Applications of Group Theory to Representation for Computational Intelligence

by

Jeremy Alexander Gilbert

A Thesis
presented to
The University of Guelph

In partial fulfilment of requirements
for the degree of
Doctor of Philosophy
in
Mathematics & Statistics

Guelph, Ontario, Canada

©Jeremy Alexander Gilbert, January, 2022

ABSTRACT

APPLICATIONS OF GROUP THEORY TO REPRESENTATION FOR COMPUTATIONAL INTELLIGENCE

Jeremy Gilbert
University of Guelph, 2022

Advisors:
Daniel Ashlock
Rajesh Pereira

This thesis introduces a novel approach to developing representations for evolutionary computation, using group theory as a foundation. The goal is to develop new representations which are better suited for navigating treacherous fitness landscapes, yielding improvements to algorithm performance over traditional methods. To construct such a representation, a selection of elements from a group are specified and used as generators to form a subgroup. The representation takes the form of words over the set of generators. An evolutionary algorithm is then able to search the space of words, which is a standard form of evolutionary algorithm. Multiple new representations are presented, built from additive vector groups, bijections of the unit interval, and affine transformations on Euclidean space. These representations can be used in a variety of applications, including real optimization, data normalization, image generation and modification, and point packing generation. Some can also be used to discretize a continuous search space, allowing the use of algorithms such as Monte Carlo Tree Search. The discrete nature of these representations also allows for use of a dictionary of previous optimal solutions. This permits

an algorithm to find a diverse set of best fit solutions, by using the dictionary to exclude parts of the search space near solutions that have already been found, realized as prefixes of stored words. A parameter study is performed for each representation, and they are compared to conventional methods on a variety of test problems.

Dedication

To all of my friends and family, both immediate and extended, who have helped me through the various phases of this journey. Thank you for always believing in me more than I could ever believe in myself, and pushing me to achieve what I never thought possible. This thesis would not exist without your love and support.

ACKNOWLEDGEMENTS

I would like to thank my advisors, Professor Daniel Ashlock and Professor Rajesh Pereira, for their incredible guidance throughout my time at the University of Guelph. The level of patience and care they show to their students is, to me, both unprecedented and something to aspire to. Thank you both for all of your advice over the years, and for your extraordinary help with my studies and this thesis.

I would also like to thank Natural Sciences and Engineering Research Council of Canada (NSERC) for supporting this work, providing the funding which made it all possible.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

This thesis is an exploration of how to pose a problem to a computer in order to make it easier to solve using evolutionary computation. Evolutionary computation, treated in detail in Section 2.2, is a simple digital instance of the theory of evolution. Instead of attempting to solve a problem directly, evolutionary computation attempts to evolve potential solutions to be of highest possible fitness when measured by an objective function. This problem is extremely general, and so to obtain a coherent topic for a thesis, I focused on one class of representations: those arising from algebraic groups.

After several years working on this project, it is clear that this, too, is more than will fit in a thesis. As the project went on, it became apparent that this was a veritable mine of new ways to represent problems. The thesis consists of a set of principles for deriving a representation for evolutionary computation from a group and a collection of examples, many of them published. The papers that have resulted from this project so far are:

A Discrete Representation for Real Optimization with Unique Search Properties (Ashlock and Gilbert (2014)). This study introduces a Walking Triangle Representation (referred to as "WTR") for real optimization, a representation which discretizes real space and is easily extended to higher dimensions. It is built around a subgroup of the affine transformations on Euclidean space, and is described in detail in Chapter 3.

Pairwise Irrational Point Packing Initialization (Ashlock and Gilbert (2019)). This study uses additive groups of vectors and a property of irrational numbers to quickly generate moderately high-quality point packings in the unit hypercube. Point packings are discussed in Section 2.3. These sets of points are then used for initial populations instead of the standard technique of randomly generating a set of points for an initialization. This method is described in detail in Chapter 5.

Evolvable Warps for Data Normalization (Gilbert and Ashlock (2016)). This study develops a new technique for for deriving an approximate inverse of the cumulative distribution function (CDF) of a set of data. The technique produces a continuous, invertible, and differentiable function, which in turn allows the retrieval of approximations of the actual CDF and the probability density function (PDF) by drawing from a subgroup of the bijections of the unit interval. This technique is described in detail in Chapter 6.

The various techniques explored in this thesis solve problems in real optimization, statistics, and even evolved art through the location and rendering of Julia Sets.

1.1 Motivation

Representation is a key part of evolutionary computation. Where tinkering with population size, mutation rate, crossover rate, and so on can squeeze out improvements of 10% to 50%, changing the representation can yield improvements up to orders of magnitude in speed (Ashlock et al. (2016)). This is referred to as *the representation problem*, and is discussed in more detail in Section 2.2.2.

When we choose the representations, we are, in effect, choosing the search landscape. This means we can eliminate low quality or worthless areas from the search space and it grants us control over connectivity. Unfortunately, this control is often neither straightforward nor obvious.

One example of an obvious change in representation to produce large improvements in both solution quality and time to solution involves the self-avoiding walk (SAW) problem. The SAW problem is a test problem which, in its simplest form, takes place on an n x m grid. The "player" begins in a corner and is allowed $nm - 1$ moves. The goal is to visit each position in the grid exactly once or, alternatively, to visit as many different positions in the grid as possible. An example of an evolved optimal SAW solution can be seen in Figure 1.1.

A typical representation for SAW would draw from a set of four possible moves $\{$Up, Down, Left, Right$\}$, moving the player's piece in that direction. This representation has a fairly obvious flaw: if the piece moves right followed by left, or up followed by down,

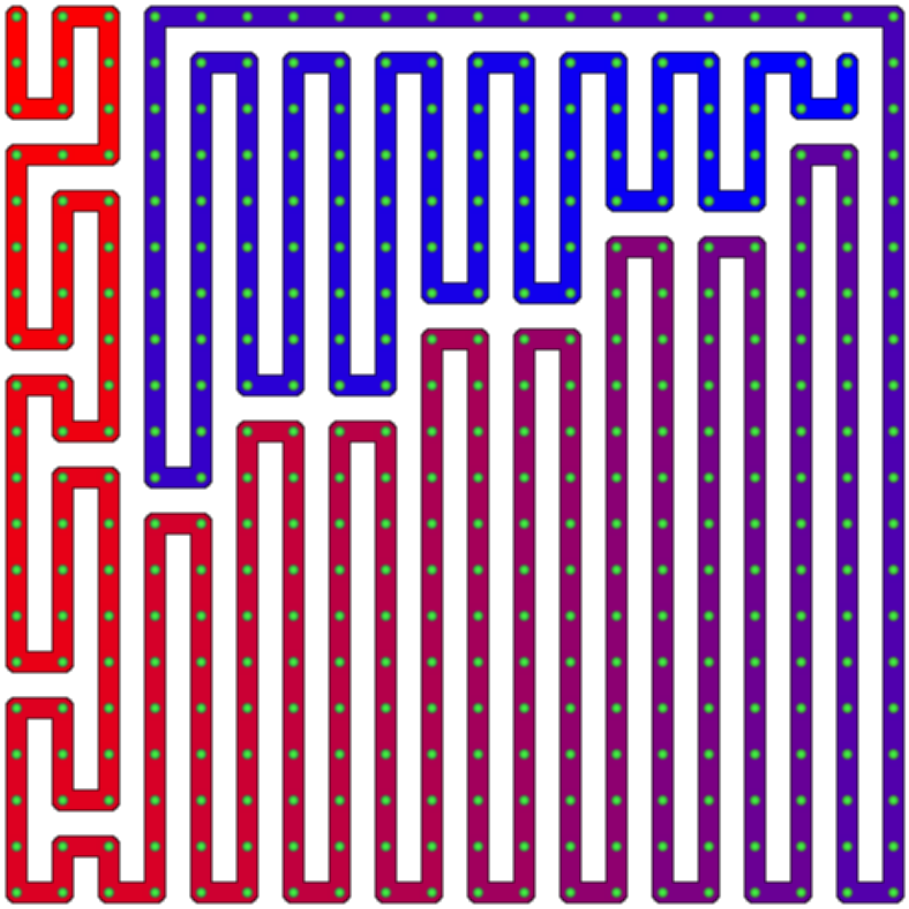


Figure 1.1: An example of an evolved optimal solution to the SAW problem.

it is necessarily wasting a move and therefore cannot lead to an optimal solution. The obvious change is then to have the representation draw instead from three possible moves {Advance, Left and Advance, Right and Advance}, where the piece either moves forward, turns left and moves forward, or turns right and moves forward (Ashlock and Montgomery (2016)). This simple change in representation will not eliminate the possibility of wasted moves, as the piece may still try to move into a wall or loop back and cross its own path. It will, however, prevent a move from being wasted by immediately reversing the previous

move. Therefore, this change in representation leads to a much smaller search space, where a very large number of sub-optimal solutions have been eliminated but all optimal solutions remain.

The SAW problem has the quality that it is generally quite easy for a human being to solve it. Typically, when faced with the SAW problem, most people would produce a solution along the lines of Figure 1.2 instead of the solution shown earlier in Figure 1.1. The fact that moving back to the space you just came from is necessarily a wasted move is obvious to us, but must specifically be programmed for a computer to recognize. Hence, the change in representation arises naturally. Most problems do not afford such a luxury.

One example of a less obvious, but extremely effective change in representation is the use of null operations in a linear representation (see Definition 1.1 below). Incorporating null operations in a linear representation allows mutation to insert or delete operations. This makes the number of potential mutants of a single gene much larger, which in turn increases the connectivity and decreases the diameter of the search space. The increase in connectivity leads previous local optima to be connected to higher-fitness solutions, thereby causing them to no longer be local optima. This can even collapse a large number of hills into a single hill in the search space, causing the fitness landscape to become much smoother and typically more friendly to an algorithm which is attempting to search it.

Definition 1.1 *In evolutionary computation, a representation is considered* **linear** *if it can*

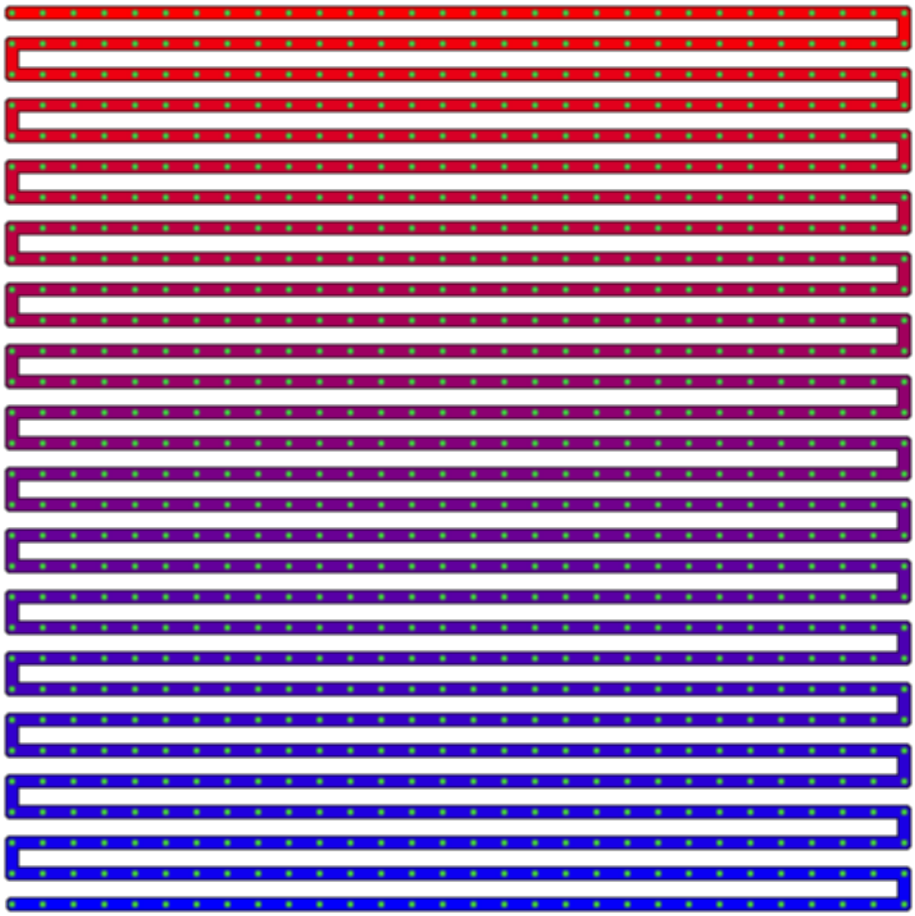


Figure 1.2: An example of an optimal solution to the SAW problem that a human being would typically produce.

be stored as a list of commands or parameters.

All of the representations presented in this thesis are linear, and null operations are used when appropriate. They are also contrasted with representations which are identical in every way except for a lack of null operations to demonstrate the advantages of their use. The use of group theoretic structures enables the construction of linear representations to be used on problems which have previously been addressed with non-linear representations, thereby allowing the representation to benefit from null operations.

In genetic programming, there are parts of code which are referred to as *introns*, sharing a name with their biological counterpart. Introns are parts of code which are redundant for the program solution (Brameier (2004)). Due to a variety of possible causes, introns have no effect on the calculation of the outputs for all possible inputs, a characteristic they share with null operations. Unfortunately, the similarities end there, as they serve otherwise different purposes in their respective roles. Introns are able to protect against disruption caused by crossover, which can improve the heritability of fitness. Null operations, on the other hand, allow dynamic selection of representation length. They have no particular role in protecting against crossover or improving heritability.

A common target of evolution is the location of orders for a set. The traveling salesman problem, which orders cities to minimize the circular tour distance, is an example of such an ordered gene problem. Much of the research on ordered gene problems operated directly on explicit orders; this was unfortunately a poor idea (Ashlock and Ashlock (2021)). Initial attempts to apply fairly standard crossover operators to ordered genes were problematic. This is because applying crossover to an ordered gene commonly does not yield an ordered gene and therefore requires repair operators. These repair operators themselves are usually disruptive, and often impair heritability, the ability to pass on good parts of a solution to other members of the population (see Section 2.2.2 for more on heritability).

The *adjacent transposition representation* is a representation for ordered genes that evolves a string of list elements (Hughes et al. (2014)). The representation starts with an ordered gene and then reads an instance of the representation as a series of elements to

swap. This means that the representation intrinsically must produce an order and so no repair operators are needed.

Notice that the adjacent transposition representation exploits a famous theorem of group theory, "every permutation is the product of adjacent transpositions". This is another example of how a knowledge of group theory can lead to substantial improvement in the design of representations for ordered gene problems.

These examples provide the motivation for the topic of this thesis: representations that arise from group theory. Please note that this is not the same as the theory of group representations from abstract algebra. The basic method is this:

1. Choose a group G.

2. Choose a finite list of elements; they generate a subgroup H.

3. Strings over the generators of H form the representation.

The ability to do this follows from Theorem 2.1 included in Section 2.1. For this to be effective, it helps to know that the desired solutions are in the subgroup H, which can be a difficult problem. One potential workaround to this problem is to develop a representation which has a set of solutions that are dense within the search space. One of the earliest projects included in this thesis evolved bijections of the unit interval (see Chapter 6). It is believed, although not proven, that this representation has a set of solutions which are dense within the search space, and limited only by the length of the representation.

The instance of WTR used in Chapter 3, the WUC representation, is a representation which is proven to have a set of solutions which is not only dense within $\mathbb{R}^n$, but exists almost everywhere (see Theorem 3.1 for proof). This is a powerful result, as it means that if the search space is any subset of real space, the question of whether a desired solution is in the generated subgroup can essentially be ignored. With a sufficiently long representation length, the WUC representation can come arbitrarily close to any potential solution, whether it is in the subgroup or not.

1.2 Organization of Thesis

The remainder of the thesis is structured as follows:

Chapter 2 contains background information on a variety of subjects in the context of this thesis. The information provided is neither an elementary introduction to, nor an in-depth discussion of the topic. Rather, the topic is introduced in the context in which it is used in this thesis in order to limit the scope of the topic to portions which are relevant here. It is intended for a reader with basic knowledge of the subjects, and references are provided which contain such basic knowledge and further reading.

Chapter 3 introduces the flagship study of this thesis: the Walking Triangle Representation (WTR). This study uses generators of a subgroup of the group of affine transformations on Euclidean space; this subgroup is the bijections of simplices in $\mathbb{R}^n$. These generators are used with an evolutionary algorithm to perform real optimization on a variety of test

problems, and are contrasted with an evolutionary algorithm performing real optimization using standard techniques. The WTR was the first group based representation studied, and was largely the motivation behind the theme of this thesis: using groups to construct representations.

Chapter 4 uses a variation of the representation used in the previous chapter, and performs complex parameter location for evolved art. It uses a technique to force diversity in the set of optimal solutions produced, and is shown to be highly effective in situations where the solutions desired are not just "good", but also noticably different from one another.

Chapter 5 is a study conducted using point packings of the unit hypercube (see Section 2.3). It uses an additive vector group to build a representation for very quickly finding good quality point packings. These packings are then used for population initialization for evolutionary algorithms instead of the standard method of initialization by randomly generating individual points. It is another example of when the most optimal solution is not necessary, but rather having a diverse set of good solutions is desired.

Chapter 6 studies a technique of data normalization using a representation built around the group of bijections of the unit interval. This technique is contrasted with standard techniques for determining the CDF of a set of data, or comparing different data sets using tools such as Q-Q plots. The data warps used to produce these results are then considered for other applications.

Chapter 7 outlines some areas of possible future work on the topics covered in the

thesis, and draws conclusions as a whole on the technique of using groups to construct representations for use in computational intelligence.

Chapter 2

Background

This thesis combines a number of diverse topics: these include group theory, evolutionary computation, point packing, and at least one form of the fractals known as Julia sets. These topics are used as building blocks for the methods or domains of application tested herein. In this chapter I set out basic facts from each of these areas to provide context. References are provided for readers that wish to delve deeper into each of these topics.

2.1 Group Theory

Groups are an algebraic object which are central to both applied and abstract algebra. Group theory has applications in many physical sciences, and plays an integral role in modern cryptography. Although there are far more recent examples, one notable application of group theory is the infamous Enigma machines used primarily during the second World War (Rejewski (1982); Christensen (2007)). These machines were designed and built using

groups of permutations. The quality of the encryption and the robustness against standard decryption techniques of its era were the greatest strengths of the Enigma machines; this was coupled with the ability for everyday members of the military with little or no knowledge of mathematics or cryptography to operate them. These powerful attributes were made possible by the ability of group theory to place a navigable structure on what appears to the naked eye to be quite chaotic. It is this property of group theory, coupled with an affinity for the subject, which has led me to following a similar path in the development of group theoretic representations for evolutionary algorithms.

The definition of a group is somewhat broad, allowing many structures to be classified as groups. This, in turn, allows the results obtained from studying groups in general to be applied in a broad scope.

Definition 2.1 *A **group** is defined as any nonempty set G with a binary operation $\circ$ which together satisfy three axioms:*

1. *The operation $\circ$ is associative: for all $a, b, c \in G$, we have $a \circ (b \circ c) = (a \circ b) \circ c$.*
2. *There is a two-sided **identity** element $e \in G$ such that for all $g \in G$ we have that $g \circ e = e \circ g = g$.*
3. *For each $g \in G$, there exists a two-sided **inverse** $g^{-1} \in G$ such that $g \circ g^{-1} = g^{-1} \circ g = e$.*

*These three axioms are sufficient for $(G, \circ)$ to be a group. However, $(G, \circ)$ is said to be **commutative** or **Abelian** if it also satisfies a fourth axiom:*

4. The operation $\circ$ is commutative: for all $g, h \in G$ we have $g \circ h = h \circ g$.

This thesis includes both commutative and non-commutative groups. There are some elementary results from group theory which are quite useful in this thesis. The first is foundational to the representations developed herein, but requires a definition:

Definition 2.2 *Let $(G, \circ)$ be a group, and let X be a nonempty subset of G. Let $\{H_i | i \in I\}$ be the family of all subgroups of G which contain X. Then $\bigcap_{i \in I} H_i$ is called the **subgroup** of G **generated by the set X** and is denoted $\langle X \rangle$.*

Theorem 2.1 *Suppose $(G, \circ)$ is a group and X is a nonempty subset of G. Then the subgroup $\langle X \rangle$ generated by X consists of all finite products ${x_1}^{n_1} \circ {x_2}^{n_2} \circ \cdots \circ {x_t}^{n_t} (x_i \in X; n_i \in \mathbb{Z})$.*

This means that for any group $(G, \circ)$, a subgroup H can be generated by simply choosing elements of G, and then closing them under the group operation $\circ$.

The next result is not so much foundational to the strategy employed for building representations in this thesis; rather, it simply provides an easy method of finding groups which can be drawn from to build a representation.

Theorem 2.2 *The set of all bijections of a set forms a group under functional composition.*

This means that for any set, there is immediately an associated group formed by taking all bijections of that set under functional composition. The group used in Chapter 6 arises

this way, and is discussed in Section 2.1.3. Proofs for these theorems and further reading on the topic can be found in (Hungerford (1974)).

With the large variety of groups and their different behaviours, it is natural that certain groups will be better attuned to solving certain types of problems. Sometimes the relationship between the group and the problem is obvious, such as in Chapter 6. In this chapter, I am attempting to locate cumulative distribution functions, which can be approximated by bijections of the unit interval. Therefore, I use elements of the group of bijections of the unit interval to form the representation. Other times, the relationship is less obvious, such as in Chapter 3, where I use elements of the group of affine transformations on Euclidean space to form a representation for performing real optimization. Finding the best group for a given problem can be a taxing endeavour, but I believe that even a good or even mediocre relationship between a group and a given problem can yield noticeable improvements over traditional methods.

The groups used in this thesis are introduced briefly in the following subsections.

2.1.1 The Group of Affine Transformations on Euclidean Space

The group of affine transformations on Euclidean space is a somewhat famous group (Ewald (1971)) which consists of the set of all functions of the form $\vec{v}(\vec{x}) = A\vec{x} + \vec{b}$ where A is an invertible matrix, $\vec{x}$ is a vector variable, and $\vec{b}$ is a vector constant. These functions form a group under functional composition. This is one example of a group which arises from Theorem 2.2.

