strategies. For the two standard deviation runs, the *global* strategy intensities continue to increase throughout the runs. The *best* strategy runs for both the two and three standard deviation runs steadily rise to peak midway through the runs; the *global* strategy exhibits similar trajectories for the three standard deviation runs.

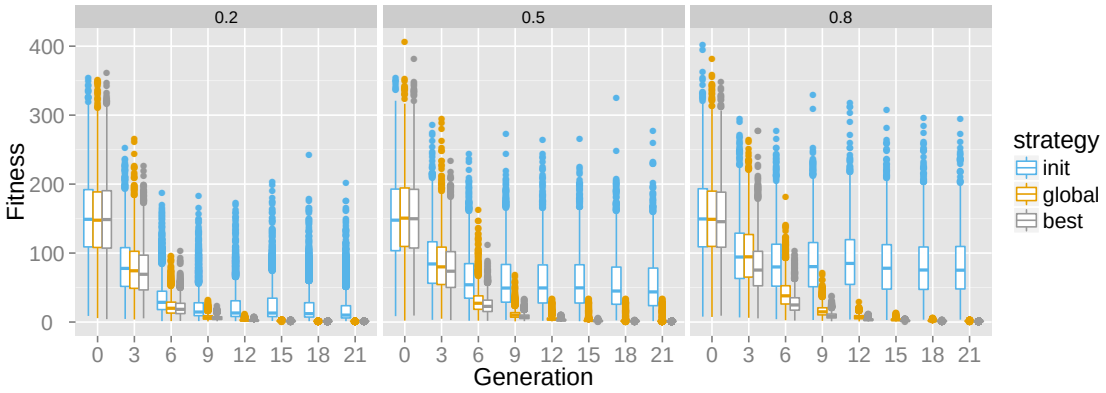


Figure A.36: Population trajectories for f_{grie} using the histogram-based distribution to sample rule intervals broken down by budget dedicated to a uniform distribution. This shows the *init* once again lagging behind *global* and *best* strategies. However, selection pressure of the *init* strategy is inversely proportional to the amount of uniform distribution mixed into the histogram-based distribution. The *global* and *best* have similar trajectories, though the *best* strategy exhibits the highest selection pressure.

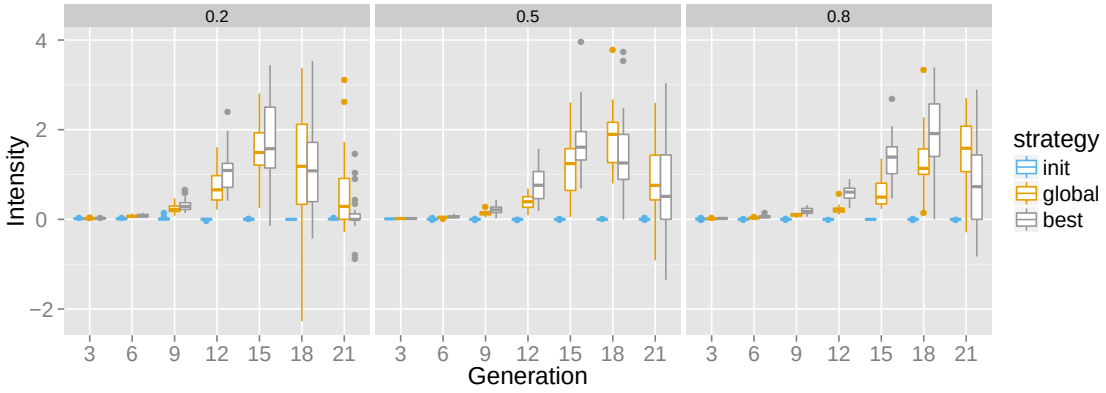


Figure A.37: Selection intensities for f_{grie} using the histogram-based distribution to sample rule intervals broken down by budget dedicated to a uniform distribution. The *init* strategy selection intensities remain flat throughout all the runs. The *global* strategy selection intensity gradually rises for the 0.8 uniform distribution mix, but exhibits a peak midway through the other runs; the *best* strategy exhibits such a peak for all its runs.

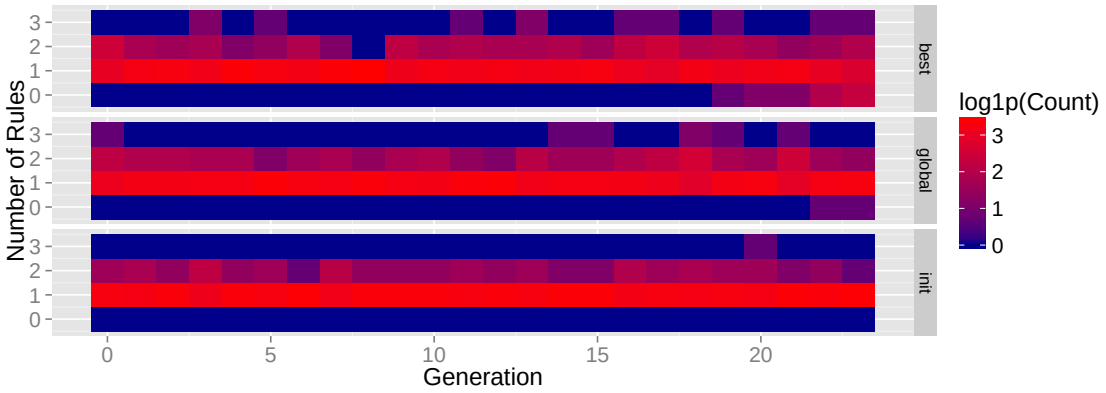


Figure A.38: Heat maps for the aggregate number of rules for f_{grie} using a uniform distribution to sample rule intervals. The frequency color scale has been log transformed to make lower counts easier to see.

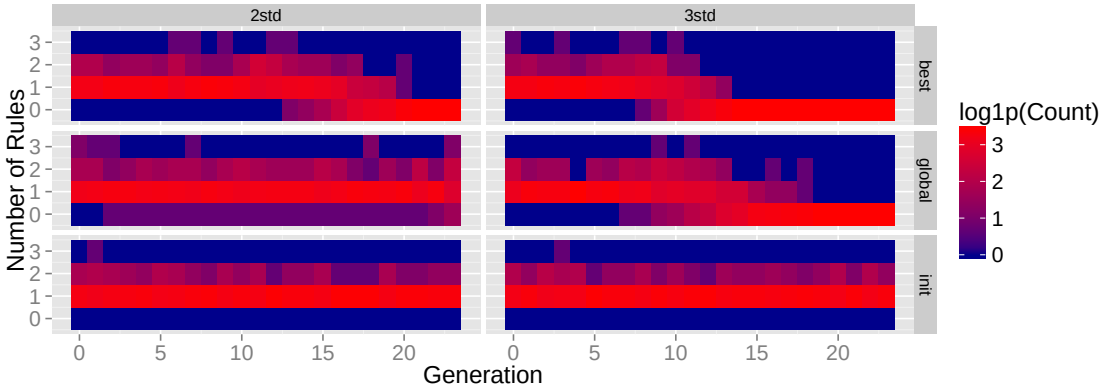


Figure A.39: Heat maps for the aggregate number of rules for f_{grie} using Gaussian distributions of two and three standard deviations to sample rule intervals. The frequency color scale has been log transformed to make lower counts easier to see.

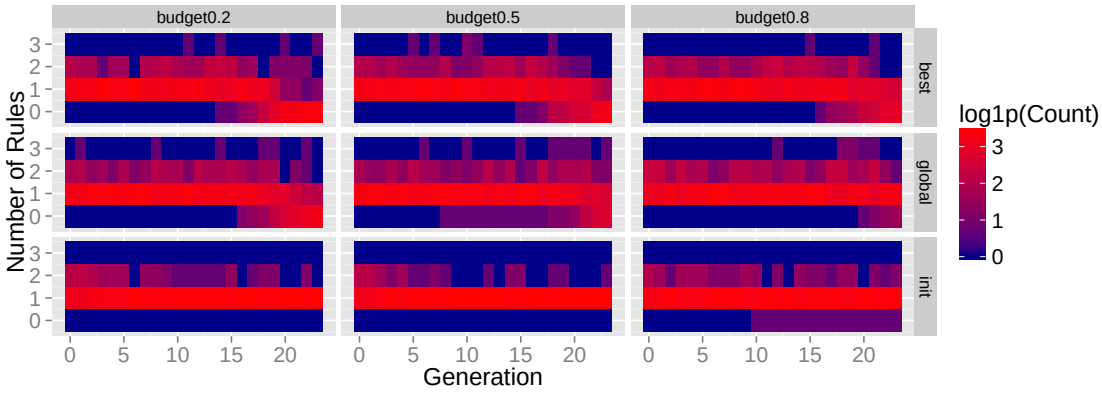


Figure A.40: Heat maps for the aggregate number of rules for f_{grie} using a histogram-based distribution further broken down by budget dedicated to a uniform distribution. The frequency color scale has been log transformed to make lower counts easier to see.

A.6 Rotated Griewangk

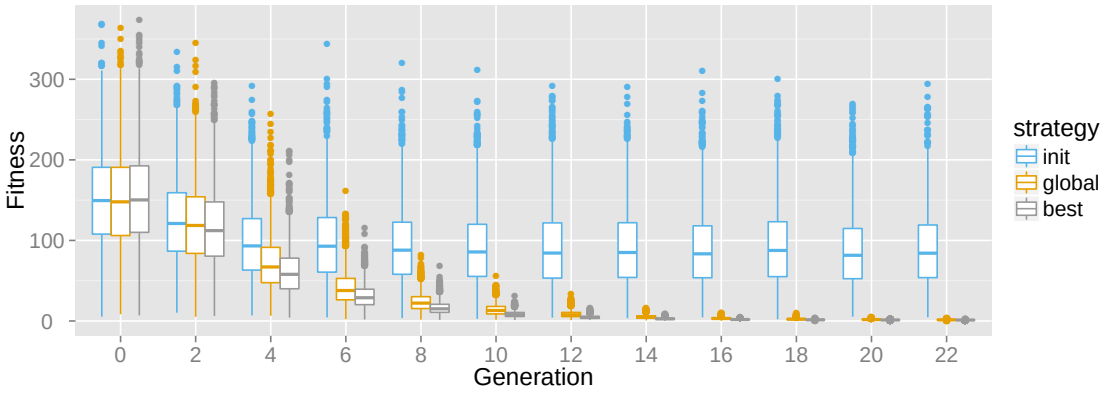


Figure A.41: Population trajectories for f_{rot_grie} using a uniform distribution to sample rule intervals.

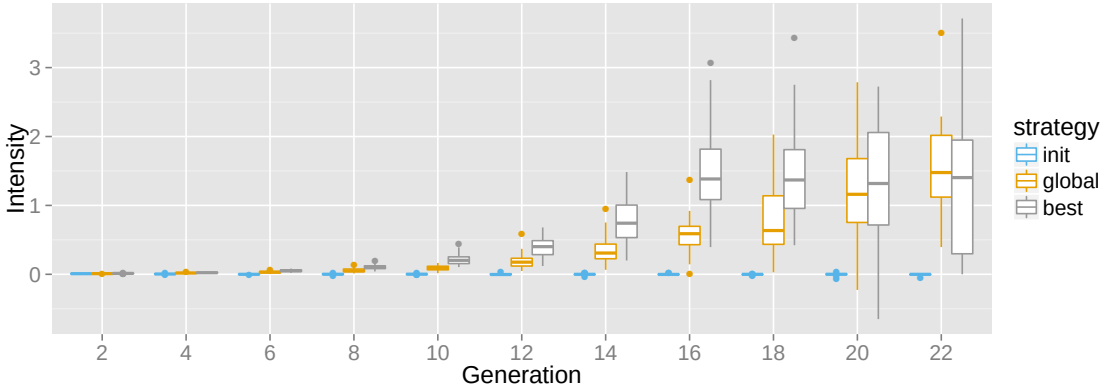


Figure A.42: Selection intensities for f_{rot_grie} using the uniform distribution to sample rule intervals.

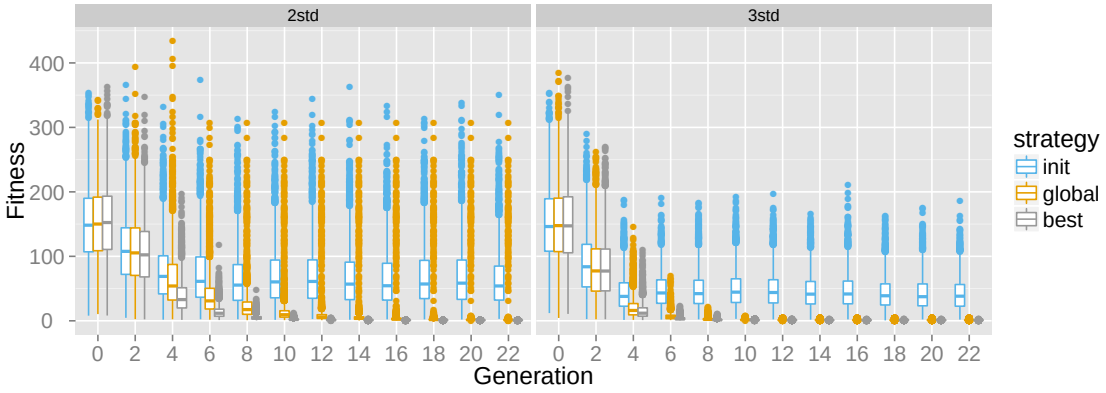


Figure A.43: Population trajectories for f_{rot_grie} using Gaussian distributions of two and three standard deviation to sample rule intervals.

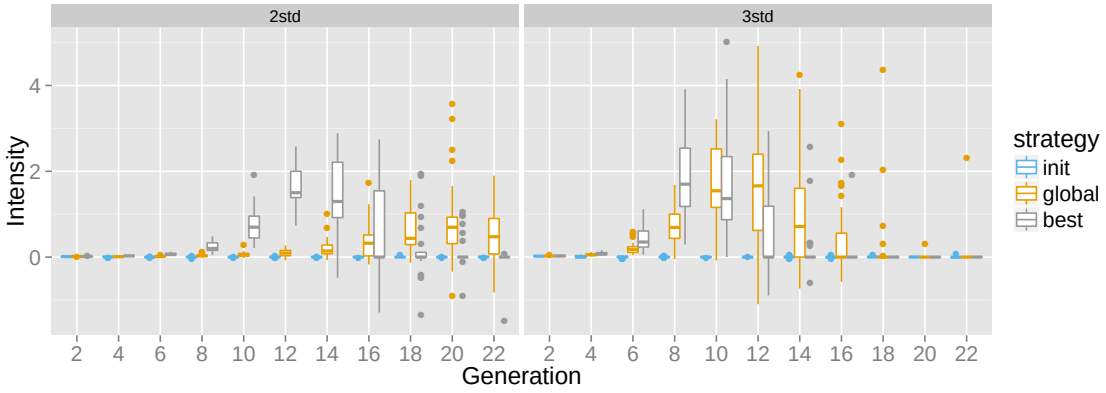


Figure A.44: Selection intensities for f_{rot_grie} using Gaussian distributions of two and three standard deviation to sample rule intervals.