Definition 2.3 *A set of n points $\{x_1, x_2, \ldots, x_n\}$ in $\mathbb{R}^d$ is said to be* **affinely independent** *if, for any $\lambda_1, \lambda_2, \ldots, \lambda_n \in \mathbb{R}$, the conditions $\lambda_1 x_1 + \lambda_2 x_2 + \cdots + \lambda_n x_n = 0$ and $\lambda_1 + \lambda_2 + \cdots + \lambda_n = 0$ together imply that $\lambda_1 = \lambda_2 = \cdots = \lambda_n = 0$.*

Definition 2.4 *A* **d-dimensional simplex** *or* **d-simplex** *is the convex hull of a set of $d + 1$ affinely independent points in $\mathbb{R}^d$.*

A simplex in two dimensions is a triangle; in three dimensions it is a tetrahedron, and so on. The WTR takes its name from this, as one of the transformations can be thought of as causing the simplex to "walk" around in $\mathbb{R}^d$. Note that defining the vertices of the simplex to be a set of affinely independent points precludes *degenerate* simplices, as they are not used by the representations described herein. The WTR discussed in this thesis only uses at most three types of transformations, one which is self-inverse and two others which are an inverse pair, each applicable to any vertex for a total of $3(d + 1)$ transformations in d dimensions. The solutions it can then represent are found in the group generated by these three types of transformations, as described in Theorem 2.1.

The choice of these three types of transformations allows for the generation of a discrete subgroup, and it is proven that the points which can be represented by this particular WTR exist almost everywhere within $\mathbb{R}^n$ (see Theorem 3.1). The ability to discretize a continuous space is one of the representation's greatest assets. This is explored further in Chapters 3 and 4.

2.1.2 Additive Vector Groups

Every vector space has an embedded Abelian group by considering its vectors under vector addition. It very conveniently inherits the group axioms by being a vector space: closure is inherited, associativity is inherited, the zero vector acts as identity, and inverses are inherited. Of course, vector addition is known to be commutative, satisfying the final axiom of an Abelian group.

While this thesis does not use many of the group properties of an additive vector group, it does use a useful feature of irrational numbers to construct a vector which can generate a dense set of points within a vector space. These points are then filtered to create a point packing (see Section 2.3) which is used in turn as the initial set of points for an evolutionary algorithm. This project is discussed in Chapter 5.

2.1.3 Group of Continuous Bijections of the Unit Interval

The group of continuous bijections of the unit interval consists of all continuous bijections of [0,1]. These can be characterized as either *increasing* whose graphs go from (0,0) to (1,1), or *decreasing* whose graphs go from (1,0) to (0,1). It is another example of a group which arises from Theorem 2.2. This group is used in Chapter 6, where the goal is to find the cumulative distribution function (CDF) of a set of data by approximating its inverse. The CDF of any finite set of data is itself an increasing function with range $[0, 1]$. If the domain is *not* $[0, 1]$, a simple affine transformation can place it in this interval. If it is *strictly* increasing, then it will now be a bijection of the unit interval, as will its inverse.

If it is not strictly increasing then it will lose its injective property and its inverse will no longer be a function, but both the CDF and its inverse can still be very closely approximated by a bijection. Thus, the selection of this group is forcing the algorithm to find potential approximate inverse CDFs, ensuring that the evolved solutions are relevant.

Once again, two elements of this group are chosen to be generators for the representation. In this case, however, these generators were paired with a real parameter. This real parameter was fairly tightly controlled in order to restrain the complexity of the representation, but is also used to provide inverses. Both types of generators are families which are closed under inversion; the inverse of a generator can be computed by taking the reciprocal of the real parameter that it is encoded with. These real parameters also allowed for finer tuning of the string of generators used for a given solution, producing higher fitness solutions that would otherwise have needed a more diverse set of generators to produce. Essentially, the added complexity of having a large set of generators was exchanged for having a single real parameter instead, and this exchange is believed to have resulted in a simpler representation which produced higher-quality solutions.

2.2 Evolutionary Computation

Evolutionary computation is an area of computer science which utilizes evolutionary algorithms to solve a wide variety of problems (Ashlock and Ashlock (2021)). The basic mechanisms of an evolutionary algorithm are motivated by the mechanisms of Charles Dar-

win's theory. Rather than attempting to directly solve a problem, evolutionary algorithms use a guided trial-and-error method to test solutions against a given fitness function to determine the solution's quality. A basic example is given as Algorithm 2.1. Evolutionary algorithms exist in many forms, but are most often derived from the following basic design: There is a pool of solutions to draw from, referred to as a population. Members of this population, called parents, will be randomly selected. The solution information they contain will be mixed together, along with randomly generated new (or altered) solution information, to create children. The mixing is referred to as crossover, while the randomly-generated information is referred to as mutation. The representation used in a given evolutionary algorithm is the choice of data structure used to hold potential solutions. Together with the mutation and crossover operators, it defines the topology of the space being searched. Evolutionary computation has proven to be very useful in solving optimization problems where there is a relatively obvious test for a good solution, but the function which needs to be optimized is either unknown, or is a function which is prohibitively difficult to optimize using conventional methods.

Algorithm 2.1 Main loop of an evolutionary algorithm

Initialize a population of structures

Evaluate the quality (fitness) of each structure

Repeat

Pick parents with a quality bias

Generate child structures, evaluate their quality

Pick decedents

Conditionally replace decedents with children

Until (Good Enough)

One natural application for evolutionary algorithms is real parameter optimization. In its simplest form, an evolutionary algorithm designed for real optimization utilizes population members which are points in Euclidean space, and their coordinates can be evolved. Real optimization is a branch of mathematics which is massive in scope and highly applicable to real world problems. Evolutionary computation excels on optimization problems where the function is not differentiable or, even worse, where the function being optimized is computed via a complex simulation. It has been used in the design of airfoils and helicopter rotors to optimize the amount of lift obtained from a given power input (Benini and Toffolo (2002)), and for VLSI (very large scale integration) microchip design layout, optimizing the layout of microchips to minimize the number of crossings that require external jumper wires (Linhares (1999)). Another application of evolutionary computation is programming artificial neural networks used for mammogram annotators, allowing radiographic images to undergo a prefilter to accent areas of concern before being read by a radiologist, greatly reducing the amount of time required of the doctor (Fogel et al. (1998)).

An important realization in evolutionary computation research is that the representation

used has enormous impact on algorithm performance, which is discussed in more detail in Section 2.2.2. The research in this thesis focuses on novel representations which incorporate group theory and allow the theory to guide the development of the representation.

2.2.1 Exploration and Exploitation

In characterizing the behaviour of an algorithm, two common traits are often considered: exploration and exploitation. Typically they are described in terms of hill-climbing. Exploration is the ability of an algorithm to find new hills, and exploitation is the ability of an algorithm to climb a hill that it has found. There is often a trade-off between the two; the more inclined an algorithm is to find new hills, the less inclined it generally is to climb those hills. In evolutionary algorithms, the exploratory and exploitative behaviours of an algorithm rely first on the representation used, and then on tuning the parameters used during genetic mixing and mutation. Due to the fact that these types of behaviour are closely tied to the representation used with an algorithm, they are discussed in further detail below.

2.2.2 The Representation Problem

As was mentioned in Section 2.2, more attention is being paid to the effects of different representations on an algorithm's behaviour and efficacy. The authors of (Ashlock et al. (2006)) were experimenting with different methods of storing solution information used for evolving game playing agents for Iterated Prisoner's Dilemma and found that depending on which representation was used, the resulting agents exhibited wildly differing behaviours.

This is referred to as the *representation problem* within evolutionary computation. This important discovery provides much of the motivation for the development of new representations found in this thesis. Examples of potential changes to representations for simpler problems was discussed in Section 1.1.

The representation used with a given evolutionary algorithm affects all aspects of the algorithm's behaviour. It tends to have a large effect on whether the algorithm is more exploratory or exploitative, and also affects the ability of parameter tuning to control or change these behaviours. Typically it is best to have an algorithm which has both exploratory and exploitative properties, but which one should receive more bias is usually dependent on the problem. Sometimes an algorithm is inherently biased to one or the other, and a slight change in representation may be required to maintain the necessary balance. Indeed, in Chapter 4, the representation is designed to have the algorithm behave in a purely exploitative manner, with a small addition to the representation forcing the algorithm to necessarily climb hills in different sections of the fitness landscape. This small addition provides the required exploratory power to prevent the algorithm from repeatedly attempting to climb the same hill.

Another large impact that a representation has on a given evolutionary algorithm is the *heritability* of its solutions. This refers to how easily the high quality portions of a population member are passed to offspring. Good representations will make it easier to pass parts of a solution which contribute to fitness along, and if possible also make it difficult for pieces that detract from fitness from being passed along. An example of this is

the use of one-point crossover with highly epistatic representations (see definition below), as the use of multi-point crossover has a much higher chance of splitting up strings of genes which are only effective as a whole.

Definition 2.5 *A representation exhibits* **epistasis** *if the values of parameters used in that representation are not necessarily good nor bad, but rather the quality of the value of a parameter can only be evaluated in the context of the values of other parameters. This parallels the biological notion of epistasis where the expression of the allele of one gene can be masked by the allelic value of a different gene.*

Given that the effectiveness of a representation is at least partially dependent on the problem at hand, it is natural to believe that the effectiveness of the representations developed herein is related to the test problems they attempted to solve. However, this points to another strong motivation for the projects covered by this thesis. The problem-specific nature of effective representations, combined with the relative novelty to building representations from group structures, is what motivates experiments of this type. Theory can be used to an extent to make informed decisions in the design of a representation, but to find true trends in the relationship between representations and problems they are effective on requires experimentation.

2.3 Point Packing in the Unit Hypercube

Finding a collection of n points in the unit square that maximize the minimum distance between any two points (Croft et al. (1991)) is a surrogate for the *two-dimensional stock cutting problem*. This consists of placing items on a sheet of material to be stamped or cut as efficiently as possible. Point packings and its extensions are areas of active research in computational intelligence and adaptive computation (Burke et al. (2004); Terashima-Marín et al. (2005); Grzegorek et al. (2020)).

Definition 2.6 *The* **point packing problem in the unit hypercube** *or* **PPUH problem** *seeks to place a specified number of points into the unit hypercube, including its boundary, so that the minimum distance between any pair of points is maximized.*

A web repository[1] gives the current best known values for the point packing problem in the unit square, which is the two-dimensional unit hypercube. The *distance* column of the table at this website contains the best known fitness.

2.3.1 Point Packings for Population Initialization for Real Optimization

While packing points into a unit hypercube arises from a real world problem, and also makes for an interesting test problem for optimization, it is used slightly differently here. An optimal solution will pack as many points as possible into a given unit hypercube, but

[1] http://hydra.nat.uni-magdeburg.de/packing/csq/csq.html

even a *good* solution will provide a set of well-spaced points. Of course, it is substantially easier (read: computationally less intensive) to get in the relative vicinity of an optimal solution than it is to find an actual optima, and as such, I set about finding a relatively cheap method of generating *good* solutions that may not necessarily be optimal. The purpose of this was to generate a set of points to be an initial population to then be used with an evolutionary algorithm, instead of the standard method of randomly generating the initial population of points.

As was mentioned in Section 1, and as is implied by its name, evolutionary computation seeks to *evolve* a solution to a problem. This, of course, necessitates some existing, possibly low-quality solutions which can be evolved. When performing real optimization with evolutionary computation, it is standard operating procedure to produce a randomly-generated set of points inside a given search domain as an initial population of possible solutions for the algorithm to then evolve. Another common technique employed is to use a grid on the search space to determine the initial set of points. Use of a regular grid suffers greatly from the curse of dimensonality, making it impractical for use in higher dimensions. Using a point packing for an initial population is more similar to the grid-based initialization than using a randomly-generated set, but with some differences that will help it scale better as dimension increases.

Optimal point packings tend to resemble hexagonal grids or their higher-dimensional analogs (Chang and Wang (2010)). This makes the claim that they are better than a standard grid seem implausible. The key fact is this: either a grid or a point packing can have its

interval (distance between points) increased to reduce the number of points. However, point packings cover space more efficiently than rectangular grids, and the parameter of minimum spacing can be continuously adjusted to permit a number of points near the limit of the optimization system. Point packings also suffer from the curse of dimensionality, but are more flexible.

Deceptive and highly multi-modal fitness landscapes are typically dealt with by using a large population size or potentially costly diversity promotion measures. Expensive fitness functions require careful allocation of fitness trials. I performed a study to test the relatively inexpensive method of finding a point packing for improving performance on both deceptive and highly multi-modal landscapes that may also be helpful for expensive fitness functions. The technique relies on the ergodicity of irrational numbers in the unit hypercube to create an evenly distributed initial population in a rectangular search domain. This technique was tested on the problem of optimizing several cases of two deceptive and two highly multi-modal functions. It was found to be helpful on all tested functions.

2.4 Julia Sets

There are many types of Julia sets which are a workhorse of complex dynamics (Julia (1918)). They are mostly, however, beyond the scope of this thesis, which only uses a quadratic Julia set. I begin with a standard Julia set for the function $f(w) = w^2 + z$. This type of Julia set takes a single complex parameter z. For any complex number w the

following series is generated:

$$w_1 = w \tag{2.1}$$

$$w_{k+1} = w_k^2 + z \tag{2.2}$$

If the sequence contains an element $|w_i| \geq 2$ then the smallest number i for which this is true is returned as the *iteration number* of w. If the sequence does not ever contain an element of magnitude two or more then w is part of the Julia set. In practice an upper bound of $i < I = 120$ is placed on the number of iterations; points whose series do not contain an element of magnitude 2 by step 120 are considered to be in the Julia set – and must be very close to points in the fractal. The iteration number of these points that are in, or considered to be in, the fractal is reported as 120.

We now generalize the quadratic Julia set by granting it N parameters $\{z_1, z_2, \ldots, z_N\}$. Aside from using the parameters in rotation, this Julia set is generated in the same way with the sequence:

$$w_1 = w \tag{2.3}$$

$$w_{k+1} = w_k^2 + z_{(k \bmod N)} \tag{2.4}$$

The generalization permits a much broader collection of appearances and has been found experimentally to permit a Julia set that has a positive area in the complex plane but is not

connected (Ashlock and Jamieson (2007)).

2.4.1 The Julia Set Fitness Landscape

In this thesis, the goal of evolution is to produce aesthetically-pleasing Julia sets. Unfortunately, there is no obvious fitness function for "looks good when rendered", and so Equation 4.1, an entropy-based fitness function defined in Section 4.2.1, is used in an attempt to produce interesting-looking fractals. The theory of fractal geometry (Mandelbrot (1983); Barnsley (1988)) is also beyond the scope of this thesis, but I borrow from it some facts that highlight features of the fitness landscape for the Julia set parameter location fitness function. Standard Julia sets are indexed by the Mandelbrot set (Mandelbrot (1983)). The Mandelbrot set itself has new features at each level of resolution. The indexing effect means that the search space of Julia sets, even before generalization, contains an infinite number of optima with this fitness function. Since standard Julia sets are present in the space of generalized Julia sets (by setting all the defining parameters equal to one another) it follows that the fitness landscape has an infinite number of Julia sets with distinct appearances. This suggests that the fitness landscape has a roughness limited only by the numerical precision of the machine it is being run on. It may help one appreciate the rugosity of this fitness function to examine the plot of the value of the fitness function along line segments in $\mathbb{R}^4$, shown in Figure 2.1. These graphs show the fitness value at 400 evenly-spaced points along the line segment joining the point (0.5,0.5,0.5,0.5) to three evolved sets of two-complex parameter Julia sets. The evolved sets are: (-0.0339221,-

0.0281978,0.497533,0.622487) for graph 1, (-0.0317053,-0.388248,-0.0108944,0.285822) for graph 2, and (-0.0465953,-0.0156858,0.499458,0.614495) for graph 3. While there are flat spots where the fitness does not change for a length of the line segment, the fuzzy areas are the result of extremely rapid changes in fitness in a relatively small portion of the line segment. The best fit solutions typically exist at points in these fuzzy regions, and this roughness exists in all four or six dimensional versions of the fitness landscape, that is for two- and three-complex parameter Julia sets, respectively.

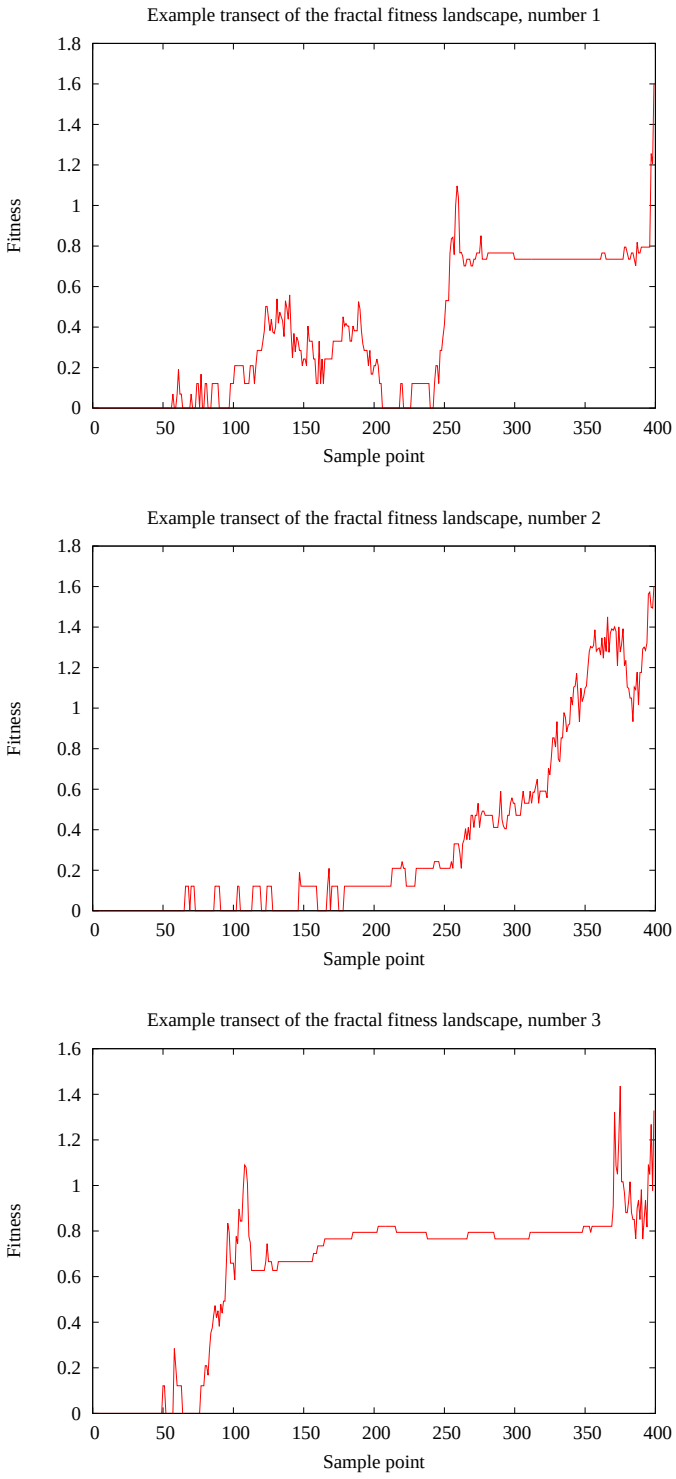


Figure 2.1: Shown are three different fitness transects running from a region of low fitness, at point (0.5,0.5,0.5,0.5), to three different sets of evolved Julia parameters.

Chapter 3

The Walking Triangle Representation

3.1 Introduction

Walking Triangle Representations (WTRs) for real optimization are linear representations constructed with elements of the group of affine transformations which act on simplices in Euclidean space. Beginning with an initial simplex, the representation encodes a series of commands to move said simplex around Euclidean space, and the point represented by a set of commands is the centroid of the resulting simplex after the commands are applied in order. Controlling the commands available, as well as placing restrictions on the order in which they can be applied, provide additional control over an algorithm's strength in exploration or exploitation. Some of these controls have not been implemented in the version in this chapter, although one variation is explored in Chapter 4. This study was based around a WTR using three possible commands: walk, center, and uncenter (WUC).

Their definitions and properties are included in Section 3.2.

The original idea to form a representation where a simplex moves around real space was inspired by the *Nelder-Mead simplex method* for minimization (Nelder and Mead (1965)). Nelder-Mead also uses a simplex moving around real space; however, it tracks the value of the function at each vertex rather than just one value at the centroid. It is a direct search method typically employed to find a local optimum of a problem in finite dimensions where the objective function is varying smoothly and is unimodal. It is typically implemented to minimize an objective function, sometimes being referred to as the *downhill simplex method*. Of course, if it is required to maximize a function $f(x)$ instead, Nelder-Mead can be used to minimize $-f(x)$.

Evolutionary algorithms used for real optimization typically use a population of points in $\mathbb{R}^n$ (Ashlock (2006)). A common modification uses a representation of the points derived from a binary string. An example of a more advanced modification uses *self adaptation* which permits the evolution of parameters which control the algorithm's search behaviour via mutation, above and beyond the evolution performed on the points themselves (Meyer-Nieberg and Beyer (2006)). Another common technique is differential evolution (Das and Suganthan (2011)), which has shown better performance than standard evolutionary computation in some domains. With this technique, controlling the degree of exploration and exploitation largely stems from the type of random distribution used for mutation, and, to a lesser extent, population size. Within the group-based representations studied, crossover can aid in exploration early in evolution, but its effect typically diminishes over the course

of evolution, as the population's diversity decreases. In this they are similar to basic evolutionary algorithms which used a string-based representation. At the same time, exploitation will begin to ramp up, as selection works to lower the population's diversity.

Choosing between uniform, Gaussian, or other distributions will often require experiments and parameter tuning, which can be time consuming. One benefit of WTRs is that the discretization of a continuous space entirely removes the need to select a probability distribution for mutation. WTRs are examples of *generative* representations (Hornsby et al. (2003)): rather than directly encoding points (potential solutions), they encode instructions for constructing potential solutions, and allow evolution to work on this set of instructions. With WTRs, the encoded point is the point that results after this set of instructions, or commands, are applied to an initial point to modify its location.

3.2 The Representation

The word "triangle" in "Walking Triangle Representation" is actually the specific two-dimensional case of the object used in this representation, a d*-dimensional simplex*, which is defined in Definition 2.4 found in Section 2.1.1.

There are theoretically an abundance of possible operations which could be used to create a walking triangle representation. This study uses only three: *walk, center,* and *uncenter*, and is therefore referred to as the "WUC representation". *Walk* is self-inverse, while *center* and *uncenter* are inverses of each other, and their exact definitions will fol-

low. These operations are defined by using standard vector arithmetic on the vertices of a simplex. Each operation can be applied to any of the vertices, and have a different effect on the simplex. Therefore, *walk, center,* and *uncenter* are denoted as w_k, c_k and u_k, respectively, where k is the index of the vertex which the operation is being applied to. To give definitions for these operations, consider a simplex S in d dimensions, making the set of vertices $S = \{v_0, v_1, \ldots, v_d\}$. Examples of the three operations being applied to a vertex in two dimensions is given in Figure 3.1.

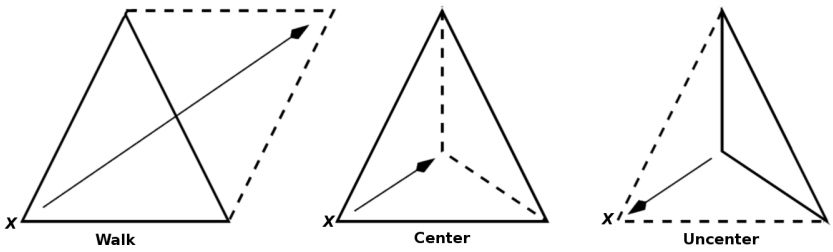


Figure 3.1: Two dimensional version of the WUC representation moves.

Definition 3.1 *The* **walk** *operation (w_k) is given by:*

$$v_k \leftarrow \frac{2}{d}\left(\sum_{i \neq k} v_i\right) - v_k$$

This command displaces the active vertex by twice the vector from it to the centroid of the opposite face of the simplex.

Definition 3.2 *The* **center** *operation (c_k) is given by:*

$$v_k \leftarrow \frac{1}{d+1}\left(\sum_{i=0}^{d} v_i\right)$$

This command moves the active vertex from its current position to the centroid of the simplex.

Definition 3.3 *The* **uncenter** *operation (u_k) is given by:*

$$v_k \leftarrow (d+1)v_k - \left(\sum_{i\neq k} v_i\right)$$

This command moves the active vertex to such a position that its original position is now the centroid of the resulting simplex.

Note that all three commands are invertible for any choice of vertex. Therefore, this set of commands, when applied to the set of all simplices, are bijections and so are members of the group of all bijections of the set of simplices in $\mathbb{R}^d$. As any command can be applied to any vertex, there are $d+1$ versions of each command in d dimensions. Each of these commands are considered as a letter in an alphabet. For a given set of moves, the representation consists of forming strings over this alphabet, and the commands are applied in the order in which they appear in a string. Evolution utilises standard crossover and mutation operators for strings. The mutation operators tested were m-point mutation operators, for a variety of fixed values for m. These operators randomly selected m distinct loci within the string,

and replaced the existing command with one generated uniformly at random. The choice of crossover operator was made based on the behaviour of the underlying group. The fact that these commands only commute with themselves or their inverse suggested that the representation would be highly epistatic (see Definition 2.5), and highly epistatic representations are typically best paired with single point crossover. Multiple crossover points would be highly disruptive to chains of commands which rely on the earlier commands being applied first.

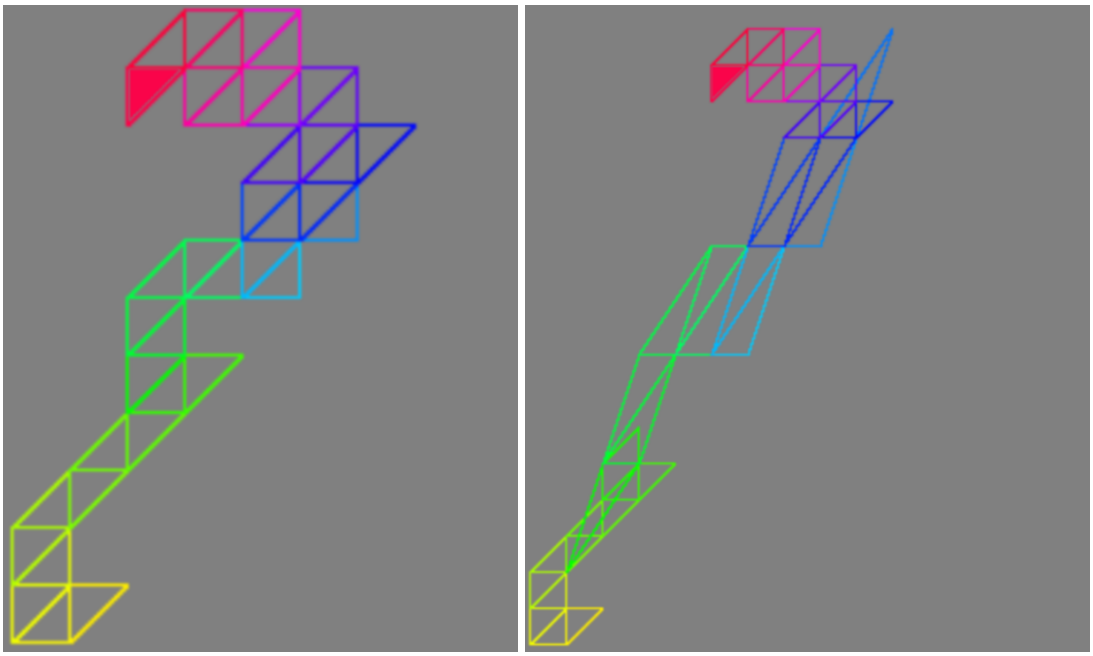


Figure 3.2: Examples of 100 random walking triangle moves. In the picture to the left, only walk moves are used. The picture to the right inserts one uncenter move after the 33rd step and one center after the 66th into the same random walk. The initial simplex is filled and step number is colour coded starting at red and progressing through the colour wheel toward yellow, via blue.

As mentioned before, there are $d + 1$ versions of each command in the alphabet in d dimensions, meaning the size of the alphabets in the WUC representation is $3(d + 1)$. The

representation was tested on various functions and with a variety of mutation operators. Figure 3.2 shows examples of decoded WUC representation genes. Since the uncenter command very quickly increases the size of a triangle, it would not be feasible to include a picture where uncenters are allowed freely, and so the examples are restricted to only using walk, as well as a lone uncenter and center.

Following is a list of several other properties of this particular set of WTR commands, followed by their proofs.

1. The walk command is self inverse when applied twice to the same vertex. The center and uncenter commands are inverses of one another when applied to the same vertex.

2. The group is non-commutative. This places a large importance on the order in which commands are applied.

3. When applied iteratively, the center command decreases the hypervolume of the simplex exponentially. The base of the exponential function of the center command in d dimensions is $d + 1$.

4. When applied iteratively, the uncenter command has similar exponential behaviour, but it increases the hypervolume of the simplex rather than decreasing it.

5. The walk command preserves the hypervolume of the simplex.

6. Given a simplex S, centering any vertex will produce a new, smaller subsimplex. The subsimplices produced from centering any two different vertices share only faces and

are equal in hypervolume. The union of all possible subsimplices produced from centering the vertices of S is equal to S. In other words, S can be partitioned into the possible subsimplices which result from applying the center command to the vertices of S.

7. Let $\epsilon > 0$. This particular set of commands can place the center of mass of a simplex within ϵ of almost any point $p \in \mathbb{R}^n$ in a finite number of applications. The number of steps required to accomplish this is proportional to the logarithm of the distance from the center of mass of the initial simplex to the point p.

Property #1 is by design, and fairly obvious upon inspection of the commands. Property #2 is also somewhat obvious but can be easily demonstrated with the visual counterexample in Figure 3.3.

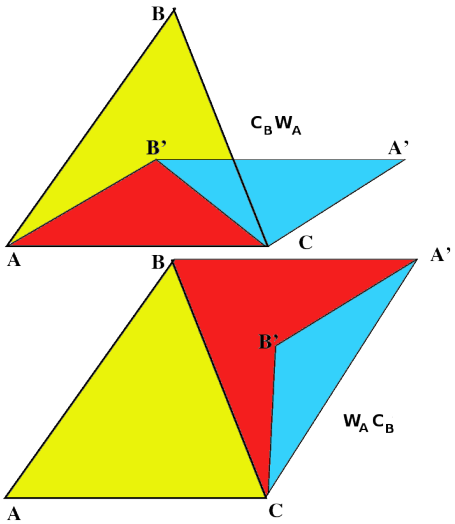


Figure 3.3: Shown above are the results of applying a center to vertex B and walk to vertex A in both possible orders. This demonstrates that the moves of the WTR are not commutative. The initial simplex is yellow, the final one is blue.

To understand how the WUC operations affect the hypervolume of a simplex, first consider the following elementary result:

Proposition 3.1 *Let $S = \{v_0, v_1, \ldots, v_d\}$ be a d-dimensional simplex (or more generally a d-dimensional cone), let B be the $(d-1)$-dimensional hypervolume of its base and let h be its height. Then its d-dimensional hypervolume V satisfies $V = \frac{1}{d}Bh$.*

Proof:

Let t be the vertical coefficent. The $(d-1)$-dimensional cross-section of S at any t between 0 and h has the same shape as the base and hence has hypervolume $\frac{B}{h^{d-1}}(h-t)^{d-1}$. Hence $V = \frac{B}{h^{d-1}} \int_0^h (h-t)^{d-1} dt = \frac{1}{d}Bh$. $\square$

From this it follows that the center operation decreases the hypervolume by a factor of $d+1$. Without loss of generality, suppose v_0 is the active vertex. Consider $\{v_1, \ldots, v_d\}$ to be the base; this base will not change when applying an operation to v_0. Then the height of the centroid is exactly $\frac{1}{d+1}$ the height of v_0. The center operation would move v_0 to the centroid, decreasing the height and therefore the hypervolume by factor of $d+1$. Since uncenter is the inverse of center, it would in turn increase the hypervolume by a factor of $d+1$. If walk is applied to v_0 instead, then the height of v_0 will remain constant by design, as it is a reflection through the base, thereby preserving the hypervolume as well. Thus, properties #3, #4, and #5 are all true.

Property #6 is also by design, but less obvious. The above result shows that the center

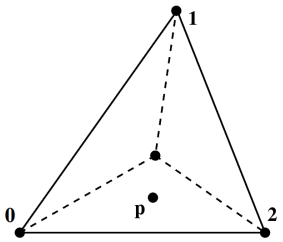
operation will decrease the hypervolume by a factor of $d+1$, regardless of which vertex is centered, showing that the subsimplices are equal in hypervolume. To see that they share only faces, consider the following: construct lines from each of the $d+1$ vertices to the centroid of the simplex. This set of lines includes all the new possible edges which a subsimplex could contain after a vertex is centered. If a vertex v_k is centered, the edges of the subsimplex will consist of the edges of the opposing face as well as d of the edges constructed above, omitting the line that was attached to v_k. Each of the $d+1$ faces has its own unique set of d constructed edges, of which there are $\binom{d+1}{d} = d+1$ possibilities. Hence the entire space inside the simplex is used. Since there are $d+1$ different subsimplices which each have a hypervolume of $\frac{1}{d+1}$ of the original simplex, the only way they can fill the entire space is if they have no intersection of positive volume, and only share faces.

Property #7 means that this representation is a *complete* one for searching $\mathbb{R}^n$. That is to say, the set of possible points which can be represented with a finite number of applied commands are *dense* within $\mathbb{R}^n$. However, for a given number of applied commands, the set of representable points is denser near the initial simplex, and the area that can be practically searched is limited by the length of the string of commands. This is a strong result, but requires some work to prove.

Lemma 3.1 WUC Crumpling Lemma

Neglecting the case where a point is on the boundary between two of the subsimplices that can be created by making a centring move, if p is a point in the interior of a simplex S then there exists a unique vertex k for which Sc_k contains p.

Proof:



This follows largely from property #6 listed above. Since p is not on a boundary, it is in one of the subsimplices created by centring. Examine the picture above - the point p is contained in exactly one of the subsimplices. This means that the unique vertex k that retains p in the subsimplex is the one that creates the subsimplex containing p. □

This lemma together with property #3 shows that once a point is in general position (not on a boundary between two subsimplices) in the interior of the simplex, this representation can iteratively shrink the simplex down to an arbitrarily small hypervolume while retaining p in the interior by selecting the correct vertex to center at each iteration. The relationship between the length of a simplex's edges and the maximum distance a point inside the simplex can be from the centroid is considered next.

Let $\{v_0, v_1, ..., v_d\}$ be the vertices of a simplex S and let $c = \frac{1}{d+1} \sum_{i=0}^{d} v_i$, the centroid of S, then let $\alpha(S) = \max_{i,j} ||v_i - v_j||$ be the maximal edge length of the simplex and $\beta(S) = \max_j ||c - v_j||$ be the maximal length from the centroid. Then we have the following:

Lemma 3.2 *Let S be a d-dimensional simplex, then $\frac{1}{2}\alpha(S) \leqslant \beta(S) \leqslant \frac{d}{d+1}\alpha(S)$*

Proof:

By the triangle inequality $||v_i - v_j|| \leqslant ||v_i - c|| + ||c - v_j|| \leqslant 2\beta(S)$. Taking the maximum over all i, j on the left hand side of this inequality and dividing both sides by two gives us $\frac{1}{2}\alpha(S) \leqslant \beta(S)$. Note that we also have $c - v_j = \frac{1}{d+1}\sum_{i=0}^{d}(v_i - v_j)$. Note that there are at most d nonzero terms in the sum on the right hand side and now using the triangle inequality we get $||c - v_j|| \leqslant \frac{1}{d+1}\sum_{i=0}^{d}||v_i - v_j|| \leqslant \frac{d}{d+1}\alpha(S)$. Taking the maximum over all j on the left hand side of this inequality gives us $\beta(S) \leqslant \frac{d}{d+1}\alpha(S)$. □

This lemma means that for any sequence of simplices $\{S_n\}_{n=1}^{\infty}$, we have $\beta(S_n) \to 0$ if and only if $\alpha(S_n) \to 0$. Thus, if the WUC representation can make the maximal edge length of the simplex arbitrarily small while keeping p in the interior, it can also make the distance between the centroid and p arbitrarily small. It has already been established that the hypervolume can be made arbitrarily small while retaining p in the interior. The problem case then arises in cases where the hypervolume can be made arbitrarily small with at least one of the edges not decreasing in length. For this to occur, at least two of the vertices (the two at the ends of the edge) must not have any center commands applied to them. This would only occur if p was on that edge or a face incident upon it, and therefore was on the boundary between two subsimplices in a previous iteration. This motivates the following result:

Lemma 3.3 *Let S be a d-dimensional simplex and let S' be the simplex formed by taking*

a string of center operations on S. If this string contains at least one instance each of d of the $d+1$ different center operations, then $\alpha(S') \leqslant \frac{d}{d+1}\alpha(S)$.

Proof: We know from the previous lemma that the distance between the centroid and the vertex of any simplex is at most $\frac{d}{d+1}\alpha(S)$. After performing these center operations, all of the new edges of the simplex must have length $\leqslant \frac{d}{d+1}\alpha(S)$. $\square$

So any string of commands, provided it keeps no more than one vertex fixed, will necessarily decrease the maximal edge length. More importantly, the decrease is exponential when applied iteratively, and

$$\lim_{n\to\infty}\left(\frac{d}{d+1}\right)^n = 0$$

so repeated iterations can make the edges arbitrarily small. There is one remaining topic related to the main result which must be discussed.

The topic of simply normal numbers is used in the proof of the following theorem, and so the relevant information is stated here. Normal and simply normal numbers as a whole are a very interesting topic of study with relevance to various areas of mathematics and computer science, but are beyond the scope of this thesis. Interested readers can profitably consult (Khoshnevisan (2006)) for an excellent consolidation of information and results in the area.

The idea of a simply normal number is fairly straightforward (one might even call it

"simple"): a number is said to be *simply normal* if its infinite sequence of digits contains each different digit with the same frequency. For a precise definition, suppose we are in integer base b. Let $x \in [0, 1)$ and consider the infinite sequence of digits in its base-b expansion. For $n \in \mathbb{N}$ and $k \in \{0, 1, \ldots, b-1\}$, define $S_{n,k}(x)$ as the number of times the digit k appears in the first n digits of the base-b expansion of x. Then we have the following:

Definition 3.4 *In a given integer base b, a number x is said to be* **simply normal** *if*

$$\lim_{n\to\infty} \frac{S_{n,k}(x)}{n} = \frac{1}{b} \quad \textit{for all } k \in \{0, 1, \ldots, b-1\}$$

There is a proper subset of simply normal numbers with a stronger condition on them referred to as *normal* numbers. Continuing from the previous paragraph, suppose that w is a finite string of base b digits with length $|w|$, and n and x are as above. Define $B_{n,w}(x)$ as the number of times the string w appears in the first n digits of the base-b expansion of x. Then we have the following:

Definition 3.5 *In a given integer base b, a number x is said to be* **normal** *if*

$$\lim_{n\to\infty} \frac{B_{n,w}(x)}{n} = \frac{1}{b^{|w|}}$$

for all finite strings w.

Beyond formal definitions, there is only one key property of normal and simply normal

numbers which is relevant here. It follows from two facts: any *normal* number is also *simply normal* by definition, and from a theorem by Borel in 1909, almost every number in $[0, 1]$ is normal (Borel (1909)). Together, these show that almost every number in $[0, 1]$ is simply normal, or the set of non-simply normal numbers in $[0, 1]$ has measure zero.

Notice that the property of being simply normal is not necessarily a numerical property, but rather a property of a string of symbols. The property can therefore be extended to symbolic representations as well. If we consider strings of center operations to be digits in a base-$(d + 1)$ expansion of a number in $[0, 1)$, then there is an analogous result to Borel's theorem related to the frequency of certain types of centering operators possible with the WUC representation. The main result can now be stated.

Theorem 3.1 *The set of points which can be represented by the WUC representation exists almost everywhere in $\mathbb{R}^n$. Let $\epsilon > 0$, and let D be the distance from the center of mass of the initial simplex to a point p in general position in $\mathbb{R}^n$, then the number of commands required to move the centroid of the simplex to within ϵ of p is proportional to $log(D)$.*

Proof:

Suppose we have a simplex S and a point p in general position. Define an order on the vertices of S. Using this order, run through the vertices cyclically uncentering each vertex one at a time until p is contained in the interior of the simplex. Note that since uncenter is increasing the hypervolume exponentially, the number of steps needed here is proportional to $log(D)$. From Lemma 3.1, there is a unique k such that applying c_k will keep p inside

the new subsimplex. These center operations can be applied iteratively, and by Lemma 3.3 can exponentially reduce the maximal side length of the simplex to be arbitrarily small. By Lemma 3.2, $\beta(S_n) \to 0$ if and only if $\alpha(S_n) \to 0$, and so the distance from the centroid of S to p can also be made arbitrarily small.

All that remains to prove is that points on the boundary between two subsimplices can be neglected. Consider the following numerical representation of strings of center operations in the context of Borel's result. Suppose we have a string of center operations on a set of vertices $\{v_0, v_1, \ldots, v_d\}$, then this can be represented as the digits of the base-$(d+1)$ expansion of a number in $[0, 1)$ where each digit corresponds to the vertex that was centered. For example, if vertex 4 is centered, then vertex 0 is centered twice, then vertex 2 is centered, this string would be represented as 0.4002.

Suppose we have point p on the boundary between two subsimplices of S. Then we can choose one of the two centerings which will gives us one of these two subsimplices, and in either case our new simplex will have p located either on a face or on an edge. Consider the case of an edge. If either of the vertices incident on that edge are centered, p will no longer be in the simplex. Thus, to reduce the hypervolume to be arbitrarily small while retaining p in the simplex, only vertices non-incident on that edge can be centered. Using the numerical representation explained above, the number representing this string of center commands will lack the two digits corresponding to the two vertices incident on the edge. If p was instead located on a face, this would be even worse, as the number representing the string of center commands would simply be the same digit repeated.

In either of these cases, if we consider an infinite string of center operations applied to retain p in the simplex while it is located on an edge or a face, the number representing this string will lack certain digits in base $d + 1$. This means that these numbers are not simply normal numbers, and are therefore a set of measure zero as described above. As these are the only problem cases, it follows that the set of points which can be represented by the WUC representation exists almost everywhere. □

Note that while this proof is constructive, it uses a rather inefficient method to place p in the interior of the simplex. Indeed, it would be far more efficient to only uncenter the smallest number of vertices necessary in order to have p contained inside the simplex. Showing this construction is possible becomes increasingly difficult as dimension increases, and so the broader and less efficient method was used to more easily generalize to any finite dimension. In practice, the convergence will be much faster than implied in Lemma 3.3, but this is enough to give logarithmic convergence in the simply normal case.

3.2.1 Algorithms

Experiments for this study were conducted in two phases. The initial set of experiments is designed to exemplify the exploratory power of the WUC representation. Two test functions are used which are known to require a high degree of exploration for an algorithm to perform well. The second set of experiments is meant to test the WUC representation on a set of five standard real optimization test problems. In both sets of experiments, a

simple evolutionary real parameter optimizer was compared with the WUC representation, however the exact design of the algorithms differ slightly between the sets.

WTRs require an initial simplex to which commands can be applied. In this study, the initial simplices had vertices consisting of the standard basis for $\mathbb{R}^n$ along with the origin. For example, a two-dimensional problem would have an initial simplex of $S = \{(0,0),(0,1),(1,0)\}$. The fact that this representation is a complete one for searching $\mathbb{R}^n$ lessens the importance of choosing a good location for the initial simplex, as evolution is capable of efficiently compensating for a poor choice of initial simplex. This study tested the performance of the WUC representation with a single, standard starting simplex. Experiments which involve changing the initial simplex will be left for future work. The decision to not test this particular possible feature of the algorithm was made upon observation of the robustness of performance to the position of target optima.

The baseline evolutionary algorithm which was used for comparison to the WUC representation in the initial set of experiments is a standard real function optimiser with the independent variables of the objective function being stored in an array of real values. Representations of this type are known to have a non-trivial degree of epistasis (see Definition 2.5), which is dependent on the objective function. I therefore used two point crossover and m-point mutation, where a value drawn from a $N(0, 2.0)$ normally distributed random variable was added to each of the coordinates which were mutated. This algorithm used size seven single tournament selection for selection and replacement, with varying population sizes. In this model of evolution, seven members of the population are chosen and

the two most fit are copied over the two least fit. The choice of size seven represents a low-intermediate level of selection pressure, which is a good generic choice. The various population sizes used are given in Section 3.3.

A simple evolutionary algorithm was used for the initial test of the WUC representation. This evolutionary algorithm is nearly identical to the baseline algorithm it is being compared to, with the sole changes being that it used one point crossover rather than two due to the highly epistatic nature of this particular instance of WTR, and the mutation operator replaced the walking triangle commands at the loci chosen for mutation rather than adding a real value.