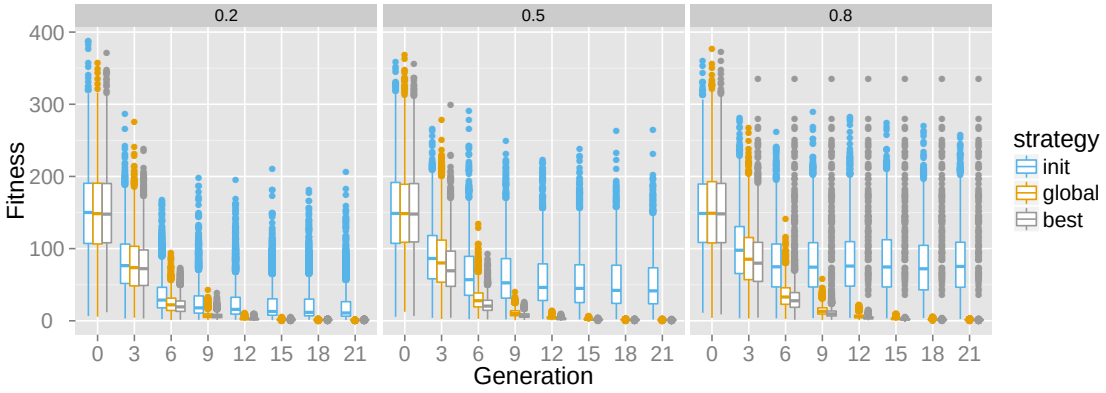


Figure A.45: Population trajectories for f_{rot_grie} using a histogram-based distribution to sample rule intervals further broken down by budget dedicated to a uniform distribution.

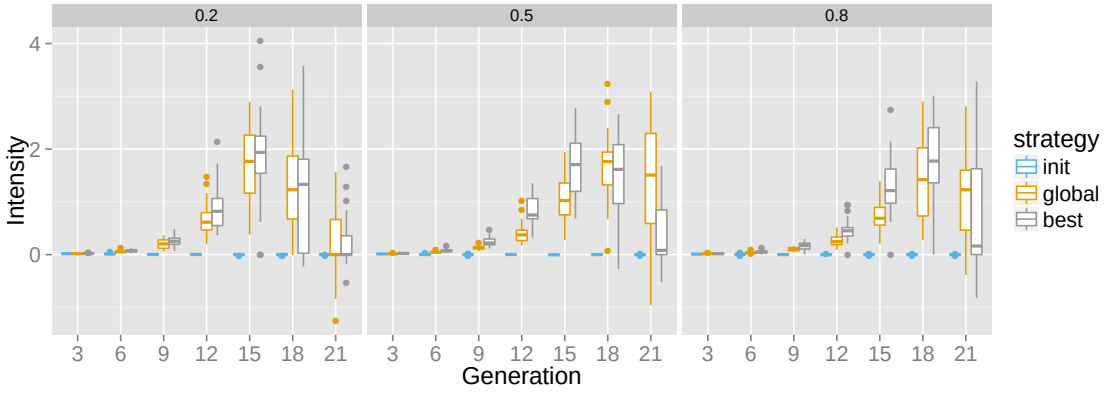


Figure A.46: Selection intensities for f_{rot_grie} using a histogram-based distribution to sample rule intervals further broken down by budget dedicated to a uniform distribution.

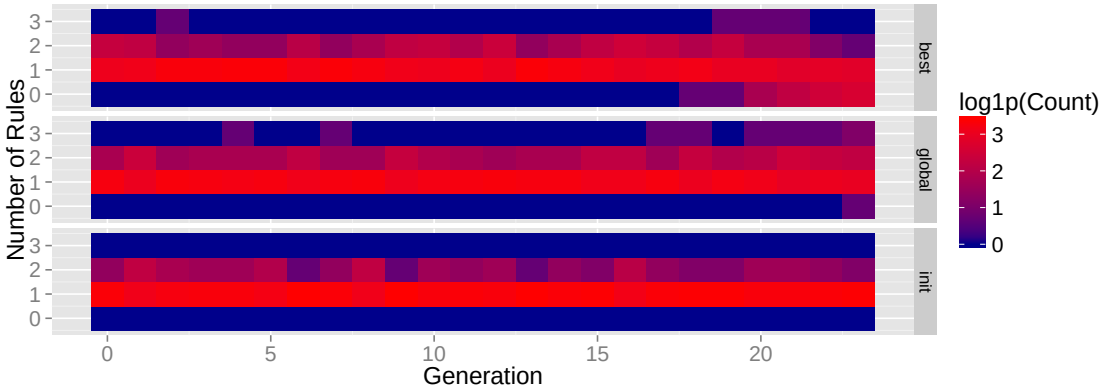


Figure A.47: Heat maps for the aggregate number of rules for f_{rot_grie} using a uniform distribution to sample rule intervals.

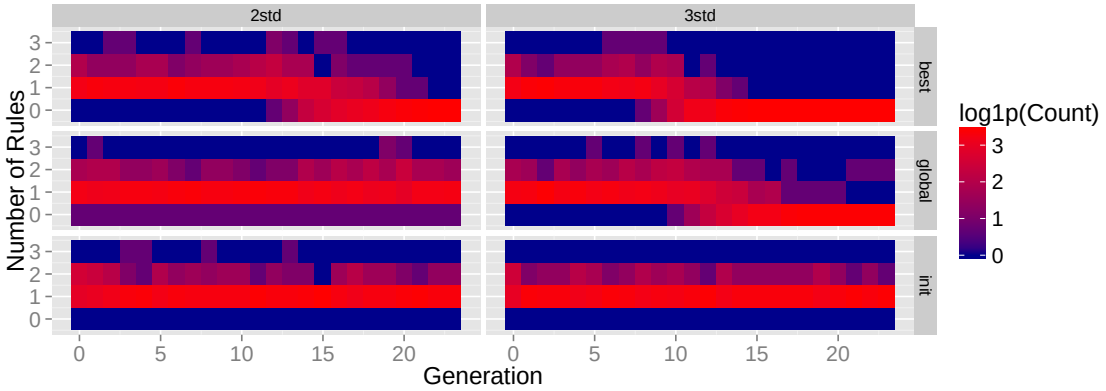


Figure A.48: Heat maps for the aggregate number of rules for f_{rot_grie} using a Gaussian distribution of two and three standard deviations to sample rule intervals.

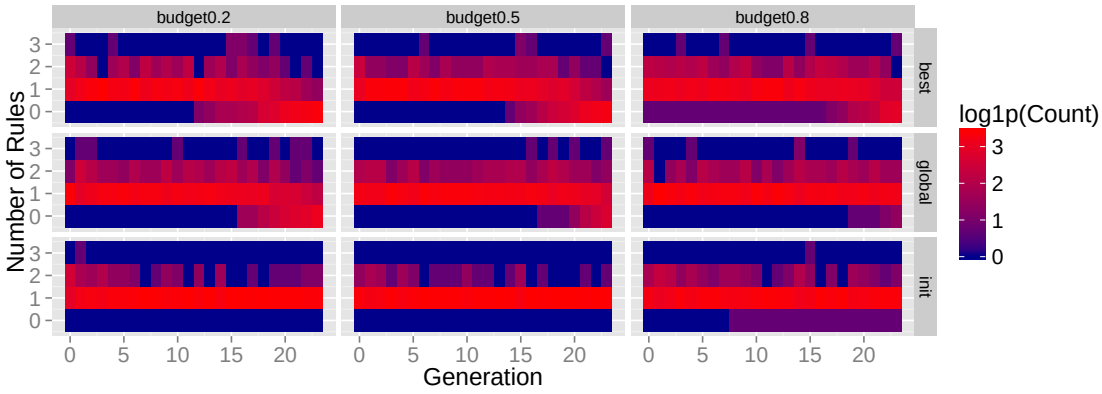


Figure A.49: Heat maps for the aggregate number of rules for f_{rot_grie} using a histogram-based distribution to sample rule intervals further broken down by budget dedicated to a uniform distribution.