The real optimizer used in the subsequent experiments was similar to the one used in initial experiments, and again used two point crossover. However, it used a mutation operator that adds a Gaussian random variable with $\sigma = 0.3$ to from 1-3 loci in the gene, with the number of loci modified selected uniformly at random. The optimizer is initialized with points chosen uniformly at random in the range $-10 \leq x_i \leq 10$. Selection and replacement are accomplished with size four tournament selection. This model of evolution picks four population members and copies the two best over the two worst, after which the copies are subjected to mutation and crossover.

For the subsequent experiments, the WUC representation's evolutionary algorithm is nearly identical to that of the initial experiments, except that it also uses size four tournament selection. It operates on a population of 100 strings of 12 walking triangle commands. Again, because the representation is highly epistatic, the algorithm uses one point

crossover and mutation replaces one of the walking triangle commands, selected uniformly at random.

3.3 Experimental Design

The first set of experiments performed parameter setting studies for gene length, mutation rate, and population size for an evolutionary algorithm using the WUC representation. The algorithm used size seven single tournament selection on a population of WUC representation genes. The algorithm performed 100,000 such tournaments, called *mating events*. Every 1,000 mating events, the current mean, standard deviation, and maximum fitness of the population is reported and recorded.

The first test function that was used for these experiments is called the *eight hill* function, and is designed to test the exploratory power of the representation. This function is given by:

$$f(x) = \sum_{k=0}^{7} \frac{(k+1)}{1 + \sum_{m=1}^{d} (x - 5 - 20 * m)^2} \tag{3.1}$$

Where x is a point in $\mathbb{R}^d$. This function has eight local optima in any finite dimension. The optima occur at points with all coordinates equal, with coordinate values of 5, 25, 45, 65, 85, 105, 125, and 145. The optima increase in value as they move away from the origin, with values slightly higher than 1, 2, 3, 4, 5, 6, 7, and 8, respectively, for the above-mentioned points. A transect of the eight hill function is shown in Figure 3.4, a plot of the

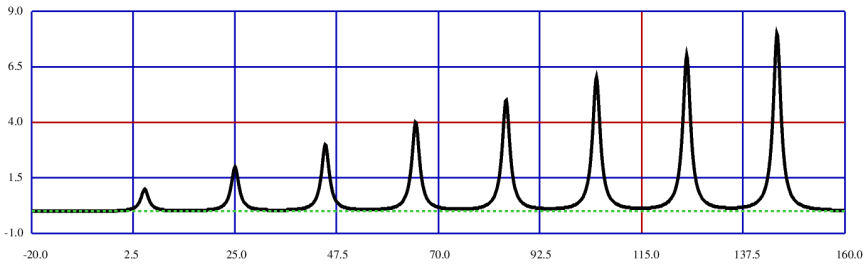


Figure 3.4: Shown is a cross section of the eight hills function along the line of points with all coordinate values equal. The transect of $\mathbb{R}^n$ includes all the optima for any n.

two-dimensional surface version of the function is shown in Figure 3.5.

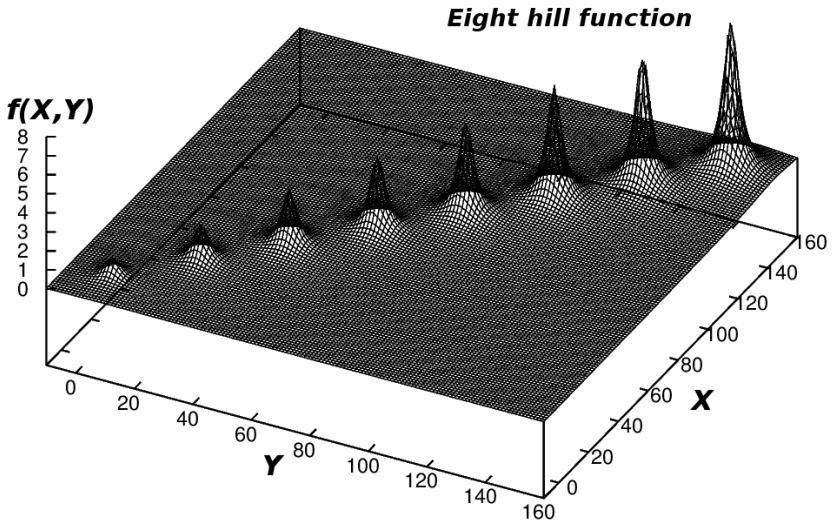


Figure 3.5: A plot of the eight hills function for $d = 2$ input dimensions.

A full factorial design was used for the parameter setting study. The population size was varied between 10, 100, and 1000, with gene lengths of 20, 40, and 60, and a maximum number of mutations of 1, 3, and 5. The actual number of mutations performed on the strings of WUC commands in each instance was selected uniformly at random between 1 and the maximum. This parameter study was completed in both $d = 2$ and $d = 3$

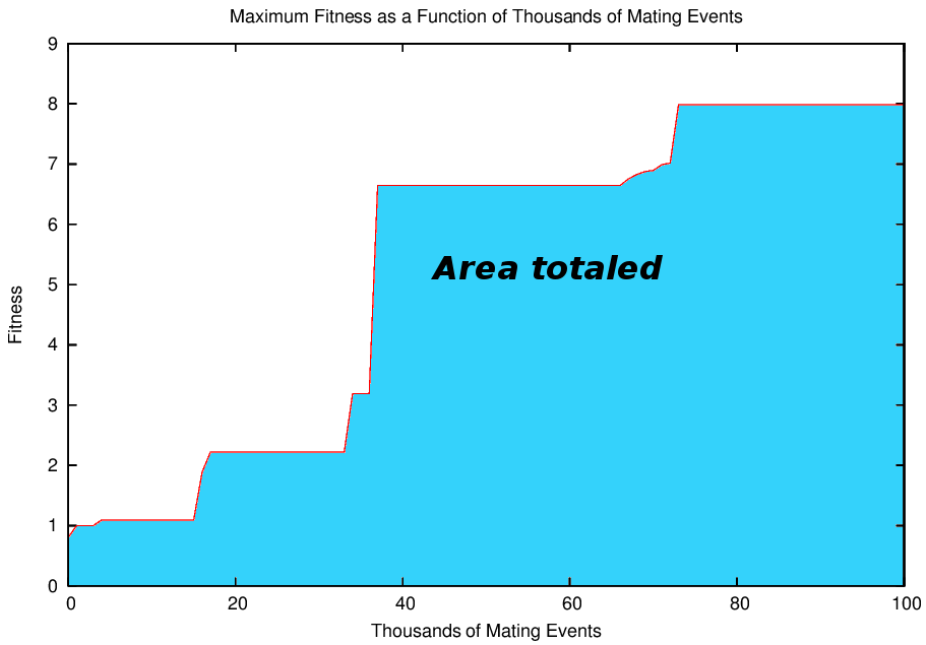


Figure 3.6: Shown is the graph of maximum fitness over evolutionary time for a run of the WUC algorithm on the eight hill function. The stair step progress represents subsequent discovery of better and better solutions by the evolving population.

dimensions.

The measure used to assess the quality of optimization over the course of evolution is called the *total maximum fitness* (TMF) statistic. This measure is a proportion of the area under the maximum fitness over time graphs, and an example is shown in Figure 3.6. The proportionality is obtained by dividing the area by the number of samples and maximum fitness. The effect of this is to normalize the number to the range $0 \leq v \leq 1$. The idea of measuring things this way is that if the algorithm is able to very quickly find the global optimum, then its TMF will be close to 1. If the algorithm is unable to produce good results over a large stretch of evolution, or potentially never finds a good solution, then its TMF will be close to zero.

3.3.1 Comparison With a Standard EA

The design of the WUC representation suggested that it would possess great exploratory power. Its ability to increase the simplex's hypervolume exponentially allows it to travel quickly and explore new areas of the search space.

In this study, the representation modifies a point that starts in the unit hypercube of the positive orthant. After the parameter study, a second set of experiments was conducted to compare the performance against a standard baseline evolutionary algorithm in $d = 2$ dimensions. The WUC representation used the parameters: population size 100, gene length 40, with a maximum number of mutations of 5. The values were chosen based on the parameter studies in $d = 2$ and $d = 3$ dimensions. The baseline evolutionary algorithm was run four times, using different initialization domains: squares with corners at (0,0) and (q, q) for $q \in \{1, 10, 100, 1000\}$. A population of 120 points was initialized uniformly at random inside these initialization domains. The first domain is simply a square which contains the starting simplex of the WUC representation. The second domain is 100 times larger, but only contains one local optimum. The third domain includes five of the eight local optima, and the final domain contains all eight local optima (including the global optimum). Due to the construction of the eight hill function, final fitness values near n strongly suggest that the algorithm has found the nth optimum out from the origin.

The second function used in the first set of experiments was first defined in (Ashlock

et al. (2013)) and is called the open-ended trendwave. The function is given by:

$$f(x) = \sum_{i=0}^{d-1} \left(\frac{x_i}{20} + cos(x_i) \right) \tag{3.2}$$

Where x is again a point in $\mathbb{R}^d$. This function has an infinite number of local optima of indefinitely increasing values. It is called open-ended because the optima continue to improve in quality the further you move outwards, and there is no global optimum. The experiment compared the baseline EA, using the same initialization domains as mentioned above, with the WUC algorithm. It used population size 100 and a maximum of five mutations. Four runs of the WUC algorithm were conducted with gene lengths of 20, 40, 60, and 80, respectively. The comparison was made by looking at the final fitness values for 30 sets of runs for each algorithm in $d = 2$ dimensions. The sets of runs for the open-ended trendwave are given in Table 3.1.

Table 3.1: The names and parameters for the four baseline and four WUC algorithms used for comparison of performance on the fitness function given in Equation 3.2.

Experiment Name	Initialization	Gene Length
E1	$0 \leq v \leq 1$	n/a
E2	$0 \leq v \leq 10$	n/a
E3	$0 \leq v \leq 100$	n/a
E4	$0 \leq v \leq 1000$	n/a
T1	Standard simplex	20
T2	Standard simplex	40
T3	Standard simplex	60
T4	Standard simplex	80

3.3.2 Subsequent Experiments

After the initial set of experiments were conducted to demonstrate the exploratory strength of the WUC representation, a set of experiments were conducted on bounded real optimization problems to test overall performance against a baseline evolutionary algorithm. The algorithms were similar to those used in the initial experiments; the minor differences are detailed in Section 3.2.1. The real optimizer operates on a population of 100 points storing a number of real parameters equal to the dimension of the problem. This value was chosen to match the population size of 100 used in the walking triangle representation algorithm.

The optimizer is initialized with points chosen uniformly at random in the range $-10 \leq x_i \leq 10$. Selection and replacement are accomplished with size four tournament selection. The algorithm is run for 1,000,000 tournament selection events and the best fitness is reported at the end of each run for analysis. The walking triangle representation also uses size four tournament selection for 1,000,000 selection events. It operates on a population of 100 strings of 12 walking triangle commands. Each algorithm was run for 30 independent replicates on each of the test functions in 4, 8, and 12 dimensions.

Function Descriptions

The WUC representation was compared with the evolutionary real parameter optimizer on five functions: the Ackley, Griewank, hidden hill, Lorentz, and Rastrigin functions. The Ackley, Griewank, and Rastrigin functions were negated so that all five functions were

maximization problems. The maximum of the Ackley, Lorentz, and hidden hill functions is one, the maximum of the Griewank function is zero, and the Rastrigin function has a maximum of ten.

The first function is the Ackley function, given by:

$$f(x_1, x_2, \ldots, x_d) = -20e^{\left(-0.2\sqrt{\frac{1}{d}\sum_{i=1}^{d} x_i^2}\right)} - e^{\left(\frac{1}{d}\sum_{i=1}^{d}\cos(2\pi \cdot x_i)\right)} + 20 + e^1 \tag{3.3}$$

The main structure of this function is a well with its bottom at the origin. The cosine function causes there to be large numbers of local optima along the sides of the well. The two dimensional Ackley function can be seen in Figure 3.7. It was negated in this study to become a maximization problem.

The second function is the Griewank function, given by:

$$f(x_1, x_2, \ldots, x_d) = \sum_{i=1}^{d} \frac{x_i^2}{4000} - \prod_{i=1}^{d} \cos\left(\frac{x_i}{\sqrt{i}}\right) + 1 \tag{3.4}$$

This function forms a parabolic bowl which, similar to the Ackley function, has a large number of local optima formed due to cosine waves. In this case, however, the $\sqrt{i}$ term causes the periods to be distinct. As the dimension increases, the optimization behaviour of the Griewank function begins to approximate the behaviour of a simple parabolic bowl. The two dimensional Griewank function can be seen in Figure 3.8. It was also negated in

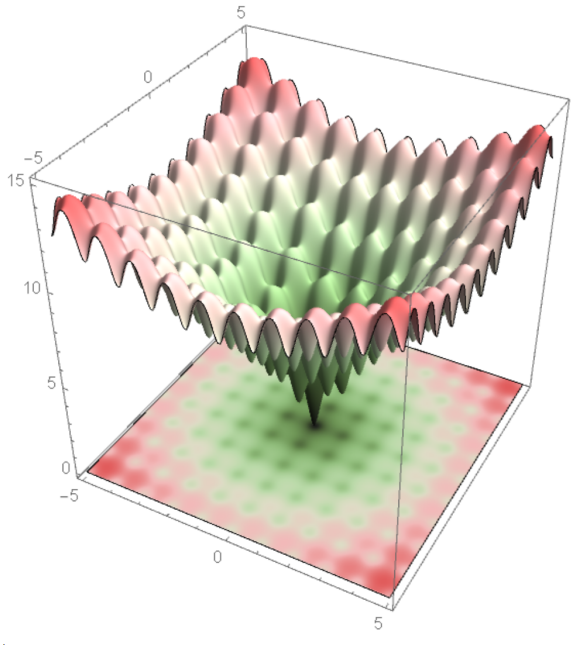


Figure 3.7: The two dimensional version of the Ackley function, picture from the Wikimedia Commons.

this study to become a maximization problem.

The third function is the hidden hill function. Let $r(x_1, x_2, \ldots, x_d) = \sqrt{\sum_{i=1}^{d} x_i^2}$, then the hidden hill function is given by:

$$f(r) = \begin{cases} \frac{2}{r^2+1} - 1 & |r| \leq 1 \\ \\ \frac{r^2}{r^2+1} - 0.5 & |r| > 1 \end{cases} \tag{3.5}$$

This function has a unique global optimum at the origin with a height of one, atop a unimodal hill contained in the unit sphere in d dimensions. Outside of that sphere, the gradient points directly away from the global optimum, and the function increases asymptotically towards a value of one half as it moves further from the unit sphere. This function's de-

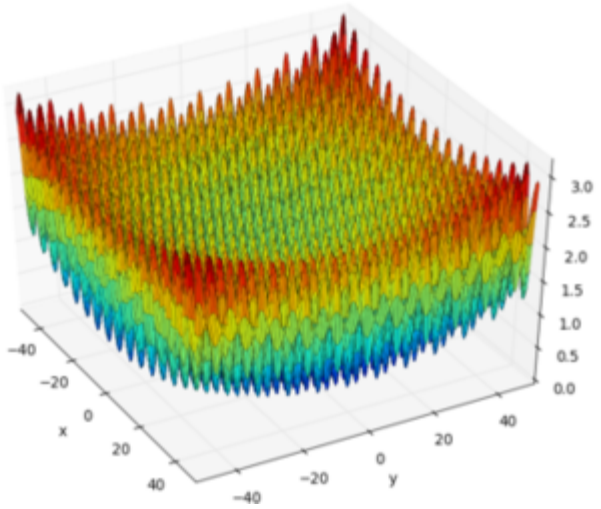


Figure 3.8: The two dimensional version of the Griewank function.

ceptiveness increases with dimension, however the function becomes a simple hill climb if the domain of initialization is too small. Figure 3.9 shows the hidden hill function in two dimensions.

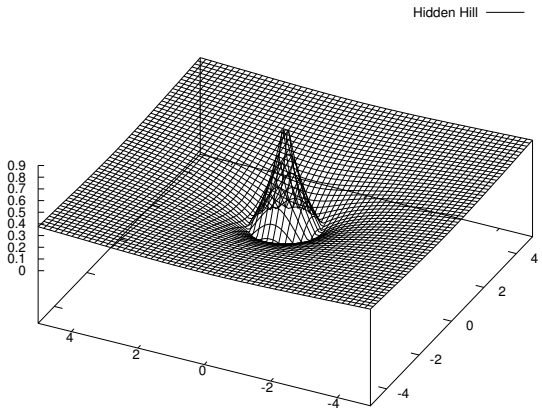


Figure 3.9: The hidden hill function in two dimensions.

The fourth function is the Lorentz Hill function, given by:

$$f(x_1, x_2, \ldots x_d) = \frac{1}{1 + \sum_{i=1}^{d} x_i^2} \tag{3.6}$$

This is a very simple function typically used as a default starting point. The entire graph is a unimodal hill with its optimum at the origin. The two dimensional version is shown in Figure 3.10.

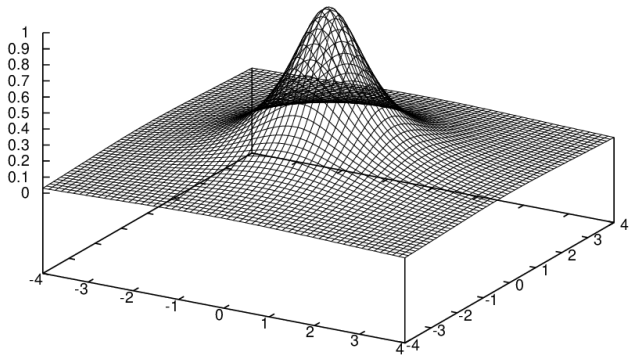


Figure 3.10: The two dimensional version of the Lorentz hill function.

The fifth and final function is the Rastrigin function, given by:

$$f(x_1, x_2, \ldots, x_d) = d + \sum_{i=1}^{d} \Big(x_i^2 - \cos(2\pi x_i) \Big) \tag{3.7}$$

Similar to the Griewank function, this function is also a quadratic basin with cosine waves to create a large number of local optima. Figure 3.11 shows the Rastrigin function in two dimensions. This function was negated in this study to become a maximization problem.

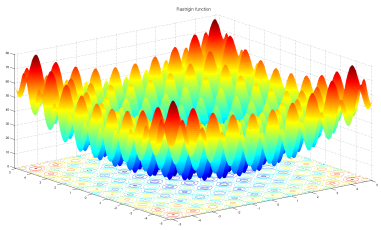


Figure 3.11: The two dimensional version of the Rastrigin function.

3.4 Results and Discussion

The results of the parameter study are shown in Figure 3.12. Between $d = 2$ and $d = 3$ dimensions, the parameter set of $P = 100$, $L = 40$, and $M = 5$ produced a fairly insignificant difference, and exhibited relatively small variability, and so this parameter set was chosen for subsequent experiments. The difference in results were considered statistically significant when the inflections on the box plots in Figure 3.12 do not overlap. The following patterns were noticed in the displayed results:

- A population size of 1000 was the worst choice and a population size of 100 was the best choice. Typically, one of the extreme values used will be superior, which made this an unusual problem.

- When using population sizes of either 10 or 100, a higher mutation rate was always better. When using a population size of 1000, a higher mutation rate was always worse, unless the gene length used was 60, in which case the mutation rate was shown to make negligible difference.

- Using a gene length of either 40 or 60 was typically superior to using a gene length

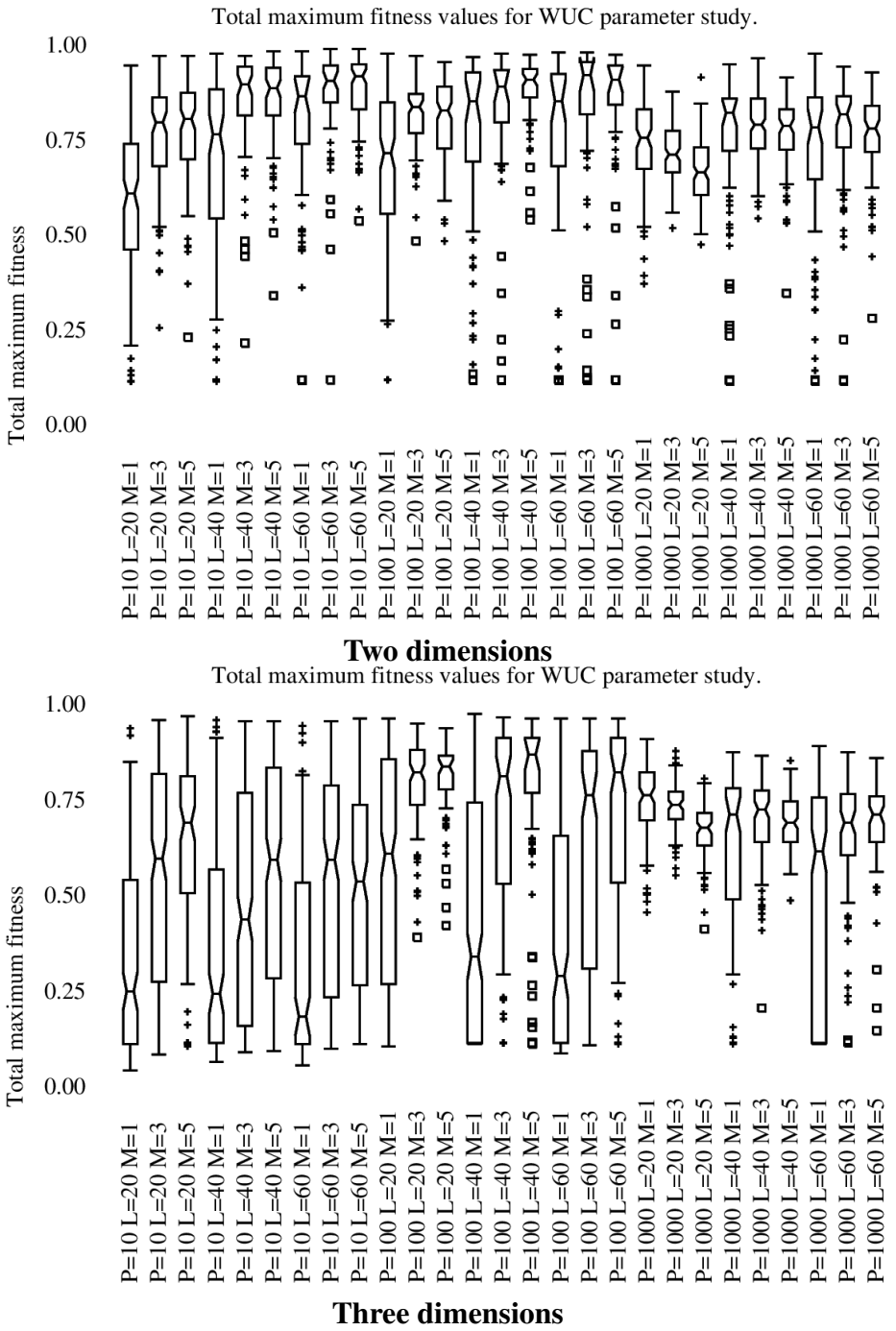


Figure 3.12: Parameter study results for the WUC representation on the eight hill function displaying the total maximum fitness statistic. In the labels **P** is population size, **L** is length, and **M** is the maximum number of mutations. The data are displayed as wasp plots with the upper panel being $D = 2$ dimensional and the lower $D = 3$.

of 20. There was very little difference between a gene length of 40 and 60, which suggests that 40 is sufficient and that extending the length to 60 simply gives the algorithm the opportunity for very minor changes in overall fitness.