A.7 Langerman

Note that the f_{lang} uniform rule interval sampling plot can be found in §4.2.3.

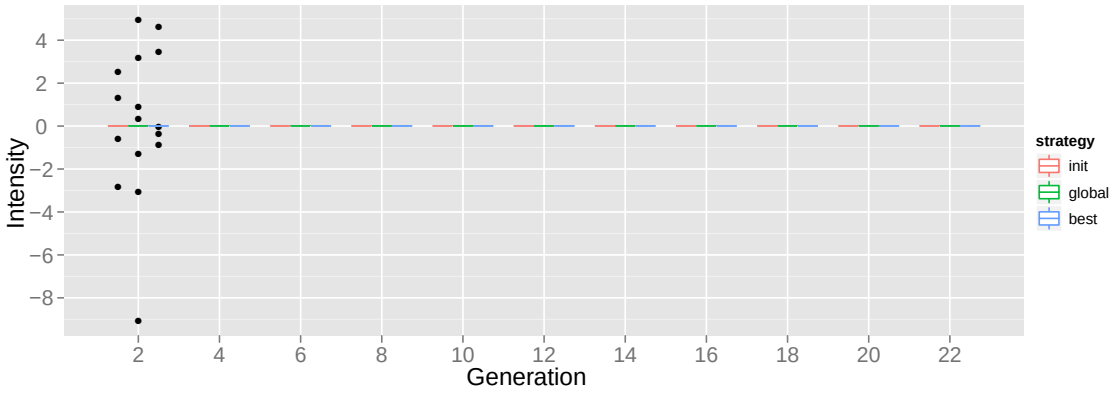


Figure A.50: Selection intensities for f_{lang} using the uniform distribution to sample rule intervals.

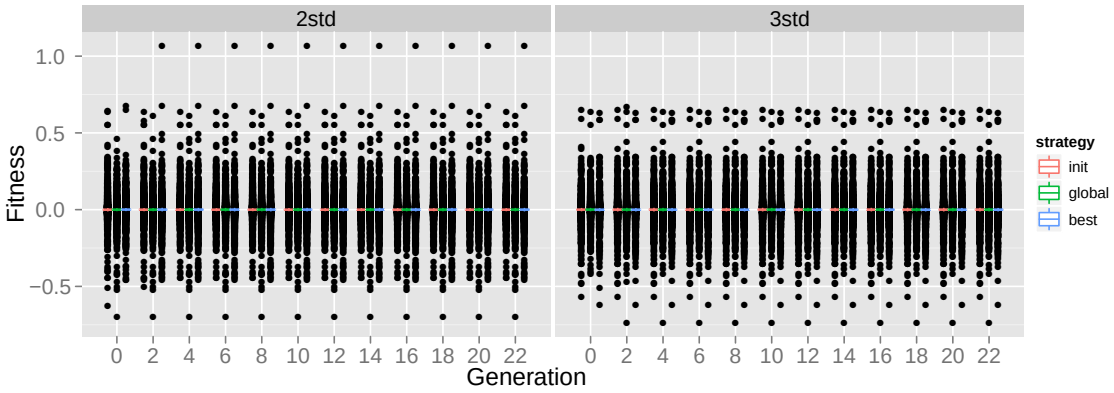


Figure A.51: Population trajectories for f_{lang} using Gaussian distributions of two and three standard deviations to sample rule intervals.

The heat map figure for the number of aggregate rules learned for f_{lang} is found in §4.2.3.

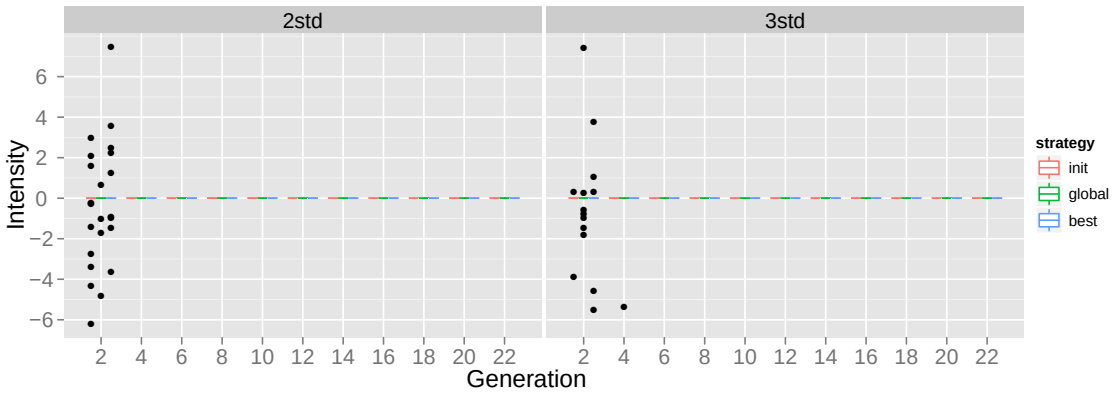


Figure A.52: Selection intensities for f_{lang} using Gaussian distributions of two and three standard deviations to sample rule intervals.

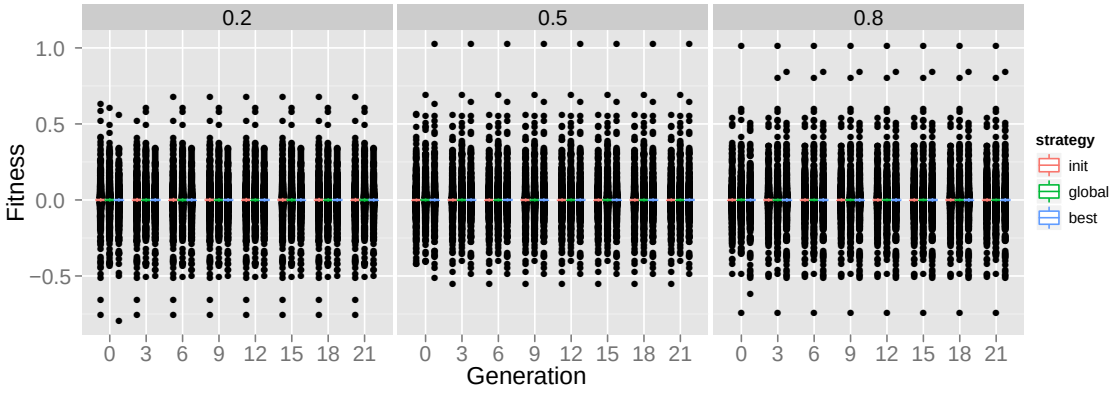


Figure A.53: Population trajectories for f_{lang} using a histogram-based distribution to sample rule intervals further broken down by budget dedicated to a uniform distribution.

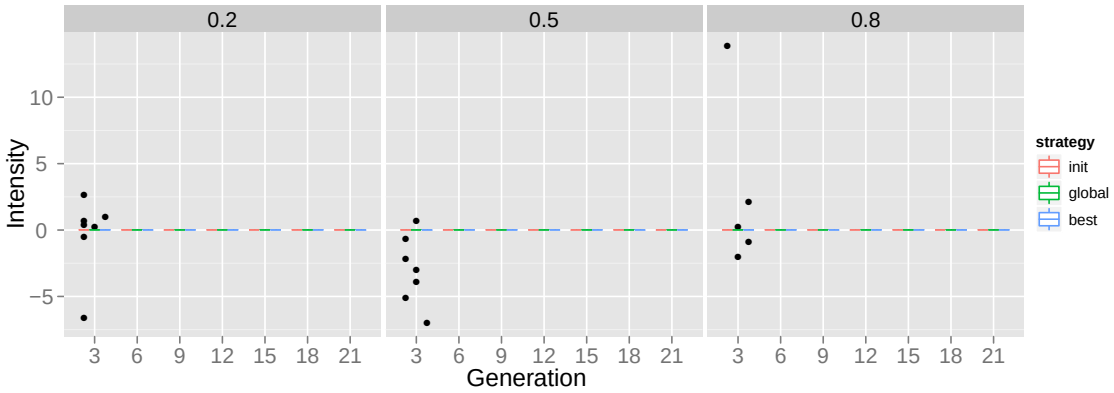


Figure A.54: Selection intensities for f_{lang} using a histogram-based distribution to sample rule intervals further broken down by budget dedicated to a uniform distribution.