The parameter study suggested a non-linear interaction between the parameters. When looking at WUC genes of a fixed length, the set of points that they can encode is a fractal dust. This set of points has the highest density near the initial simplex, and the furthest point from the initial simplex which can be encoded grows exponentially with the length of the gene. Furthermore, the fractal is self similar in the sense that the pattern made by all the suffixes of a given length is the same at each point encoded by the corresponding prefixes, except for scale. Considering the possible encoded points from this viewpoint suggests that, given more time to evolve, longer genes should always achieve superior results. When using shorter gene lengths, the search space encoded becomes much smaller. This trade-off is largely what explains the pattern of the results found in the two parameter studies.

Table 3.2 shows a summary of the optima located for the baseline evolutionary algorithm and the WUC representation. An optimum is considered located if the algorithm is within a Euclidean distance of 0.1 of the optimum. These results show that, despite using Gaussian mutation which is known to be highly exploratory (Ashlock (2006)), the baseline evolutionary algorithm will find optima inside its domain of initialization and, as the experiment with the largest initialization domain demonstrates, often not the best optimum in said domain. Contrarily, the WUC algorithm, which uses a starting simplex with no

Table 3.2: Shown is the pattern of optima located by different instances of the evolutionary algorithm and the walking triangle representation. In each experiment 30 instances of the algorithm were run.

Initialization	Optima located
Baseline EA	
$0 \leq x, y \leq 1$	1st x 30
$0 \leq x, y \leq 10$	1st x 30
$0 \leq x, y \leq 100$	5th x 30
$0 \leq x, y \leq 1000$	7th x 11, 8th x 19
WTR Algorithm	
Standard simplex	7th x 1, 8th x 29

useful information in its interior, was able to find the best optimum 29 of 30 times, with the second-to-best optimum being found once.

These results are a perfect demonstration of how robust the WUC representation is to its initialization compared to the baseline evolutionary algorithm. This is not often an issue for most test problems, as the problem will specify a reasonable initialization domain. However, with applied problems, there is often the potential to not have a well-defined sense of a good initialization domain for an evolutionary algorithm. This is particularly true for fitness functions which have the form of a simulation and lack physical constraints on the parameter values.

The results of the comparison of the baseline evolutionary algorithm with four different initialization domains and the WUC representation with four different gene lengths on the open-ended trendwave function are shown in Figure 3.13. The baseline evolutionary algo-

rithm finds an optimum in or near its initialization domain, again despite the exploratory nature of Gaussian mutation. The WUC representation detects the underlying trend of the fitness function and so produces maximum fitness values proportional to an exponential function of the gene length used. This experiment demonstrates the powerful exploratory nature of the WUC representation when compared to a standard evolutionary algorithm. The most-fit result from the first run of the gene length 20 WUC experiment is:

U2U2U2U2U2U2U2U1U1U1U1U1U1U0U0U0U0U0

All twenty loci are uncentering moves: seven uncenterings of vertex 2, followed by seven uncenterings of vertex 1, followed by six uncenterings of vertex 0. The initial population has loci that have equal chance of being a center, uncenter, or walk command.

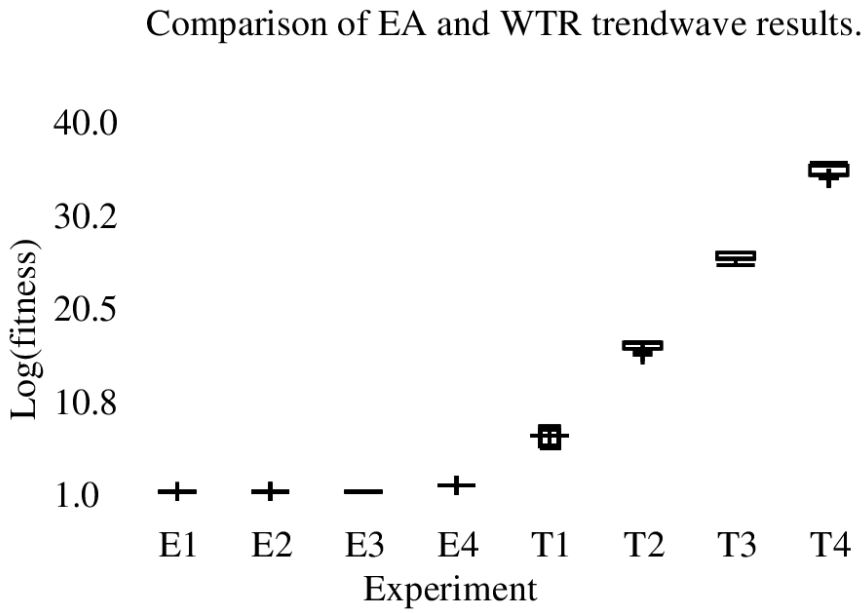


Figure 3.13: Shown are box plots for the distribution of the base-10 log of final fitnesses for the experiments listed in Table 3.1

The results for the five functions tested in the second set of experiments are given in Figure 3.14. All five functions have been considered as maximization problems by inverting the Ackley, Griewank, and Rastrigin functions. The results from the simple optimizer are shown in red, as distributions of fitness values, while the WUC representation's results are shown in blue. In no case did the simple evolutionary optimizer outperform the WUC representation to a significant degree, while the WUC based optimizer outperformed the simple optimizer ten times in fifteen comparisons.

3.5 Conclusions and Next Steps

This study gave a number of theorectical properties of the WUC representation, including the ability to perform open-ended search within a broad domain, controlled by gene length. The WUC representation, as with all WTRs, transforms real optimization into a discrete search problem in a novel manner. Given the principles revealed by the no free lunch theorem (Wolpert and Macready (1997)), the WUC representation will have specific domains of applicability where it is superior, compared to other problems on which it may struggle. These experiments demonstrated that problems where exploratory behaviour is highly beneficial, and open-ended searches where the obvious search boundaries are not clear are types of problems on which the WUC representation will excel.

This study confirms the results in (Ashlock et al. (2013)) for open-ended search for the WUC representation, different from the representation used in the earlier study. A new

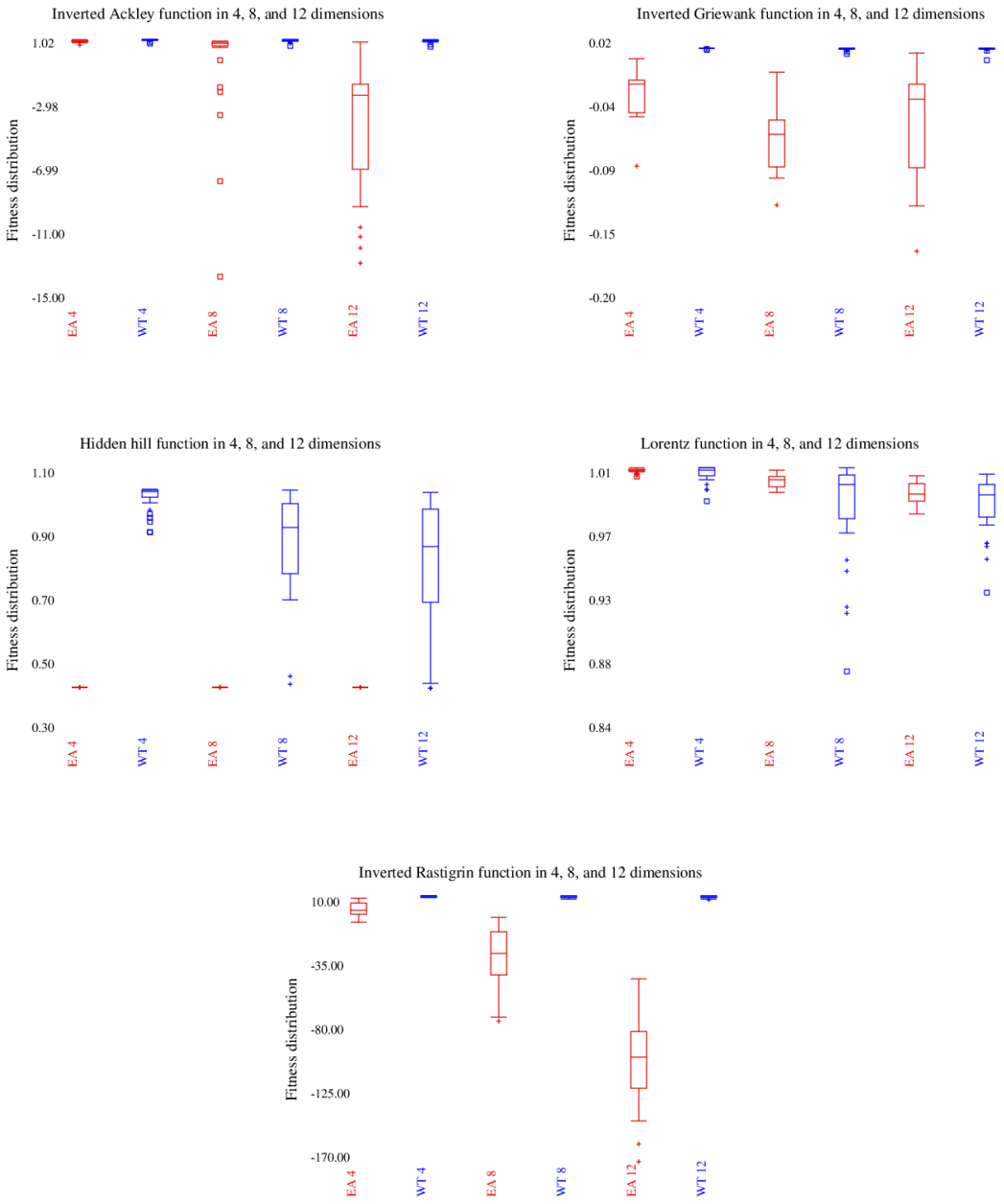


Figure 3.14: Results of the experiments of a standard evolutionary real optimizer (red) and the WUC represetation (blue) on the five test functions described in Section 3.3.2.

test function, the eight hill function given in Equation 3.1, was introduced. The function is specifically designed to document the ability of an algorithm to explore. It is noted that the number and spacing of the hills in said function can be modified to create a collection of test functions with varying difficulty to exploratory optimization. The results on this function, as well as those on open-ended optimization, in both this and the earlier study, also demonstrated the ability of the WUC representation to find optima well outside the region bounded by the initial simplex. This diminishes the importance of the selection of the initial simplex.

3.5.1 Domains of Initialization

The experiments in this study compared evolutionary algorithms with very different initialization domains with a WTR. Since the initial simplex used by the WUC representation is a slice of the unit hypercube closest to the origin, it might seem that this is comparable to initialization of the baseline evolutionary algorithm in the unit hypercube, but this is clearly not the case. Unless the gene produces only centering moves, or a combination of centering moves with possible consecutive centering/uncentering inverses, the points which will be encoded by the WUC representation will necessarily be outside the initial simplex. All points generated in initialization of the baseline evolutionary algorithm would be located within the hypercube, while the WUC representation genes are free to rapidly wander outside of the initial simplex.

The approximate equivalent to the initialization domain of the baseline evolutionary

algorithm is the gene length of the WUC representation. The points in $\mathbb{R}^n$ representable by the WUC representation are a discrete collection of points. As gene length is increased, these points become denser near points encoded by shorter genes, and also begin to include points which are further from the initial simplex. The practical effect of this is that the domain of search is self organised to a much greater degree in algorithms that use the WUC representation. Suppose, for example, that a small region of space contains the best optima. With the baseline evolutionary algorithm, the population would converge to this region, escaping from it at a rate dependent on the exploratory nature of the mutation operators. For the WUC representation, a prefix of commands could become dominant in the population, focusing search in the subdomain fixed by said prefix. If this self-organised localization of search is desirable for a problem, then the use of an elitist algorithm can ensure that it occurs.

Another property of the WUC representation is that it is naturally *multi-scale*. The center and uncenter commands permit a gene to select the scale on which search takes place. The contrasting property in a standard evolutionary algorithm is the selection of the variability or exploratory behaviour of the mutation operators. A WUC representation-based algorithm is both self-organised and dynamically selects its scale. The mutation rate experimentally had very little impact on performance, and as mentioned in Section 3.1 the WUC representation removes the need to choose a probability distribution. In essence, it is far more difficult to get the variability of the mutation operators wrong when using the WUC representation.

3.5.2 Restarting-Recentering Algorithms

Any generative representation with a starting configuration can perform incremental evolution by periodically replacing the starting configuration with the best result found so far. Algorithms using this technique are called *restarting-recentering* evolutionary algorithms. They have been applied to network evolution problems (Ashlock and Jafargholi (2007); Ashlock et al. (2011)) which use representations that modify a user-specified network. WTR-based algorithms also modify a starting configuration, the initial simplex, and so are a natural choice for the use of restarting-recentering techniques. Such an algorithm would periodically replace the starting simplex with the highest fitness simplex located so far and then also reinitialize the population of modification commands.

Originally, the reason for adopting restarting-recentering algorithms was that the space of networks being searched was so large that incremental evolution was required. Used with the WUC representation, restarting-recentering techniques could be used to ameliorate the fact that points far from the initial simplex have fewer points in the neighbourhood. For example, a relatively good, yet far away point with few representable neighbours, after a number of recenterings, would be surrounded by the densest region of representable points.

Chapter 4

Producing Diverse Sets of High Fitness Solutions Using Dictionary Exclusion

4.1 Introduction

A very common use of evolutionary computation is for real parameter estimation. This includes traditional evolutionary algorithms (Bäck and Schwefel (1993); Deb et al. (2002)), differential evolution (Addis et al. (2008)), ant colony optimization (Mohan and Baskaran (2012)), and other algorithms (Karaboga et al. (2014)), such as artificial bee colony algorithms. The type of algorithm which is best for a particular problem has been found to depend strongly on the character of the problem being solved (Bryden et al. (2006)). This study focuses on the problem of optimization when the fitness landscape has a large number of local optima and for which the fitness measure is suggestive, not definitive.

A *generalized Julia set* is a type of fractal specified by N complex parameters: $z_1 = x_1 + i \cdot y_1, z_2 = x_2 + i \cdot y_2, \ldots, z_N = x_N + i \cdot y_N$, so that $2N$ real parameters are required to specify one of the fractals. The exact definition of the fractal was given in Section 2.4. The objective function used to estimate fitness of a set of parameters is based on an entropic measure of sampled points in the fractal. The measure uses Shannon entropy (Shannon (1948)). This fitness measure encourages the search for a fractal with some complexity; this often yields an interesting-looking fractal but does not directly measure the quality "interesting". The entropic measure is a plausible surrogate for "interest" and so is suggestive, not definitive, in guiding the evolutionary search of the space of fractals.

The significant result in this study is a method of using previously-located high fit solutions to encourage evolution to locate a diverse set of high fit solutions that is lower-cost and simpler than niching (Shir (2012)). Niching adjusts the fitness of an organism in an evolving population by reducing the fitness if many other organisms are nearby. This encourages the population not to pile up in a single high fit solution, encouraging exploration. Problems with this approach include the cost of determining the distance between organisms, and the potential for preventing the location of some good solutions with poorly adjusted penalty functions.

Another technique that has been developed to profitably store previously-located solutions is called tabu search (Glover (1986)). Tabu search is a technique which is added to an iterative local search method to improve its performance and prevent it from "getting stuck". During the search process, tabu search employs a tabu list which, in its simplest

form, stores the solutions found in recent previous iterations to prevent the search from revisiting those solutions. This helps to guide the search out of low fitness regions, or any flat region of the search space where all nearby solutions are equally fit, and it has been implemented effectively in the decades since its inception. Tabu search uses previous solution data to guide it during its current search process, whereas the dictionary exclusion technique described in this chapter saves data of the best fit solution found during a search to be used on all subsequent searches. Tabu search also excludes specific solutions to prevent them from being found again, whereas dictionary exclusion can exclude anything from a specific solution to entire regions of various sizes in the search space, depending on the length of the stored prefixes (see Section 4.2.3).

In this study the evolutionary algorithm is run many times and a dictionary of previously-located best fit solutions is used to encourage the location of new high fitness solutions, by excluding those already located. This approach is most valuable exactly when there are many high fitness solutions and the fitness function is suggestive; the goal is to locate many different high fitness solutions that the researcher can then choose amongst based on other criteria. For the fractals in this study, that secondary selection method is visual inspection.

4.2 Background

This study attempts to locate parameter sets for a type of fractal known as *generalized Julia sets* (Ashlock and Jamieson (2007)). A rendering of a fractal is an assignment of

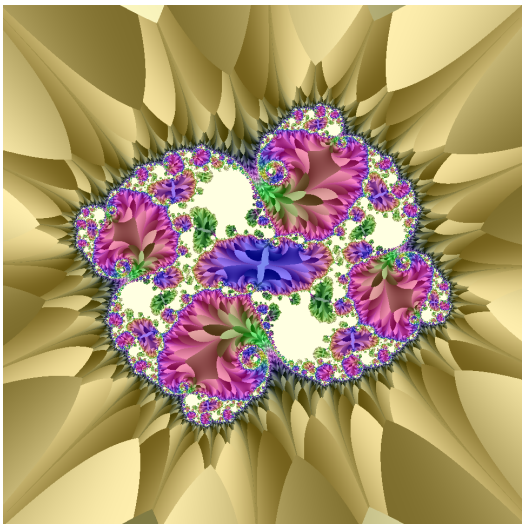


Figure 4.1: An rendering of an evolved generalized Julia set with three complex parameters.

colours to the pixels both inside and outside of the fractal. In this case, the points that are part of the fractal are the manilla coloured regions. An example of a rendering of one of these fractals appears in Figure 4.1.

4.2.1 The Objective Function

An equally spaced 11×11 grid of points is placed, aligned with a square with corners $-1.6 - 1.6i$ and $1.6 + 1.6i$, in the complex plane. The iteration number of each of these 121 points is computed. The resulting iteration numbers are then binned into sixteen bins numbered $0, \ldots 15$ using the formula $bin = \lfloor \frac{I*16}{121} \rfloor$, where I is the iteration number. These bins are normalized by dividing by 121 to form an empirical probability distribution $p_0, p_1, \ldots, p_{15}$ on the bins. Please note that this denominator is achieved by adding one to the iteration limit, so that points with an iteration number of 120 are correctly placed in bin 15. The fact that there are also 121 sample points is coincidental and unrelated. I

humbly admit that, in hindsight, a slightly different iteration limit would have been useful in avoiding any potential confusion. The objective function is:

$$E = -\sum_{k=0}^{15} p_k \cdot \log_2(p_k) \tag{4.1}$$

This is the *Shannon entropy* (Shannon (1948)) of the empirical distribution. Shannon entropy is maximized when a probability distribution is uniform and so this objective function rewards splitting the 121 sample points as evenly as possible among the bins.

A fractal that has a wide distribution of iteration numbers typically has both a complex shape and, when the points in the view window are rendered with colors chosen by iteration number, yield a complex and often interesting picture. The rendering of a fractal shown in Figure 4.1 is created by using the iteration numbers of the point corresponding to each pixel to index a periodic palette, save for iteration numbers of 120 which are assigned to a background colour, in this case manilla.

This objective function is drawn from an earlier study (Ashlock and Jamieson (2007)); it uses the smallest number of sample points and bins previously studied. Increasing the number of sample points can improve the resolution of the search, as can increasing the number of bins. In this case, however, the smallest previously-studied values yield a fitness landscape with numerous local optima, which is all that is required to demonstrate the benefits of dictionary exclusion. Increasing the number of sample points or iteration number both linearly increase the cost of fitness evaluation, which is in the tightest loop of the evolutionary algorithm.

4.2.2 The Contracting Simplex Representation

The representation used in this study is referred to as the Contracting Simplex Representation (CSR). It is a restricted version of the WUC representation introduced in Chapter 3, so it is a *generative representation* (Hingston et al. (2008)) which performs real parameter estimation with a discrete chromosome. In this study, only the center operation is utilized. Therefore, a d-simplex has $d+1$ possible commands – one for each of its vertices. The CSR is a very simple generative representation; it has a small number of commands that all do very similar tasks.

An initial simplex and the results of executing two of these commands are shown in Figure 4.2 for a 2-simplex. Each command selects one of $d+1$ possible disjoint subsets of the simplex; any point inside the simplex is contained in exactly one of the resulting contracted simplices (see Lemma 3.1). This means that, by iteratively choosing the unique move that leaves a given point inside the initial simplex inside the new simplex, we may cause the centroid to approach arbitrarily close to the point. This neglects points on a boundary between two of the partitions, but the set of boundary points have measure zero and so are vanishingly rare (see Theorem 3.1).

The fact that a given point inside the initial simplex is inside only one of the possible contracted simplices means there is the property that a sequence of moves which yields a simplex containing a point p is unique among sequences of moves the same length. In other words, the CSR uniquely approximates any point inside its initial simplex for any

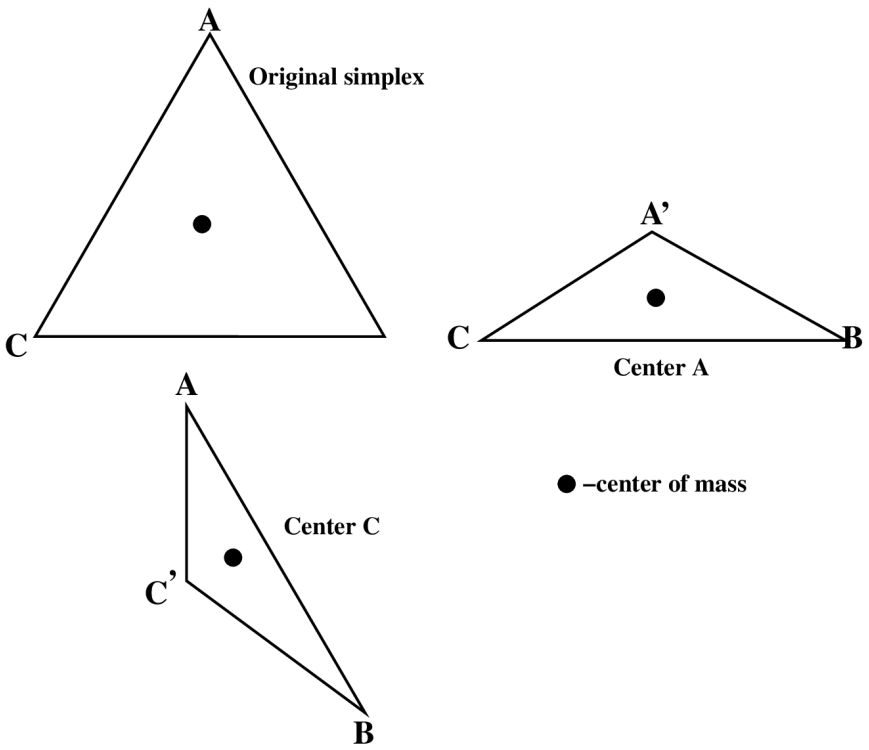


Figure 4.2: Shown is an initial simplex in 2 dimensions and two of the possible centering moves, moving vertices A to A', or C to C'. The center of mass is the represented point.

given length of a sequence of commands. Longer sequences of commands are better able to approximate a given point, and the length of the sequences of commands used is called the *representation length* of an instance of the CSR.

4.2.3 Dictionary Exclusion

The idea of dictionary exclusion was first tested with a less efficient representation, the Sierpinski representation (Ashlock et al. (2009)). The Sierpinski representation is similar to the CSR representation but requires 2^d commands to function in d dimensions. The idea of dictionary exclusion is relatively simple. Once there is a discrete representation for real

parameters like the CSR, any solution located may be stored in a dictionary as a string of commands. The objective function is then modified as follows: A prefix length P is chosen. Any member of the population which has a string of commands that has the same prefix as a solution previously stored in the dictionary is awarded a fitness of zero, i.e. excluded from consideration.

The evolutionary algorithm is run many times. The best solution located in each run is placed in the dictionary and used to prevent that solution being located again. Checking if a string is in the dictionary requires constant time (proportional to $\log(P)$) and also, when a population member is excluded in this fashion, avoids a relatively expensive fitness evaluation involving iterating 121 points. This means the cost of dictionary exclusion is nugatory. While the algorithm must be run many times to find many high fitness solutions, it also achieves the goal of niching, broad and diverse search, without the cost of computing the number of "nearby" points needed to apply a niching penalty.

The prefix length is important. If it is too short then the region of the search space excluded is quite large and so may exclude additional good solutions. If it is too long then multiple good solutions may be located that are actually on the same peak of the fitness landscape. The fitness landscape for generalized Julia sets is remarkably rugose (see Section 2.4.1) – rare tiny regions contains millions of optima – and so the problem of tuning the prefix length was deferred to future studies.

The mechanism of dictionary exclusion looks as if it could be applied to any discrete representation – and it can – but this application may not be profitable. There are two

qualities that make the CSR such that dictionary exclusion will work well on it. First, at a given representation length, the CSR represents sets of parameters *uniquely*. A many-one representation could locate the same optimum over and over with different genes; the CSR cannot. The second important factor is that instances of the CSR which share a prefix represent points that are close to one another in parameter space. The longer the shared prefix, the closer the points are. While these features limit the utility of dictionary exclusion, those limits include any bounded real parameter estimation problem.

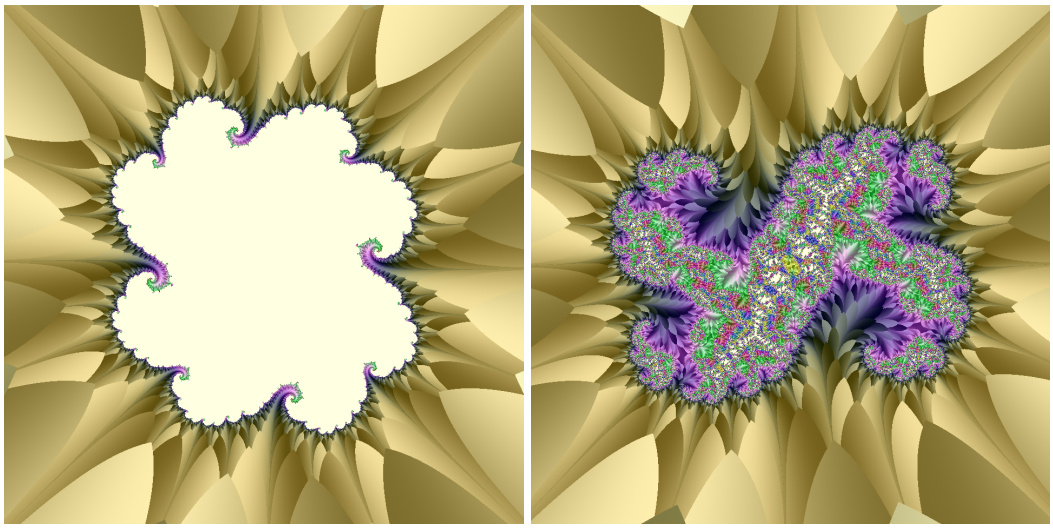


Figure 4.3: Examples of most-fit fractals from a collection of initial populations used in evolutionary runs. These fractals have two basic characters: mostly empty inside the circle used to test for iteration number, and fractals with little empty space. The instance ratio of the these two types among the evolved solutions is close to 1:1.

4.3 Design of Experiments

The hypothesis under test is that a version of an evolutionary algorithm using dictionary exclusion will locate more optima than that same algorithm when not using dictionary exclusion. While this may seem like an obvious outcome, it would verify that dictionary exclusion is working as intended, a feature which is made possible by this representation. It would also verify that the algorithm was previously finding identical high fitness solutions multiple times. Beyond simply confirming the hypothesis, the difference in the number of distinct solutions found with and without dictionary exclusion is useful information in determining whether there exists a problem worth solving. A relatively simple algorithm is chosen with a population size of 100 CSR chromosomes. Selection and replacement use size seven single tournament selection. Children are subject to single-point crossover and then the commands at 1-3 loci are changed to new commands. The changes to loci and the number of loci changed are selected uniformly at random.

The commands in a CSR chromosome have exponentially decreasing impact on the simplex specified the further they are along the chromosome's length. This is because the volume of the simplex is divided by the number of dimensions plus one in each step. This asymmetry in the importance of commands was the reason that one-point crossover was chosen.

The algorithms are run for 1,000,000 instances of tournament selection before a best fit solution is selected. If dictionary exclusion is in use, the solution is placed in the dictionary and, in any case, it is saved for examination. Two collections of 200 runs of the evolutionary

algorithm were performed, one with and one without dictionary exclusion.

The experiments searched $\mathbb{R}^6$ to locate sets of three complex parameters using representation length 10 and prefix length $P = 6$ for the runs employing dictionary exclusion. These parameters were chosen through preliminary exploration and probably could be beneficially tuned. These parameters, however, yielded conclusive results.

In order to search for generalized Julia sets with three complex parameters, six real parameters are necessary, representing the real and complex parts of the three complex parameters. This means that the initial simplex is a 6-simplex in $\mathbb{R}^6$. The initial simplex used has the following vertices:

$$v_0 = (-2, -2, -2, -2, -2, -2) \tag{4.2}$$

$$v_1 = (0, 0, 0, 0, 0, 2) \tag{4.3}$$

$$v_2 = (0, 0, 0, 0, 2, 0) \tag{4.4}$$

$$v_3 = (0, 0, 0, 2, 0, 0) \tag{4.5}$$

$$v_4 = (0, 0, 2, 0, 0, 0) \tag{4.6}$$

$$v_5 = (0, 2, 0, 0, 0, 0) \tag{4.7}$$

$$v_6 = (2, 0, 0, 0, 0, 0) \tag{4.8}$$

The initial simplex has center of mass (0,0,0,0,0,0) (the origin) and properly contains the regions of complex parameter space that do not yield uniformly low iteration numbers.

This fact is not obvious, but can be extracted from the theory of fractal geometry (Mandelbrot (1983)). It is also beyond the scope of this thesis and not critical to its conclusions.

4.4 Results and Discussion

The runs without dictionary exclusion located 142 distinct best fit solutions in 3-parameter generalized Julia fractal parameter space in 200 runs. With dictionary exclusion in place, 200 of 200 runs found distinct sets of parameters. Two parameter sets were judged distinct if they differed at all; the 58 non-unique sets located were identical to some other located set to eight decimal points in each of their six real parameters. Since dictionary exclusion constructively forces best fit solutions to be distinct, achieving 200 distinct results when dictionary exclusion is employed is exactly the conclusion which was expected. Finding only 142 distinct solutions without dictionary exclusion demonstrates that there was a problem to be solved.

Figure 4.4 shows the final fitness values for the 200 runs performed with and without dictionary exclusion. Since dictionary exclusion prevents locating the same best fit solution again, it lowers fitness. Left to its own devices, the evolutionary algorithm locates some high fitness solutions multiple times. The plots in Figure 4.4 demonstrate that dictionary exclusion is forcing the algorithm to discover high fitness solutions it would not have otherwise discovered in 200 runs. The higher fitness from not using dictionary exclusion results from the rediscovery of high fitness fractals that the dictionary prevents – a more di-

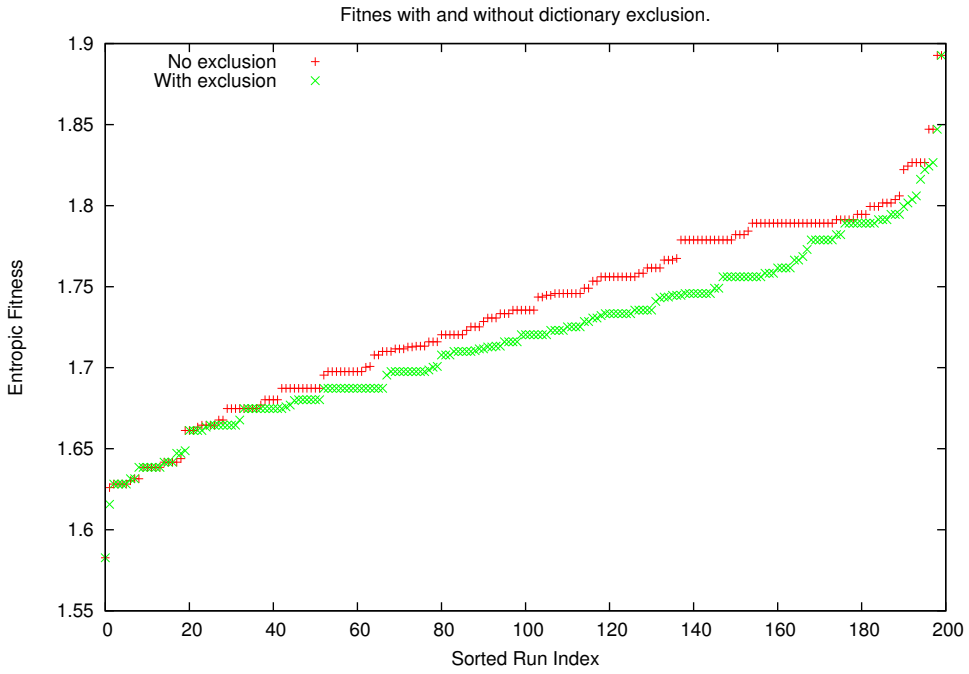


Figure 4.4: Shown are the fitness values, sorted into increasing order, of the runs with and without dictionary exclusion.

verse collection of fractals possesses lower average fitness. While these solutions are lower fitness, the entropic fitness function finds complicated fractals for later sorting by visual inspection. Having higher overall fitness does not necessarily translate to having better results; obtaining a variety of good results is the goal, and is what causes this particular problem to be an excellent candidate for dictionary exclusion. Figure 4.5 shows a selection of renderings of the best fit solutions located with dictionary exclusion.

The execution speed of the algorithm with and without dictionary exclusion is very similar; dictionary exclusion was slightly quicker because worse fractals take less time to evaluate fitness for. The fitness function requires from one to one hundred twenty iterations of the formula that computes members of the sequence for a given complex number

to establish its iteration number *and* it does this 121 times – once for each point in the sampling grid. Checking a gene for dictionary exclusion is very fast and saves this work, but it does not save it often. The evolving population of CSR commands rapidly evolves to avoid excluded regions of the search space and so this savings is not often realized.

4.5 Conclusions and Next Steps

This study demonstrates the ability of dictionary exclusion to locate a greater diversity of high fitness solutions at a low cost. The renderings of Julia sets shown in Figure 4.5 demonstrate that not only is the space rich in optima, solutions near those optima yield fractals with dissimilar appearance. Among the 200 distinct parameter sets located with dictionary exclusion, the renderings were all quite distinct from one another.

Real parameter optimization is a pervasive application of evolutionary computation. Dictionary exclusion is a generic technique for forcing a diverse search that applies to any real parameter optimization problem. It is only useful when there are many optima and is more useful when the fitness measure is a surrogate or indirect measure of the properties which are actually desired. With these caveats, the dictionary exclusion technique, enabled by the CSR's transformation of real parameter optimization into a form of dictionary-storable discrete string chromosome, has the potential for broad application.

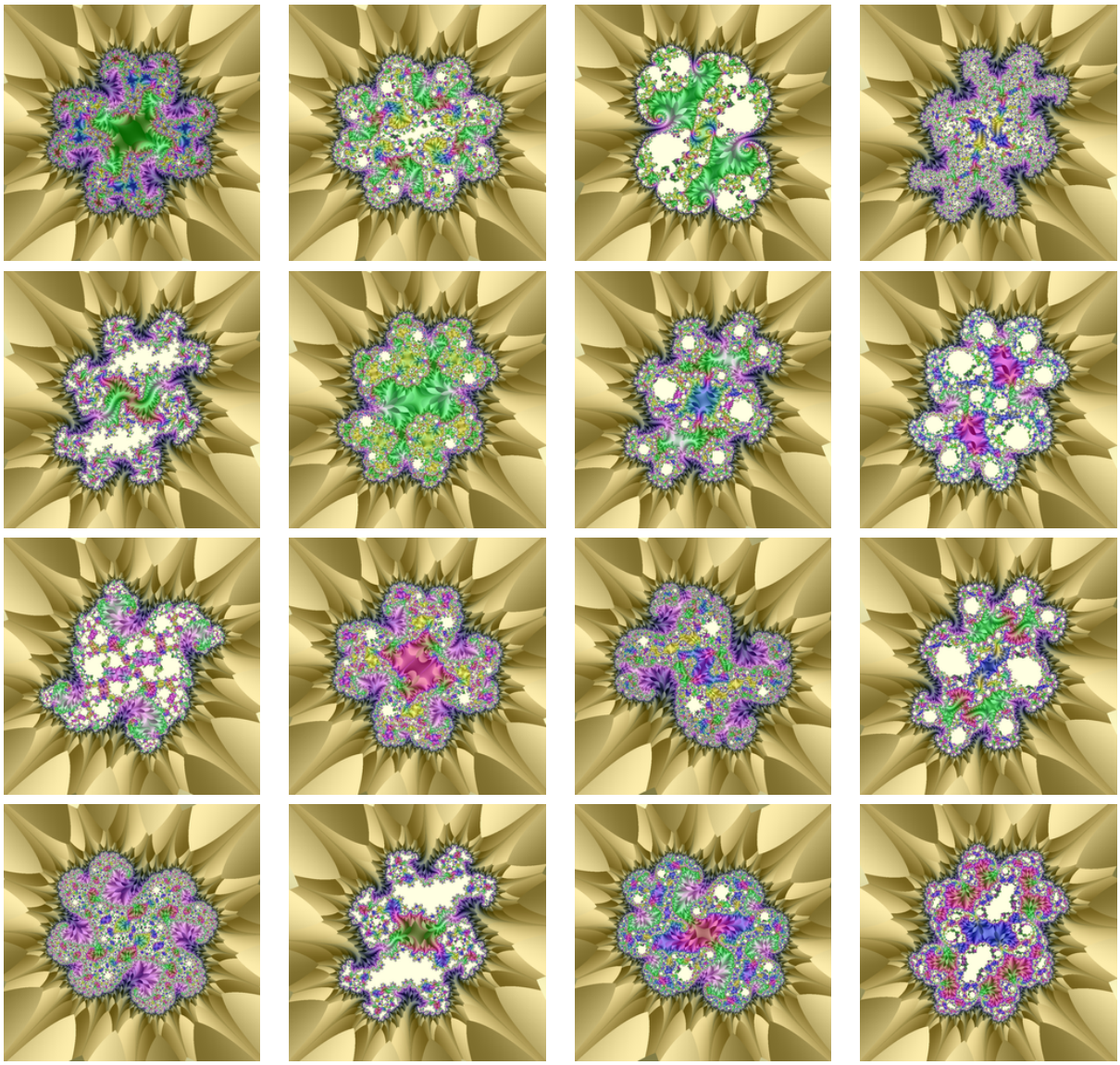


Figure 4.5: A sampling of evolved fractals from the dictionary-exclusion runs.

4.5.1 The Julia Set Fitness Landscape

The theory of fractal geometry (Mandelbrot (1983); Barnsley (1988)) is beyond the scope of this study, but is discussed briefly in Section 2.4.1.

If indefinite precision arithmetic is used, all of the optima are accessible to the CSR by increasing the representation length. There are regions of the parameter space with finite hypervolume that contain an infinite number of optima. The fitness landscape is, itself, fractal with infinitely complex structure. This means that using dictionary exclusion, while encouraging the location of a broad variety of high fitness solutions, will also prevent the location of an infinite number of optima in any given run. These facts suggest that the fitness landscape defined in this study is a challenging and interesting one for testing systems that attempt to locate multiple distinct high fitness solutions. The evolved solutions also have appealing visualizations.

This study chose three-complex parameter Julia sets to demonstrate that the CSR can function in a respectable number of dimensions, but its complexity scales linearly with dimension. This means that if more interesting fractals are desired, then increasing the number of parameters which are being searched for is a virtually trivial generalization of the search process. The practical limit is that the number of parameters should be less than the iteration limit. The number of sample points and the iteration limit can also be increased to obtain a space containing more fractals.

4.5.2 Multi-Resolution Search With the CSR

A given instance of the CSR is defined by its initial simplex, the representation length, and the prefix length used for dictionary exclusion if it is employed. Section 4.5.1 noted that the use of exclusion results in a large number of optima being discarded. It is not difficult to roll this back.

The commands in a CSR chromosome modify the initial simplex. After each command is executed, a new simplex is specified. This means that the excluded region associated with a prefix of a CSR chromosome is, itself, one of the simplices that the expression process passed through on its way to specifying the final simplex encoded. If a given chromosome encodes an interesting solution, the exclusion simplex could then be used as the starting simplex of a new search. This search will be restricted to the excluded region and so acts to locate high fitness solutions in this region.

This thought experiment demonstrates that it is possible to use the CSR to partition a search space into simplices containing interesting parameter sets, and these simplices can be used to direct search. It is possible to make the process more conservative as well. Either the exclusion simplex may be scaled about its centroid to include a larger region about the solution of interest *or* a starting simplex of reasonable scale may be translated to place a solution of interest at its centroid.

There is even the potential to automate multi-resolution search. An initial search at a moderate representation length can be used to find promising regions of the search space. These regions can then be prioritized, and new instances of CSR search can be initialized to

refine the parameters in the interesting regions. For the Julia parameter location problem, this is ideal because one simplex can contain an infinite number of distinct optima. Most problems will not have this bizarre, fractal character, but automated refinement could still be used as a form of hill-climbing. When the prefix for dictionary exclusion is set to a short value, it causes the algorithm to tile the space with large simplices – permitting exhaustive search of a hilltop at a user-selected resolution.

4.5.3 A Possible Hybrid CSR Algorithm

The CSR is ideal for searching rugose parameter spaces; this is why the infinitely rugose fractal parameter selection problem was chosen for the initial demonstration. For searching a smooth space, the CSR is somewhat cumbersome. Many parameter spaces are rugose only at higher levels and, once partitioned into small pieces, yield relatively smooth subspaces. This suggests that beginning with the CSR, and then handing off to a different optimization algorithm, for example differential evolution (Neri and Tirronen (2010)), may prove to be an effective technique.

At present, differential evolution has trouble operating in a large number of dimensions. The centering operations tend to flatten the simplex into splinter-shaped objects. This means that a simplex specified by the CSR has far greater diameter in some directions than others. In effect, it "squashes" the space it specifies, removing almost all variation in some directions. Therefore, the CSR may be able to not only hand differential evolution smooth subsets of a rugose search space, but also perform effective dimension reduction

before handing off.

4.5.4 Alternate Fractal Fitness Functions

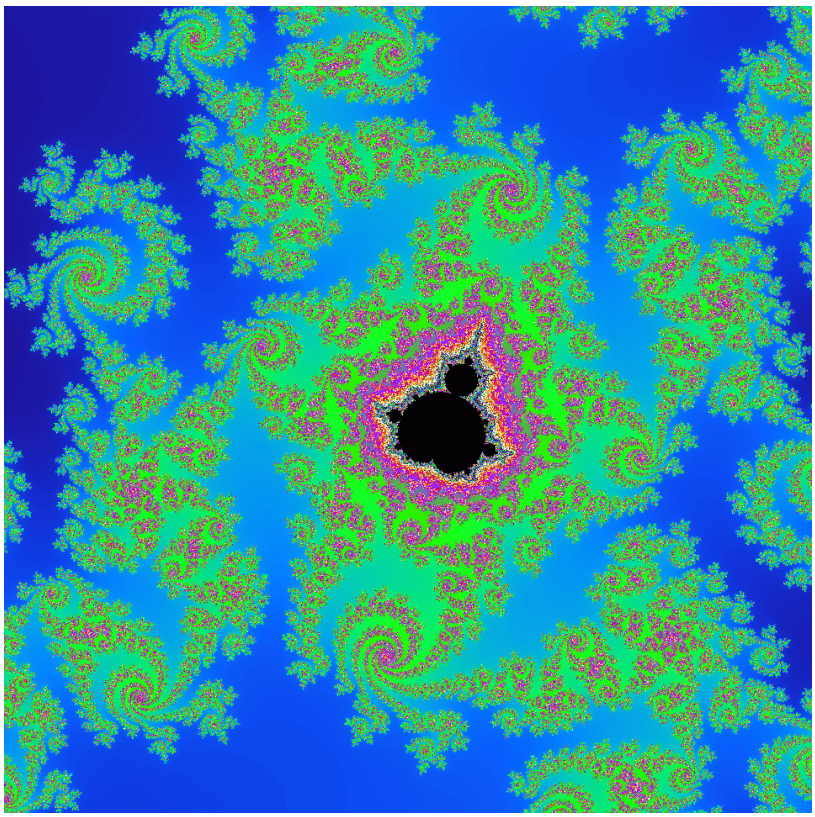


Figure 4.6: An example of a ”minibrot”.