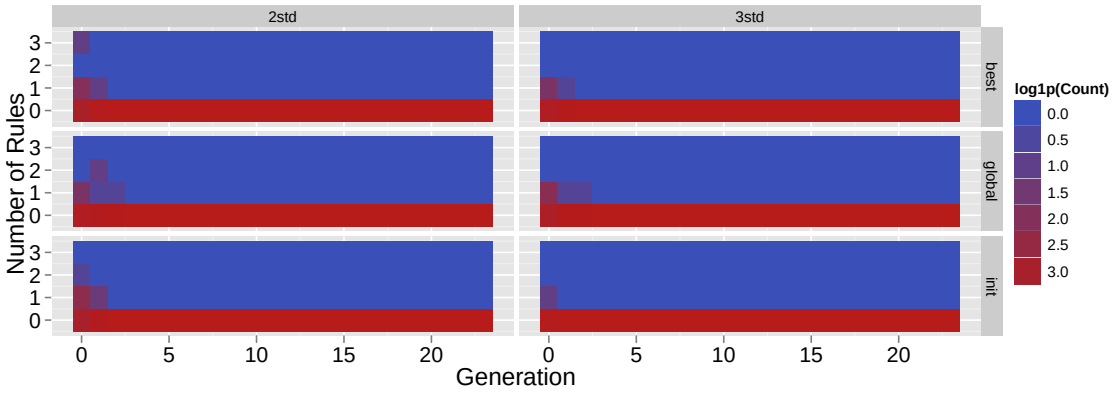


Figure A.55: Heat maps for the aggregate number of rules for f_{lang} using Gaussian distributions of two and three standard deviations to sample rule intervals.

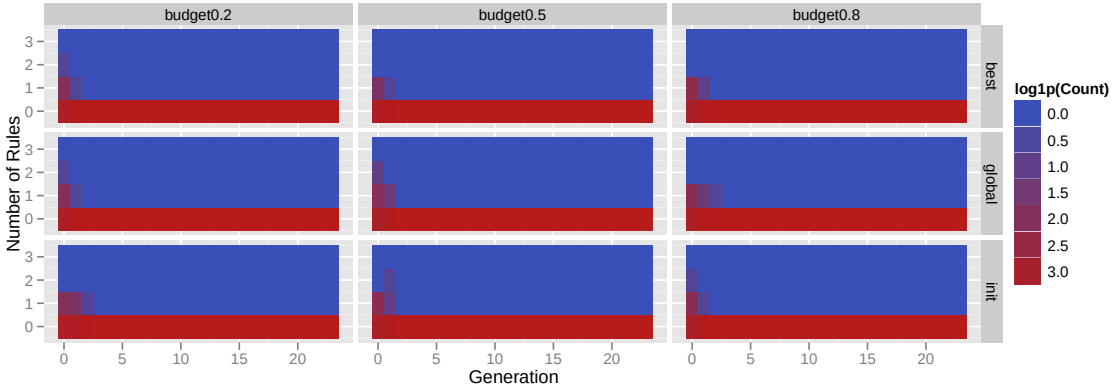
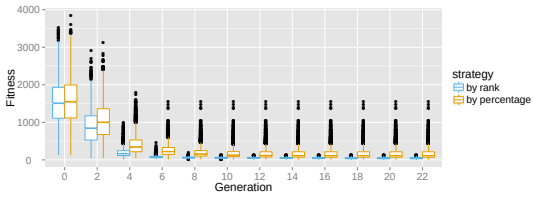
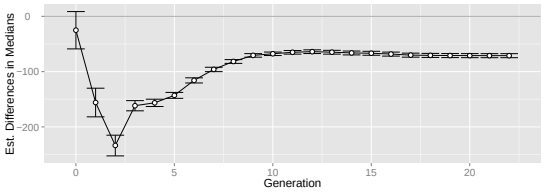


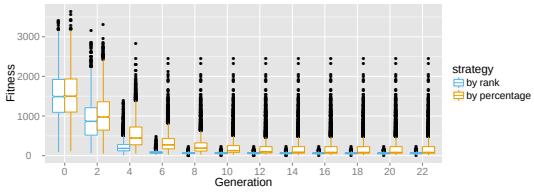
Figure A.56: Heat maps for the aggregate number of rules for f_{lang} using histogram-based distributions to sample rule intervals further broken down by budget dedicated to a uniform distribution.



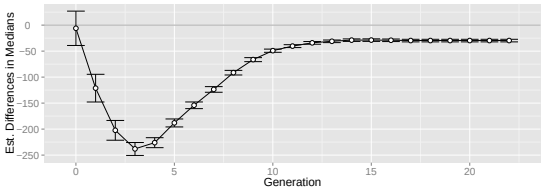
(a) This depicts the aggregate population trajectories for f_{rast} for the by-fitness-rank and by-fitness-percentage runs using the *gap* strategy.



(b) This shows the by generation differences between the rank and percentage fitness approaches for creating training sets for f_{rast} using the *gap* training set configuration. Values below zero favor by fitness rank runs, those above zero by fitness percentage.



(c) This depicts the aggregate population trajectories for f_{rast} for the by-fitness-rank and by-fitness-percentage runs using the *split* strategy.

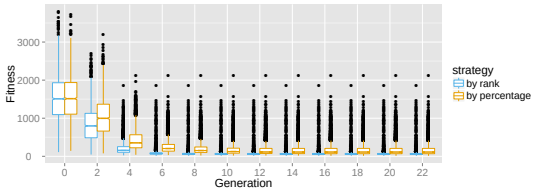


(d) This shows the by generation differences between the rank and percentage fitness approaches for creating training sets for f_{rast} using the *split* training set configuration. Values below zero favor by fitness rank runs, those above zero by fitness percentage.

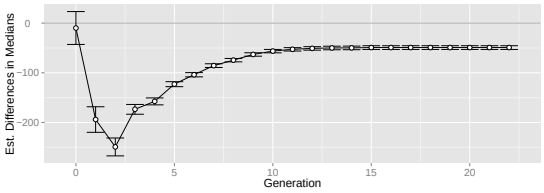
Figure B.1: This compares the by-rank and by-percentage of fitness runs for f_{rast} using the *gap* and *split* training set configurations, which shows that the by-rank runs converged more quickly to the global optimum througout all the runs.

Appendix B: Training Set Extra Material

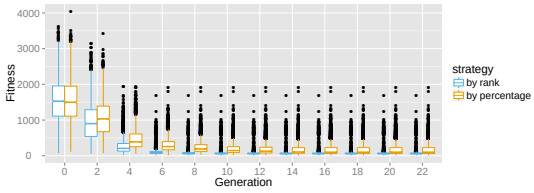
This appendix contains plots associated with Ch. 5.



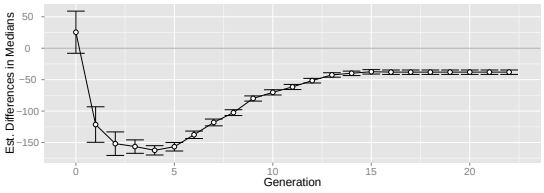
(a) This depicts the aggregate population trajectories for f_{rot_rast} for the by-fitness-rank and by-fitness-percentage runs using the *gap* strategy.