Earlier studies that seek to evolve fractals have used the variance of the iteration number as a fitness function (Ashlock and Tsang (2015)). Another approach is to supply target iteration numbers for the sampling grid (Ashlock and Jamieson (2008)). These target numbers form a mask that specifies the approximate character of the fractal. A mask with large iteration numbers near the middle and smaller ones near the edge proved adept at locat-

ing “minibrots” (small copies of the Mandelbrot set found within the Mandelbrot set). An example of a minibrot is shown in Figure 4.6.

The sampling scheme returns a grid of points. All of the fitness functions devised thus far perform calculations on those points to obtain a fitness value – but there are many, many other possible functions. It is possible to control the fraction of empty space, regional variability, or even use the iteration numbers of an intriguing fractal as a target.

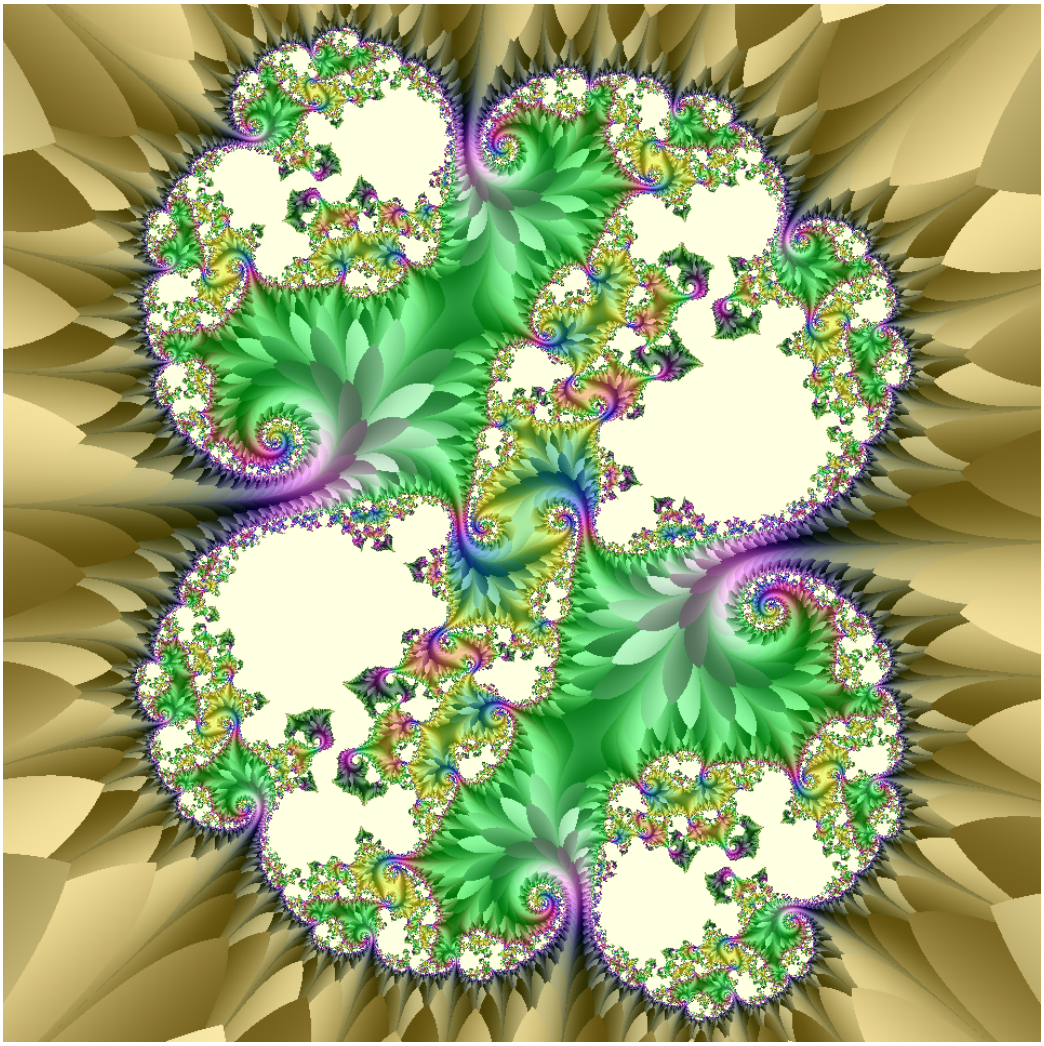


Figure 4.7: A full-sized rendering of an evolved Julia set.

Chapter 5

Point Packing in the Unit Hypercube for Population Initialization

5.1 Introduction

Point Packing in the Unit Hypercube is the process of filling the unit hypercube with a fixed number of points while attempting to maximize the minumum distance between any two points (see Section 2.3). This chapter examines the dual problem of packing as many points into the hypercube as possible, given a specified minimum distance. In (Ashlock et al. (2015)), two representations for evolving solutions to the point packing problem were compared. This earlier study used the $n = 8$ point version of the problem to compare the representations. This small problem size was used for the comparison because the representations used in the earlier study did not scale efficiently. This chapter introduces

a new representation to address this problem by specifying a large number of points with relatively few real parameters. It also introduces a new application for the point packings generated, point packing initialization, mentioned in Section 2.3.1 and explained in more detail in Section 5.2.

When using evolutionary computation with an initial population of randomly-generated points, the problem is to get points into the basin of attraction of the desirable, usually global, optimum. A unimodal function has a universal basin of attraction and so becomes difficult only if it is in a large number of dimensions. Two sources of potential trouble for evolutionary optimization are *deceptive* functions and *multi-modal* functions.

Point packing initialization is a method which attempts to space the initial population out evenly in the search space. Performing this sort of even-handed distribution with structures like a sampling grid suffers from the curse of dimensionality. The representation in this study is inspired by a property of irrational numbers, and is used to form a point packing algorithm to generate well-spaced-out sets of points of tunable size that suffer far less from the curse of dimensionality.

5.2 Background

The goal of this study is to validate a method for evenly distributing an initial population through a hyper-rectangular search domain. To achieve this, a point packing of the search domain is located and these points are used for the initial population, instead of plac-

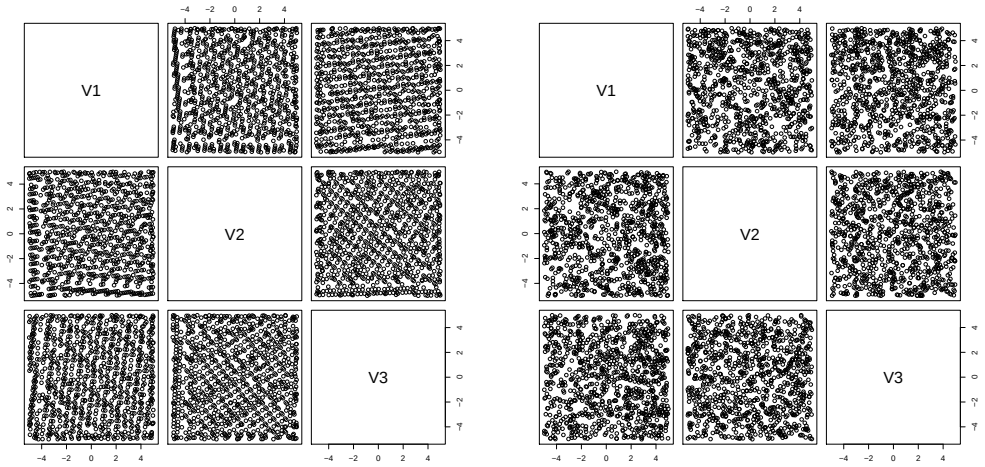


Figure 5.1: A contrast of an irrational point packing (left panel) with a random initialization (right panel) in three dimensions with $-5 \leq x, y, z \leq 5$. Each set has 1000 points and shows XY, XZ, and YZ viewpoints on XYZ-3 space.

ing points uniformly at random. This technique is called *point packing initialization*. An initial study (Ashlock and Graether (2016)) checked the effect of initializing evolutionary search with point packings and obtained positive results. The technique used in the previous study, however, located very high quality point packings with a time-consuming process. The method described here initializes the population much more quickly. The standard initialization technique used by many evolutionary algorithms is to place the points uniformly at random. Figure 5.1 contrasts such a random initial population, in three dimensions, with one generated by point packing. These plots were produced with the package **R** using the plot() function (R Core Team (2013)).

In a small number of dimensions, this sort of population initialization is not likely to yield a large performance improvement. As the number of dimensions grows, populations with manageable sizes will be increasingly unlikely to sample the problem space well. This

is one of the motivations for using random initial populations, in analogy to the sampling strategies used for numerical techniques like Monte-Carlo integration (Press et al. (2007)). Point packings located by the technique described in this chapter are likely, based on theory described subsequently, to have better and better coverage properties as the dimension of the search space increases when compared to randomly selected points. These improved coverage properties are also likely to result in improved performance on deceptive and multi-modal functions by decreasing the likelihood that a basin of attraction is "missed" by a random initial population.

A function is *deceptive* if the optimization process of the algorithm moves away from the basin of attraction of the global optimum. This means that no function is intrinsically deceptive; the quality only exists relative to the mechanic of the optimizer being used. Functions are usually called deceptive if they are deceptive for a gradient-following optimizer in a large fraction of their search space. Equation 3.9 used for testing in this study is an example of a function that is deceptive to a gradient optimizer.

Three multi-modal functions are used for testing in this study. The first, given in Equation 5.1, has two optima and is constructed so that the inferior optimum has a larger basin of attraction. The second, given in Equation 5.2, is a typical type of multi-modal function; it uses cosine waves to create a function with dozens to thousands of optima as the dimension of the problem increases. The third multi-modal function is an example of a function without a formula; it estimates the aesthetic quality of a fractal generated from a set of complex parameters. This function has, in abstract, an infinite number of distinct optima

within the finite search domain used. Due to the use of double precision arithmetic, it is limited to a very large number of local optima.

It is possible to allocate fitness trials dynamically, based on information gathered as trials proceed. *Bandit Optimization* (Kunanusont et al. (2017)) is an example of such a dynamic technique. Dictionary exclusion techniques, such as those presented in Chapter 4 or in (Ashlock et al. (2009)), are also examples of methods that dynamically allocate fitness trials, using the results of earlier trials to constrain later ones. The technique in this study performs its allocation at the beginning of each evolutionary run, without reference to previous runs.

Hierarchical evolutionary algorithms (de Jong et al. (2004)) use evolutionary algorithms to prepare initial populations for other evolutionary algorithms. In (Rahnamayan et al. (2007)) the authors use opposition based learning to generate initial populations for an evolutionary algorithm; this is much more computationally expensive than the techniques presented in this chapter.

The key fact which this method of generating point packings is inspired by is that the multiples of an irrational number $(mod\ 1)$ form a dense subset of the unit interval (Birkhoff (1931)). A subset of a set is *dense* if every member of the set has a member of the subset within any positive distance of it. The classical example of a dense subset are the rational numbers within the real numbers. Birkhoff's result generalizes to hyper-rectangular domains with rational side lengths as follows. Suppose $\vec{w}$ is a vector with all irrational coordinates; if the ratio of any two coordinates of $\vec{w}$ is also irrational, then

multiples of the vector, when taken modulo the boundaries of the rectangle by making it toroidal, will form a dense subset.

Unfortunately, any floating point number system operates exclusively in the rational numbers, so the construction of such a vector is impossible on a computer. However, the idea to use a dense subset was simply to provide a set with extremely good coverage, of which a smaller subset could be selected for the actual point packings. Thus arose the idea to mimic the effect of a dense subset as well as possible.

Taking multiples of an irrational number $(mod\ 1)$ provides a dense subset due to the fact that no multiples of an irrational number will ever equal 0 $(mod\ 1)$, even if an infinite number of multiples are taken (and indeed, an infinite number of multiples must be taken to generate the dense subset). So when moving from irrational numbers to rational numbers, the goal is to have "infinite number" be replaced with "very large number". If rational numbers are considered in their lowest form, then the larger their denominators are, the more multiples are required for the number to equal 0 $(mod\ 1)$. This idea also extends to $(mod\ k)$ for any integer k. In search of such a number, there is a property of algebraic numbers which is extremely useful. First, some definitions.

Definition 5.1 *A number $r \in \mathbb{R}$ is called an* **algebraic number** *if there exists some non-zero polynomial with integer coefficients $p(x)$ such that $p(r) = 0$.*

Definition 5.2 *Suppose $r \in \mathbb{R}$ is an algebraic number. Then the* **degree** *of r is the degree of the lowest degree non-zero polynomial with integer coefficients $p(x)$ such that $p(r) = 0$.*

An approximation theorem by Joseph Liouville[1], found in (Burger and Tubbs (2004)) as Theorem 1.1, guarantees that any good rational approximation to an algebraic number of degree 2 must have a relatively large denominator. Thus, if a vector $\vec{v}$ is constructed with coordinates that are algebraic numbers of degree 2, and the ratio between any two coordinates is also an algebraic number of degree 2, then multiples of the rational approximation of $\vec{v}$ on a floating-point number system should achieve the desired coverage of a hyper-rectangular domain. An easy source of algebraic numbers of degree 2 are simply the square roots of non-perfect square integers. Furthermore, if selected properly, the ratio between any two of them will also be an algebraic number of degree 2, and it retains these properties when scaled to be a unit vector. The specific construction of $\vec{v}$ is explained in detail in Section 5.4.1.

The multiples of such a vector are the raw material for initializing a population. Algorithm 5.1 is used to select the initial population. It is a simple linear congruential generator with tunable parameters. The size of the resulting population is controlled by the parameter Q, and tuning this parameter is covered in Section 5.4.1.

Algorithm 5.1

Input: *Unit vector $\vec{v}$ as described above.*

A failure limit K,

A minimum distance Q.

A hyper-rectangular domain with integer side lengths D

[1]https://en.wikipedia.org/wiki/Joseph_Liouville

Output: *A set of points in* D

Details:

Initialize an empty set of points $P \subset \mathbb{R}^d$

Zero the number of failures.

Select a starting point $\vec{r}$ *uniformly at random in* D*.*

Using the points $\vec{u}_n = \vec{r} + n \cdot Q \cdot \vec{v}$*,* $n = 1, 2, \ldots$

Considered in order

If $\vec{u}_i$ *is at least distance* Q *from all* $\vec{p} \in P$

Add $\vec{u}_i$ *to P*

Zero the number of failures

Else increment the number of failures

While(number of failures $< K$*)*

Return(P*)*

The use of Q in the scaling of the vector $\vec{v}$ in Algorithm 5.1 prevents a point from automatically being too close to the point before it in the consideration order. The algorithm is a greedy one and the distance restriction means it can only find a finite number of points. Sometimes there is a long run of points that are too close to points already chosen, which is why a number of failures (in a row) to find a new point are tolerated before giving up; in this work K was set to 100,000. While the algorithm specifies that D have integer side lengths, use on domains with non-integer side lengths is possible and is discussed in more

detail in Section 5.6.3.

Suppose we place the vector $\vec{v}$ from Algorithm 5.1 at points in the unit square and start with $\vec{r}$ at the origin. Figure 5.2 shows the number of points located for minimum distance $Q = 0.1$ on the unit square. This figure shows the variability of the size of initial populations generated and the criticality of using properly-constructed vectors, as described in Section 5.4.1.

5.3 Specification of Test Functions

Four test functions are used to test the population initialization method. The first is the hidden hill function, explained in detail in Section 3.3.2. This function has a unique global optimum at the origin whose basin of attraction is a unit ball centered at the origin. At all other points in space the gradient points directly away from the global optimum.

The *dual hill* function is given by

$$f(x_1, x_2, \ldots, x_n) = \frac{3}{1 + 2\sum_{k=1}^{n} x_i^2} + \frac{2}{1 + \sum_{k=1}^{n} \left(\frac{x-4}{3}\right)^2} \tag{5.1}$$

A picture of this function in two dimensions is given in Figure 5.3. This function has a tall, narrow hill at the origin and a broader, lower hill centered at the point where $x_i = 4$, for all i. This function tests the ability to find a better optimum with a smaller basin of attraction. This function is, in general, easier than the hidden hill function once the search domain is

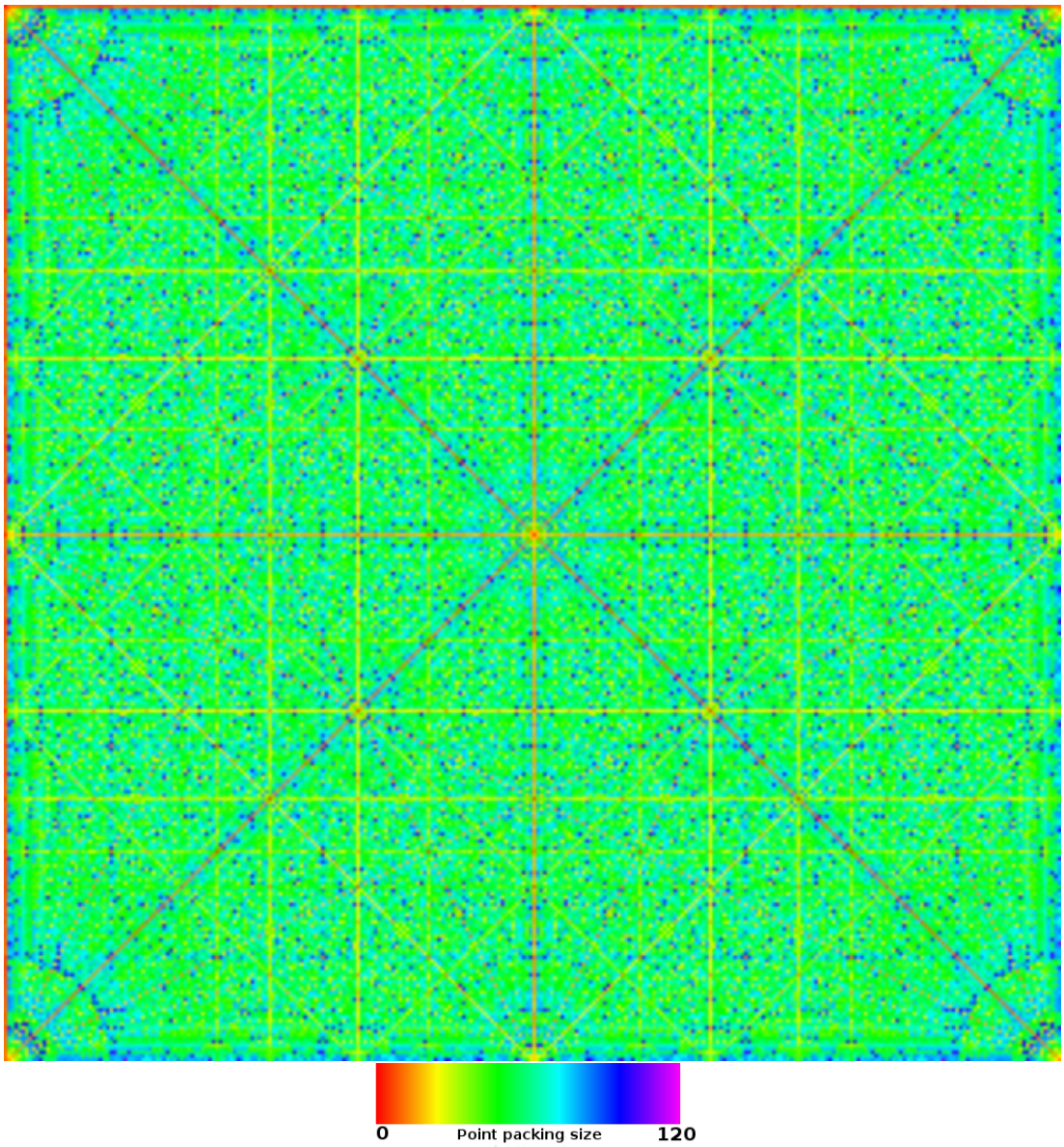


Figure 5.2: Shown is a heat map of the fitness, for the point packing problem in the unit square, of single vectors using a sampling depth of 5000. Vectors are from (0,0) to the lower left corner of each displayed pixel. The minimum distance parameter used is $Q = 0.1$

large enough to incorporate all of both functions' features. This difference in difficulty is apparent in the experimental results.

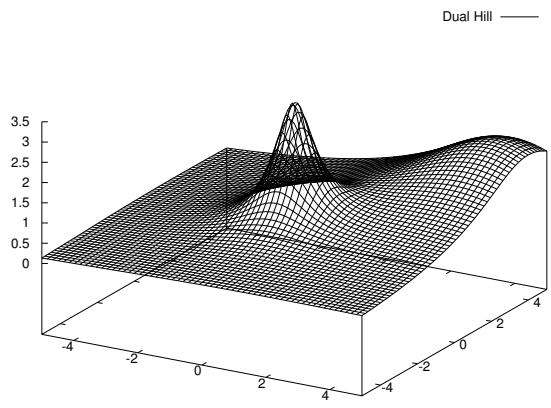


Figure 5.3: The dual hill function in two dimensions.

The *multi-hill* function is a simple sum of cosine waves with a linear trend given by

$$h(x_1, x_2, \ldots, x_n) = \sum_{k=1}^{n} \cos(10x_i) + \frac{1}{20}x_i \tag{5.2}$$

This function has a very large number of hills, all of different height because of the linear trend. This function tests the relative ability of algorithms to locate the best optima in a very optima-rich search space.