(b) This shows the by-generation differences between the rank and percentage fitness approaches for creating training sets for f_{rot_rast} using the *gap* training set configuration. Values below zero favor by fitness rank runs, those above zero by fitness percentage.



(c) This depicts the aggregate population trajectories for f_{rot_rast} for the by-fitness-rank and by-fitness-percentage runs using the *split* strategy.



(d) This shows the by-generation differences between the rank and percentage fitness approaches for creating training sets for f_{rot_rast} using the *split* training set configuration. Values below zero favor by fitness rank runs, those above zero by fitness percentage.

Figure B.2: This compares the by-rank and by-percentage of fitness runs for f_{rot_rast} using the *gap* and *split* training set configurations, which shows that the by-rank runs converged more quickly to the global optimum througout all the runs.

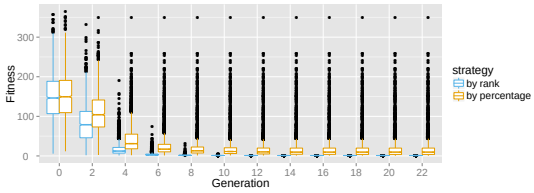
B.1 Ranked Vs. Percentage Plots

Rastrigin

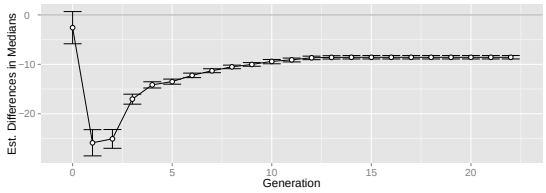
Fig. B.1 shows the results between the by-fitness rank and by-fitness percentage runs using the *gap* and *split* training set configurations for f_{rast}.

Rotated Rastrigin

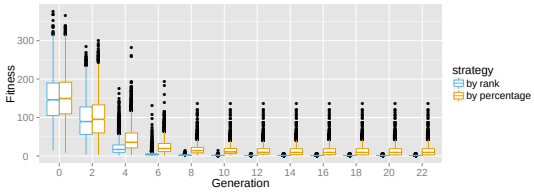
Fig. B.2 shows the results between the by-fitness rank and by-fitness percentage runs using the *gap* and *split* training set configurations for f_{rot_rast}.



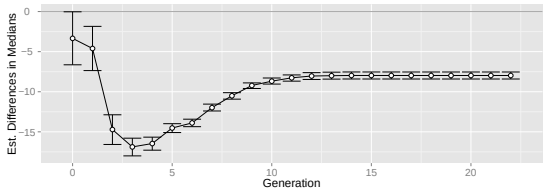
(a) This depicts the aggregate population trajectories for f_{grie} for the by-fitness-rank and by-fitness-percentage runs using the *gap* strategy.



(b) This shows the by-generation differences between the rank and percentage fitness approaches for creating training sets for f_{grie} using the *gap* training set configuration. Values below zero favor by fitness rank runs, those above zero by fitness percentage.



(c) This depicts the aggregate population trajectories for f_{grie} for the by-fitness-rank and by-fitness-percentage runs using the *split* strategy.



(d) This shows the by-generation differences between the rank and percentage fitness approaches for creating training sets for f_{grie} using the *split* training set configuration. Values below zero favor by fitness rank runs, those above zero by fitness percentage.

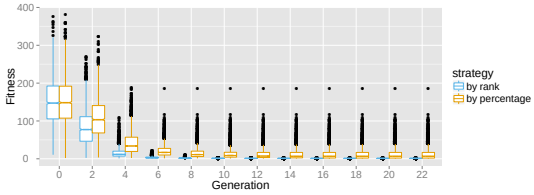
Figure B.3: This compares the by-rank and by-percentage of fitness runs for f_{grie} using the *gap* and *split* training set configurations, which shows that the by-rank runs converged more quickly to the global optimum througout all the runs.

Griewangk

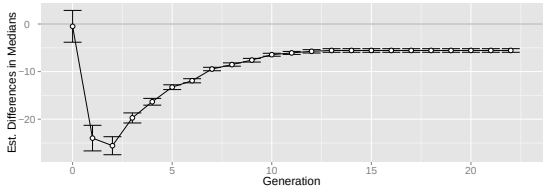
Fig. B.3 shows the results between the by-fitness rank and by-fitness percentage runs using the *gap* and *split* training set configurations for f_{grie}.

Rotated Griewangk

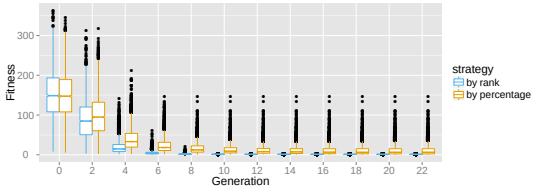
Fig. B.4 shows the results between the by-fitness rank and by-fitness percentage runs using the *gap* and *split* training set configurations for f_{rot_grie}.



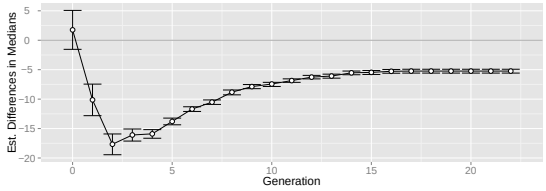
(a) This depicts the aggregate population trajectories for f_{rot_grie} for the by-fitness-rank and by-fitness-percentage runs using the *gap* strategy.



(b) This shows the by-generation differences between the rank and percentage fitness approaches for creating training sets for f_{rot_grie} using the *gap* training set configuration. Values below zero favor by fitness rank runs, those above zero by fitness percentage.



(c) This depicts the aggregate population trajectories for f_{rot_grie} for the by-fitness-rank and by-fitness-percentage runs using the *split* strategy.



(d) This shows the by-generation differences between the rank and percentage fitness approaches for creating training sets for f_{rot_grie} using the *split* training set configuration. Values below zero favor by fitness rank runs, those above zero by fitness percentage.

Figure B.4: This compares the by-rank and by-percentage of fitness runs for f_{rot_grie} using the *gap* and *split* training set configurations, which shows that the by-rank runs converged more quickly to the global optimum througout all the runs.

Bibliography

Bibliography

[Bäck et al., 1997] Bäck, T., Fogel, D. B., Michalewicz, Z., et al. (1997). *Handbook of Evolutionary Computation*. IOP Publishing Ltd.

[Baluja, 1994] Baluja, S. (1994). Population Based Incremental Learning: A Method for Integrating Genetic Search Based Function Optimization and Competitve Learning. Technical report, Carnegie Mellon University, Department of Computer Science.

[Baluja and Davies, 1997] Baluja, S. and Davies, S. (1997). Using Optimal Dependency-Trees for Combinational Optimization. In *ICML '97: Proceedings of the Fourteenth International Conference on Machine Learning*, pages 30–38, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

[Bersini et al., 1996] Bersini, H., Dorigo, M., Langerman, S., Seront, G., and Gambardella, L. (1996). Results of the first international contest on evolutionary optimisation (1st ICEO). In *Evolutionary Computation, 1996., Proceedings of IEEE International Conference on*, pages 611–615. IEEE.

[Blickle and Thiele, 1996] Blickle, T. and Thiele, L. (1996). A comparison of selection schemes used in evolutionary algorithms. *Evolutionary Computation*, 4(4):361–394.

[Cervone, 1999] Cervone, G. (1999). Learnable Evolution Methodology. Master's thesis, George Mason University.

[Cervone, 2012] Cervone, G. (2012). Personal communication. Regarding use of Gaussian probability distributions to sample LEM rule intervals.

[Cervone and Franzese, 2011] Cervone, G. and Franzese, P. (2011). Non-Darwinian Evolution for the Source Detection of Atmospheric Releases. *Atmospheric Environment*, 45(26):4497–4506.

[Cervone et al., 2010] Cervone, G., Franzese, P., and Gradjeanu, A. (2010). Characterization Of Atmospheric Contaminant Sources Using Adaptive Evolutionary Algorithms. *Atmospheric Environment*, 44:3787–3796.

[Cervone et al., 2000a] Cervone, G., Kaufman, K., and Michalski, R. (2000a). Experimental validations of the learnable evolution model. In *Evolutionary Computation, 2000. Proceedings of the 2000 Congress on*, volume 2, pages 1064–1071 vol.2.

[Cervone et al., 2002] Cervone, G., Kaufman, K., and Michalski, R. S. (2002). Recent results from the experimental evaluation of the learnable evolution model. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO)*.

[Cervone et al., 2000b] Cervone, G., Michalski, R. S., Kaufman, K., and Panait, L. (2000b). Combining Machine Learning with Evolutionary Computation: Recent Results on LEM. In *Proceedings of the Fifth International Workshop on Multistrategy Learning (MSL-2000)*, Guimaraes, Portugal.

[Chung, 1997] Chung, C.-J. (1997). *Knowledge-based approaches to self-adaptation in cultural algorithms*. PhD thesis, Wayne State University.

[Coletti, 2002] Coletti, M. (2002). Preliminary Results of Learnable Evolution Methodology (LEM) Using C4.5. In *Proceedings of the Congress on Evolutionary Computation*, Honolulu, Hawaii.

[Coletti, 2009] Coletti, M. (2009). Learnable Evolution Model Performance Impaired by Binary Tournament Survival Selection. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO)*, Montréal, Canada. Graduate Student Workshop.

[Coletti, 2012] Coletti, M. (2012). The Effects of Training Set Size and Keeping Rules on the Emergent Selection Pressure of Learnable Evolution Model. In *Genetic and Evolutionary Computing Conference Proceedings*.

[Coletti et al., 1999] Coletti, M., Lash, T., Mandsager, C., and Michalski, R. S. (1999). Comparing Performance of the Learnable Evolution Model and Genetic Algorithms on Problems in Digital Signal Filter Design. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO)*, Orlando, Florida.

[De Bonet et al., 1996] De Bonet, J. S., Isbell Jr, C. L., and Viola, P. (1996). MIMIC: Finding Optima by Estimating Probability Densities. In *Advances in Neural Information Processing Systems*, page 424. The MIT Press.

[De Jong, 1975] De Jong, K. (1975). *An Analysis of the Behavior of a Class of Genetic Adaptive Systems*. PhD thesis, University of Michigan.

[De Jong, 2006] De Jong, K. (2006). *Evolutionary Computation: A Unified Approach*. The MIT Press, The MIT Press, 55 Hayward St., Cambridge, MA 02142.

[Deb and Agrawal, 1999] Deb, K. and Agrawal, S. (1999). A niched-penalty approach for constraint handling in genetic algorithms. In *Artificial Neural Nets and Genetic Algorithms*, pages 235–243. Springer.

[Frank and Witten, 1998] Frank, E. and Witten, I. H. (1998). Generating Accurate Rule Sets Without Global Optimization. In Shavlik, J., editor, *Fifteenth International Conference on Machine Learning*, pages 144–151. Morgan Kaufmann.

[Friedman and Goldszmidt, 1996] Friedman, N. and Goldszmidt, M. (1996). Building classifiers using Bayesian networks. In *Proceedings of the national conference on artificial intelligence*, pages 1277–1284.

[Glover, 1989] Glover, F. (1989). Tabu search-part I. *ORSA Journal on computing*, 1(3):190–206.

[Google Scholar, 2014] Google Scholar (2014). Estimated number of Model-based Evolutionary Algorithms. `http://scholar.google.com/`. Search for memetic algorithm(s), estimation of distribution algorithm(s), cultural algorithm(s), and learnable evolution model.

[Guimaraes et al., 2007] Guimaraes, F., Campelo, F., Igarashi, H., Lowther, D., and Ramirez, J. (2007). Optimization of Cost Functions Using Evolutionary Algorithms With Local Learning and Local Search. *Magnetics, IEEE Transactions on*, 43(4):1641–1644.

[Hall et al., 2009] Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., and Witten, I. H. (2009). The WEKA Data Mining Software: An Update. *SIGKDD Explorations*, 11(1).

[Hansen and Ostermeier, 2001] Hansen, N. and Ostermeier, A. (2001). Completely Derandomized Self-Adaptation in Evolution Strategies. *Evolutionary Computation*, 9(2):159–195.

[Harik et al., 1999] Harik, G., Lobo, F., and Goldberg, D. (1999). The compact genetic algorithm. *Evolutionary Computation, IEEE Transactions on*, 3(4):287–297.

[Harik et al., 2006] Harik, G., Lobo, F., and Sastry, K. (2006). Linkage Learning via Probabilistic Modeling in the Extended Compact Genetic Algorithm (ECGA). In Pelikan, M., Sastry, K., and CantúPaz, E., editors, *Scalable Optimization via Probabilistic Modeling*, volume 33 of *Studies in Computational Intelligence*, pages 39–61. Springer Berlin / Heidelberg.

[Jones, 1995] Jones, T. (1995). *Evolutionary algorithms, fitness landscapes and search.* PhD thesis, Citeseer.

[Jourdan et al., 2004] Jourdan, L., Corne, D., Savic, D., and Walters, G. (2004). Hybridising rule induction and multi-objective evolutionary search for optimising water distribution systems. In *Hybrid Intelligent Systems, 2004. HIS '04. Fourth International Conference on*, pages 434–439.

[Kaufman et al., 1995] Kaufman, K., Bloedorn, E., Michalski, R. S., and Wnek (1995). Inductive learning system AQ15c: the method and user's guide. Technical report, George Mason University.

[Kaufman and Michalski, 1999a] Kaufman, K. and Michalski, R. (1999a). Learning from inconsistent and noisy data: the AQ18 approach. *Foundations of Intelligent Systems*, pages 411–419.

[Kaufman and Michalski, 1999b] Kaufman, K. and Michalski, R. S. (1999b). Learning from inconsistent and noisy data: the AQ18 approach. In *Foundations of Intelligent Systems, 11th International Symposium, ISMIS*, Warsaw, Poland.

[Krasnogor and Smith, 2005] Krasnogor, N. and Smith, J. (2005). A tutorial for competent memetic algorithms: model, taxonomy, and design issues. *Evolutionary Computation, IEEE Transactions on*, 9(5):474–488.

[Llora and Goldberg, 2003] Llora, X. and Goldberg, D. E. (2003). Wise breeding GA via machine learning techniques for function optimization. In *Genetic and Evolutionary Computation—GECCO 2003*, pages 1172–1183. Springer.

[Luke, 2008] Luke, S. (2008). Personal communication. Regarding use of combined parent and offspring populations for training sets for LEM.

[Luke et al., 2013] Luke, S. et al. (2013). ECJ 21: An Evolutionary Computation Library in Java. `http://cs.gmu.edu/~eclab/projects/ecj/`.

[Macready and Wolpert, 1995] Macready, W. G. and Wolpert, D. H. (1995). No free-lunch theorems for search. Technical report, Working Paper 95-02-010, Santa Fe Institute.

[Mascato, 1989] Mascato, P. (1989). On Evolution, Search, Optimization, Genetic Algorithms, and Martial Arts: Towards Memetic Algorithms. Technical report, Caltech Concurrent Computation Program, California Institute of Technology.

[Michalski, 1986] Michalski, R. (1986). Mozeti˜, I. Hong, J., and Lavra˜, N.(1986). The multipurpose incremental learning system AQ15 and its testing application to three medical domains. In *Proceedings of the Fifth National Conference on Artificial Intelligence*, pages 1041–1045.

[Michalski, 2000] Michalski, R. S. (2000). *Machine Learning*, volume 38, chapter LEARNABLE EVOLUTION MODEL: Evolutionary Processes Guided by Machine Learning, pages 9–40. Kluwer Academic Publishers.

[Michalski et al., 2006] Michalski, R. S., Wojtusiak, J., and Kaufman, K. A. (2006). Intelligent Optimization via Learnable Evolution Model. In *Tools with Artificial Intelligence, 2006. ICTAI '06. 18th IEEE International Conference on*, pages 332–335.

[Michalski et al., 2007] Michalski, R. S., Wojtusiak, J., and Kaufman, K. A. (2007). PROGRESS REPORT ON LEARNABLE EVOLUTION MODEL. Technical report, Machine Learning and Inference Laboratory, George Mason University.

[Michalski and Zhang, 1999] Michalski, R. S. and Zhang, Q. (1999). "Initial Experiments Learnable Evolution Model: An Application of LEM1 to Function Optimization and Evolvable Hardware". Technical Report MLI99-4, George Mason University, Machine Learning and Inferencing Laboratory.

[Mitchell, 1997] Mitchell, T. (1997). *Machine Learning*. The McGraw–Hill Companies, Inc.

[Mühlenbein and Paaß, 1996] Mühlenbein, H. and Paaß, G. (1996). From recombination of genes to the estimation of distributions I. Binary parameters. In Voigt, H.-M., Ebeling, W., Rechenberg, I., and Schwefel, H.-P., editors, *Lecture Notes in Computer Science 1141: Parallel Problem Solving from Nature - PPSN IV*, pages 178–187, Berlin. Springer-Verlag.

[Mühlenbein and Voigt, 1995] Mühlenbein, H. and Voigt, H.-M. (1995). Gene Pool Recombination in Genetic Algorithms. In *Metaheuristics: Theory and Applications*, pages 53–62. Kluwer Academic Publishers.

[Ortiz-Boyer et al., 2005] Ortiz-Boyer, D., Hervás-Martínez, C., and García-Pedrajas, N. (2005). CIXL2: A Crossover Operator for Evolutionary Algorithms Based on Population Features. *J. Artif. Intell. Res.(JAIR)*, 24:1–48.

[Pelikan et al., 1999] Pelikan, M., Goldberg, D. E., and Cantú-Paz, E. (1999). BOA: The Bayesian Optimization Algorithm. IlliGAL Report 98013, Illinois Genetic Algorithms Laboratory.

[Quinlan, 1986] Quinlan, J. R. (1986). Induction of decision trees. *Machine learning*, 1(1):81–106.

[Quinlan, 1993] Quinlan, J. R. (1993). *C4.5, Programs for Machine Learning*. Morgan Kaufmann Publishers.

[Radcliffe and Surry, 1994] Radcliffe, N. J. and Surry, P. D. (1994). Formal memetic algorithms. In *Evolutionary Computing*, pages 1–16. Springer.

[Ravisé and Sebag, 1996] Ravisé, C. and Sebag, M. (1996). An Advanced Evolution should not repeat its past errors. In *Proceedings of the 13th International Conference on Machine Learning*, pages 400–408.

[Reynolds, 1994] Reynolds, R. (1994). An Introduction to Cultural Algorithms. In *Proceedings of the 3rd Annual Conference on Evolutionary Programming*.

[Reynolds, 1999] Reynolds, R. G. (1999). Cultural Algorithms: Theory and Applications. In *New Ideas in Optimization*, chapter Cultural algorithms: theory and applications, pages 367–378. McGraw-Hill Ltd., UK, Maidenhead, UK, England.

[Salomon, 1996] Salomon, R. (1996). Re-evaluating genetic algorithm performance under coordinate rotation of benchmark functions. A survey of some theoretical and practical aspects of genetic algorithms. *BioSystems*, 39(3):263–278.

[Schlierkamp-Voosen, 1993] Schlierkamp-Voosen, D. (1993). Predictive models for the breeder genetic algorithm. *Evolutionary Computation*, 1(1):25–49.

[Schwefel, 1995] Schwefel, H.-P. P. (1995). *Evolution and Optimum Seeking*. John Wiley & Sons, Inc.

[Sebag et al., 1996] Sebag, M., Ravisé, C., and Schoenauer, M. (1996). Controlling Evolution by means of Machine Learning. *Evolutionary Programming*, pages 57–66.

[Sheri and Corne, 2008] Sheri, G. and Corne, D. (2008). The simplest evolution/learning hybrid: LEM with KNN. In *Evolutionary Computation, 2008. CEC 2008. (IEEE World Congress on Computational Intelligence). IEEE Congress on*, pages 3244–3251.

[Wallin and Ryan, 2009] Wallin, D. and Ryan, C. (2009). Evaluation of population partitioning schemes in bayesian classifier EDAs: estimation of distribution algoithms. In *Proceedings of the 11th Annual conference on Genetic and evolutionary computation*, GECCO '09, pages 469–476, New York, NY, USA. ACM.

[Whitley et al., 1995] Whitley, L. D., Mathias, K. E., Rana, S. B., and Dzubera, J. (1995). Building Better Test Functions. In *ICGA*, pages 239–247.

[Wojtusiak, 2004] Wojtusiak, J. (2004). The LEM3 Implementation of Learnable Evolution Model User's Guide. MLI 04- 5, Machine Learning and Inference Laboratory, George Mason University.

[Wojtusiak, 2007] Wojtusiak, J. (2007). *Handling constrained optimization problems and using constructive induction to improve representation spaces in Learnable Evolution Model*. PhD thesis, George Mason University.

[Wojtusiak, 2008] Wojtusiak, J. (2008). Data-driven Constructive Induction in the Learnable Evolution Model. In *Intelligent Information Systems*, pages 191–200. ISBN 978-83-60434-44-4.

[Wojtusiak et al., 2006] Wojtusiak, J., Michalski, R. S., Kaufman, K. A., and Pietrzykowski, J. (2006). The AQ21 natural induction program for pattern discovery: initial version and its novel features. In *Tools with Artificial Intelligence, 2006. ICTAI'06. 18th IEEE International Conference on*, pages 523–526. IEEE.

[Yuan and Gallagher, 2005] Yuan, B. and Gallagher, M. (2005). On the importance of diversity maintenance in estimation of distribution algorithms. In *GECCO '05: Proceedings of the 2005 conference on Genetic and evolutionary computation*, pages 719–726, New York, NY, USA. ACM.

Curriculum Vitae

Mark A. Coletti received his undergraduate degree in Computer Science at Southern College of Technology, Marietta, Georgia in 1989. Thereafter he worked as a software engineer in the Washington, DC, area on projects for the National Oceanic and Atmospheric Administration, Federal Highway Administration, U. S. Army's Materiel Command, the U. S. Army Topographic Engineering Center, and the United States Geological Survey. These projects included an expert system to correct human sourced sea surface meteorological data, an expert system for validating materiel purchases, a topographic visualization system, a road surface wear calculator, and a toolkit for spatial data format conversion. He also worked at George Mason University as a research associate and graduate research assistant where he helped develop an evolutionary computation C++ toolkit; a biologically inspired cognitive model for a DARPA Grand Challenge; a Joint Improvised Explosive Device Defeat Organization related multiagent simulation; an Office of Naval Research Multidisciplinary University Research Initiative Office sponsored massive multiagent simulation of pastoral and farming behavior in eastern Africa; and a geospatial extension, GeoMason, for the multi-agent simulation toolkit MASON. He graduated from George Mason University, Fairfax, Virginia with a Masters of Science in computer science in 2007.