The same *Julia parameter location* function from Chapter 4 is used to locate complex parameters that generate interesting looking *generalized Julia sets*. An example of a rendering of one of these fractals appears in Figure 5.5. This fractal appears as a subset of the complex plane and depends on N complex parameters, $z_1, z_2, \ldots, z_N$. More information

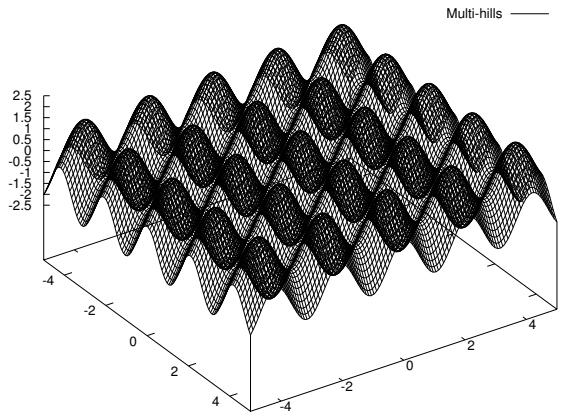


Figure 5.4: The multi-hill function in two dimensions.

on generalized Julia sets and the fitness landscape for this function can be found in Section 2.4.

The following is also identical to the method used in Chapter 4, but is included here for completeness. An equally spaced 11×11 grid of points is placed, aligned with a square with corners $-1.6 - 1.6i$ and $1.6 + 16.i$ in the complex plane. The iteration number of each of these 121 points is computed. The resulting iteration numbers are then binned into sixteen bins numbered $0, \ldots 15$ using the formula $bin = \frac{I*16}{121}$, were I is the iteration number. These bins are normalized by dividing by 121 to form an empirical probability distribution $p_0, p_1, \ldots, p_{15}$ on the bins. Identical to Equation 4.1, the objective function is

$$E = -\sum_{k=0}^{15} p_k \cdot \log_2(p_k), \tag{5.3}$$

the *Shannon entropy* of the empirical distribution. Shannon entropy is maximized when a

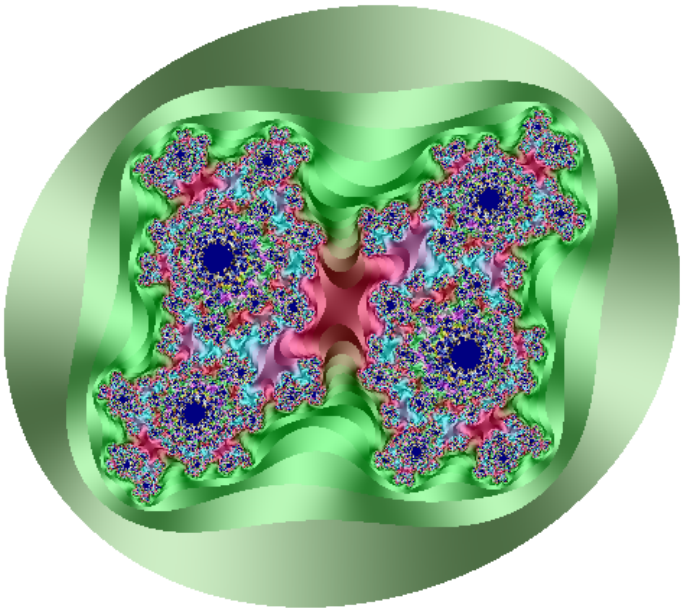


Figure 5.5: An example of the fractal specified by an evolved parameter set.

probability distribution is uniform and so this objective function rewards splitting the 121 sample points as evenly as possible among the bins.

5.4 Experimental Design

A simple evolutionary algorithm, described subsequently, is initialized with a population of 1000 genes and then run for 100,000 fitness evaluations. The hidden hill, dual hill, and multiple hill functions are tested in 3, 4, 5, and 6 dimensions and the Julia parameter location function is tested in 4 and 6 dimensions (thereby locating 2 or 3 complex parameters, respectively). Each experiment is run 100 times with random and point packing initialization.

5.4.1 Choosing the Packing Vector and Initial Population

The theoretical requirements on the direction vector $\vec{v}$ from Algorithm 5.1 is that the coordinates, as well as the ratio of any two coordinates, be rational numbers with large denominators. Rational approximations to algebraic numbers of degree 2 make excellent candidates, and therefore the coordinates are first filled with the square root of successive prime numbers. A proof by Hippasus[2] in antiquity can be used to demonstrate that these numbers have irrational ratios. Furthermore, their ratios will be algebraic numbers of degree 2. The numbers are then scaled by dividing by the smallest power of 2 that beings them into the range $1 < x_i < 2$. If the coordinates differ too much in size then the filling of the search domain will preferentially fill one side of the search domain first. This normalization prevents this inefficiency. The vector is then divided by its length to obtain a unit vector; both sorts of scaling do not disrupt the properties of the ratio of the coordinates or the coordinates themselves of $\vec{v}$.

For each run of the evolutionary algorithm, the coordinates of $\vec{v}$ are randomly permuted, each coordinate is negated with probability 0.5, and the starting vector for use in Algorithm 5.1 is selected uniformly at random. The effects of these randomizations on $\vec{v}$ and $\vec{r}$ ensure that the well-spaced initial populations (see Figure 5.1) are different from one another in each run. The initial populations for point-packing experiments are chosen with Algorithm 5.1.

The use of a random starting point $\vec{r}$ for point packing initialization and the shuffling

[2]https://en.wikipedia.org/wiki/Hippasus

and negation of the coordinates of $\vec{v}$ are to randomize the points chosen by point packing initialization while permitting them to retain their desirable coverage properties. These techniques give us $2^n \cdot n!$ forms of $\vec{v}$. This sampling technique is derived from the symmetry group of the hypercube. The symmetry group of the hypercube is the *hyperoctahedral group* (Conway and Smith (2003)).

Table 5.1: Values of Q used in initialization of populations.

Dimension	Distance Q
3	0.95
4	1.8
5	2.68
6	3.6

The parameter Q was tuned by successive bifurcation to locate values, in each of 3-6 dimensions, that cause Algorithm 5.1 to yield between 900 and 1000 points. Points from this set are then randomly copied to fill out the full 1000 member population. It is important that the point packing initializer generate no more than 1000 points; if it generates more points than this then some points will have to be neglected, and it may only be packing part of the space, as the points fill in along the way the "slope" of $\vec{v}$ wraps across the search domain. These values are given in Table 5.1. They are specific to the search domain $-5 \leq x_i \leq 5$.

5.4.2 The Evolutionary Algorithm

A relatively simple evolutionary algorithm was used. All fitness functions were chosen so that the goal is to maximize fitness. Selection and replacement are performed with size seven single tournament selection. Reproductions are completed on the two copied genes by subjecting them to two point crossover and from 1-3 point mutations. The mutation operator adds a Gaussian random variable with a standard deviation of $\sigma = 0.1$ to the coordinates selected for mutation. The relatively small standard deviation of the mutation operator is intended to bias the mutational part of the search in favor of hill climbing, a good choice for comparing the efficiency of the initial populations at locating particular basins of attraction.

5.4.3 Analysis Techniques

For the hidden hill and dual hill problems, the number of runs out of one hundred that found the global optimum are assessed. For the multi-hill and Julia parameter location functions, the *total maximum fitness* (Kim and Ashlock (2017)), which was explained in Section 3.3, is computed. This statistic is the average, over the course of evolution, of the populations current maximum fitness. This statistic is larger if a better solution is located and is also larger if the high fitness solution is located earlier. It forms a good single-number statistic for measuring performance on functions with large numbers of distinct optima.

5.5 Results and Discussion

The results for the hidden hill and dual hill functions are given in Tables 5.2 and 5.3, respectively. For both functions, the packed initialization was superior; the results suggest that more aggressive mutation operators or even larger population sizes are a good idea. The benefits of packed initialization were greater on the more difficult hidden hill function.

Table 5.2: Number of runs finding the global optimum in 3-6 dimensions for random and point packing initialization on the hidden hill function, along with p-values to four decimal places calculated via Fisher's exact test.

Dimension	Random	Packed	p-value
3	50	100	0.0000
4	5	23	0.0004
5	0	7	0.0140
6	0	1	1.0000

Table 5.3: Number of runs finding the global optimum in 3-6 dimensions for random and point packing initialization on the dual hill function, along with p-values to four decimal places calculated via Fisher's exact test.

Dimension	Random	Packed	p-value
3	94	100	0.0289
4	21	38	0.0128
5	0	7	0.0140
6	0	1	1.0000

The first two fitness functions give us the luxury of not only knowing the global optimum but having the algorithm locate it fairly often. These two functions are highly multimodel. The first has a global optimum that is seldom located by a simple evolutionary

algorithm, and the second does not have a known global optimum. In fact the details of which optima are possible to locate for the Julia parameter location function change somewhat whimsically with the implementation of real arithmetic used.

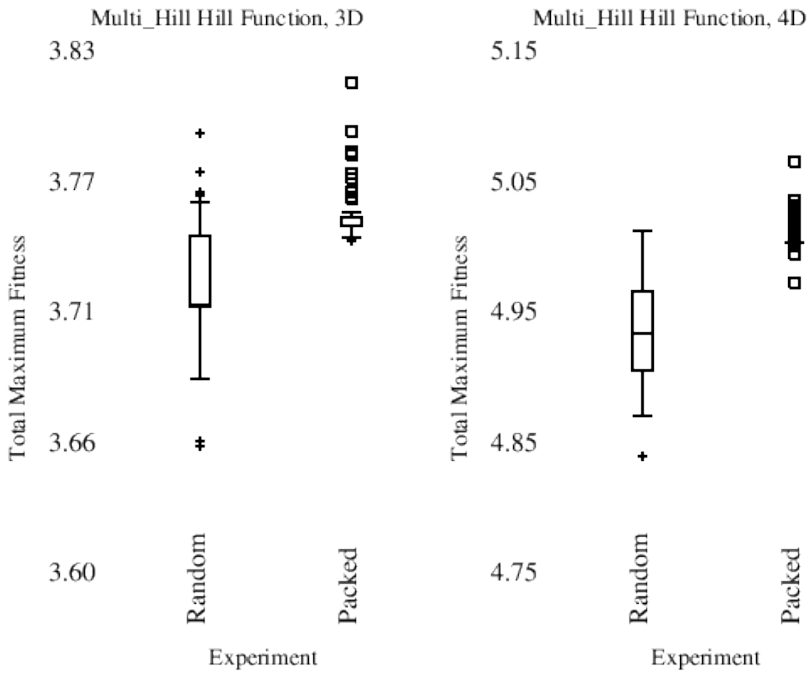


Figure 5.6: Box plots of the total maximum fitness for random and packed initialization of the multi-hill function in 3 and 4 dimensions.

The relative performance of the random and packed initialization on the multi-hill function are compared using total maximum fitness in Figures 5.6 and 5.7. For all for sets of experiments, point packing exhibits superior performance, and in fact locates higher quality solutions.

Figure 5.8 shows the results for random and packed initialization for two two cases of the Julia set parameter location problem. Using point packing still yields an advantage,

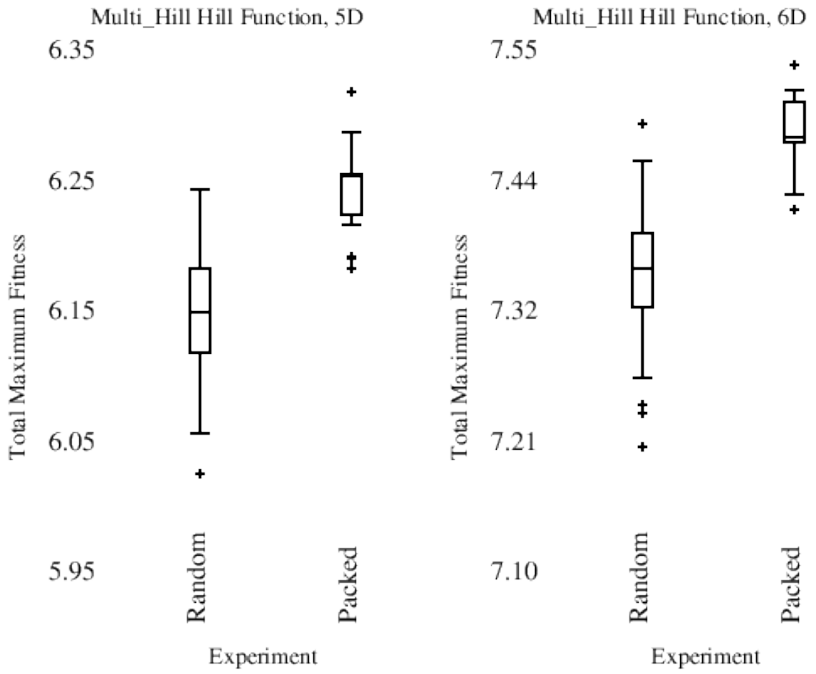


Figure 5.7: Box plots of the total maximum fitness for random and packed initialization of the multi-hill function in 5 and 6 dimensions.

but one that is not as pronounced as for the multi-hill function. The fitness landscape is extremely rough.

5.6 Conclusions and Next Steps

This study exploits the ergodicity of multiples of an irrational number in the unit interval, combined with techniques for point packing, to create an inexpensive population initialization scheme that yields improved performance of evolutionary optimization on deceptive and multi-modal landscapes. This performance boost can probably be improved

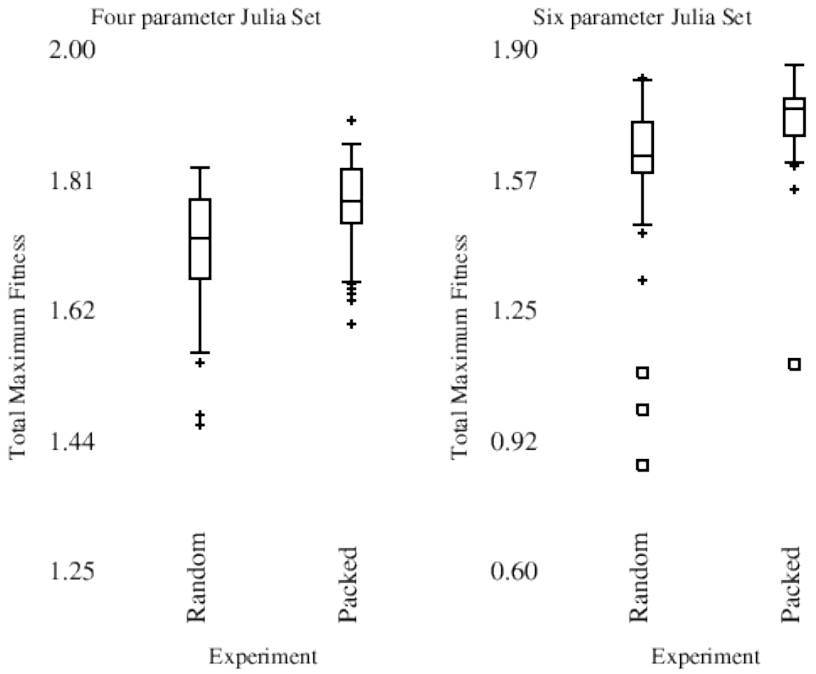


Figure 5.8: Box plots of the total maximum fitness for random and packed initialization of the Julia parameter set location fitness function in 4 and 6 dimensions (for 2 and 3 complex parameters, respectively).

as more is learned about both parameter tuning and the useful size of packed initialization populations. In this first study of the technique, conservatively large population sizes were used.

Point packing is able to place points into a rectangular search domain far more efficiently than regular grids (Ashlock and Graether (2016)), reducing the degree to which the technique suffers from the curse of dimensionality. Figure 5.1 shows that the distribution of points is more irregular, leaving large lacunae, for random initialization. This effect grows with dimension, something reflected in the increasing advantage of point packing initialization from 3 to 6 dimensions for the multi-hill function. While the dimension even begins to overwhelm the packed initialization at 6, causing the result to not be considered

significant, one cannot help but notice that random initialization failed to find the optimum of either function in any dimension above 4, while packed initialization successfully found the optimum of both functions in all dimensions tested. Point packing is not, however, immune to the curse of dimensionality – it just falls to it more slowly.

The Julia set parameter location fitness function was computationally far more expensive than the other functions used in this study. It required running an iterative process to completion on 121 sample points. This is at least weak proof of concept for the use of point packing initialization on expensive fitness functions. Adaptive techniques, like *bandit optimization* (Kunanusont et al. (2017)), are probably better for this sort of fitness allocation, except in their initial stages where little information is available. It may be that point packing is a way to get a good initial set of trials, just enough to jump-start the adaptive mechanism.

5.6.1 Functions Where Packing Initialization Doesn't Help

The Lorentz Hill function was tested in the preliminary work for this study (see Section 3.3.2). Packed and random initialization found the optimum every time; point packing yielded a minor advantage in time to solution, at a population size of 1000. However, this sort of unimodal function has been studied extensively and it is known that small population sizes, such as a size of two, are optimal. Point packing is meaningless at this population size and the standard optimizer with population size two out-performs the point packing initialized optimizer with 1000 points.

This example re-enforces the fact that point packing initialization is needed only when there are properties of the fitness landscape, such as deceptions or roughness, that cause large population sizes and even sampling to be desirable.

5.6.2 Structured Algorithmic Initialization

The point-packing algorithm is a simple greedy algorithm. Its power comes from exploiting intrinsic properties of irrational numbers; Figure 5.2 demonstrates that this property is useful. Point packing initialization is an example of selecting an initial population with a structured algorithm. It seems clear that there are many possible forms of structured algorithmic initialization and this may be a fruitful areas for additional exploration.

5.6.3 Domains With Non-Integer Side Lengths

For simplicity for the sake of this study, the side lengths of domain D in Algorithm 5.1 are restricted to being integers. Of course, this naturally gives rise to the question: what if the domain does not have integer side lengths? There are a couple potential solutions.

The first solution is the rather obvious one: simply rescale. If one of the side lengths of D is not an integer, then choose the closest integer and generate the point packing with Algorithm 5.1. Next, scale the coordinate for the dimension with a non-integer side length for each point in the point packing to fit that side length. Unfortunately, this type of scaling in one dimension only (or different scalings in different dimensions) will disrupt the spacing of the point packing. The purpose of choosing the closest integer for the initial point

packing before scaling is an attempt to minimize this disruption.

A more complete solution would be to construct a vector $\vec{v}$ which could fill the space in the same manner that it does for domains with integer side lengths. In that case, rational numbers with very large denominators in their lowest form were chosen. In this case, the key is to final a rational number which, when divided by the non-integer side length, still has a very large denominator in its lowest form. Problem cases can arise when the denominator of the side length divides the denominator of the vector's coordinate in that dimension (or at the very least has common factors). This is clearly somewhat problem-dependent, but should be relatively easy to avoid in general. Indeed, vectors constructed in the manner described in Section 5.4.1 should be effective on the vast majority of domains with non-integer side lengths, but is not guaranteed to work for them all.

5.6.4 Other Future Work

The coordinates of $\vec{v}$ could also be multiplied by random numbers near one to make the way the point packings interact with the search space more random.

This study initializes in a rectangular domain – and in fact one that happens to be a hypercube. The generalization to non-cuboidal domains is trivial. Domains with more complex shapes, perhaps with shapes induced by constraint functions, can be packed by the simple expedient of adding admissibility tests to the distance tests for accepting a point as a member of the initial population. There is a tactical question of which sort of exclusion to apply first; inexpensive constraints should come before distance exclusion, expensive ones

after.

The use of 1000-member populations in this study was a cautious choice. For expensive fitness functions there is a need to determine which population sizes are useful. The tuning of the minimum distance parameter Q was done by iterated bifurcation in this study and could easily be automated.

Chapter 6

Evolving Bijections for Warping Space

6.1 Introduction

This chapter describes an evolutionary computation system for finding continuous *data warps*, functions that reparameterize a continuous variable. If evolution produces a good solution, these data warps are normalizers for data that approximate the inverse of the cumulative distribution function (CDF) of the data. Beyond the value of being able to recover a good distribution model for a data set, data warps of this sort are useful for comparing the shapes of different distributions. Normalizing one data set with a data warp fitted to the inverse of the CDF of another permits a clear and intuitive visualization of the difference between the distributions. Comparisons can be performed between data sets by simply applying a standard normalization, like the affine normalization in Equation 6.1 below, but a fitted data warp makes it easier to see the difference from a standard reference.

Furthermore, if multiple distributions are compared, a fitted data warp permits visualization of differences from each member of the comparison group individually.

One such commonly used visualization method is the Q-Q plot, which plots the quantiles of two probability distributions against one another for the purpose of comparison (Wilk and Gnanadesikan (1968)). The most basic process of construction for a Q-Q plot is only possible when comparing two distributions of equal size. In this case, both distributions are sorted and then paired in ascending order to generate the ordered pairs for the plot. When comparing distributions of different sizes, interpolation is utilized to fill the gaps in the smaller distribution before the plot is constructed. The evolved models in this study seamlessly include the ability to interpolate.

The technique presented here requires the data all be both univariate and in the unit interval so that $0 \leq x_i \leq 1$ for all i. In order to work with data with other ranges, a simple affine normalization of the data into the interval [0,1] is performed:

$$N(x) = (x - min)/(max - min), \tag{6.1}$$

where *max* and *min* are the maximum and minimum values of any data item. Once a data warp has been located, this normalization can be easily inverted to place the evolved distribution back into its original data range.

The algorithm that locates a data warp for a set searches a rich space of bijections of the unit interval for a map that, when applied to a data set in the unit interval, attempts

to make the between-point intervals of the sorted data as close to uniform as possible. If $\{x_1, x_2, \ldots, x_n\}$ are the data sorted into ascending order then we are searching for a function $f(x)$ so that f is an increasing bijection of [0,1] with the following additional property: If we set $\delta_i = f(x_{i+1}) - f(x_i)$ and let μ be the mean value of the δs then $f(x)$ makes the quantity

$$\epsilon = \sqrt{\frac{\sum_{k=1}^{n-1}(\delta_k - \mu)^2}{n-1}} \tag{6.2}$$

as small as possible. Notice this is simply the sample standard deviation of the δ values. This objective function attempts to space the points out as equally as possible. This optimization yields a function that approximates the inverse of the CDF of the data, because it transforms the data into a nearly uniform distribution - one that would lie on the line $y = x$.

If $f(x)$ is approximately the inverse of the CDF of the data, then the approximate CDF is $f^{-1}(x)$, the functional inverse and the approximate probability distribution function (PDF) of the data is $(f^{-1}(x))'$. An essential feature of the evolvable data warps is that they are built from basic elements all of which are invertible, continuous, and differentiable, enabling the computations that permit the recovery of approximations of the CDF and PDF from the evolved approximate inverse CDF.

6.2 The Representation

The representation used to specify bijections of the unit interval takes the form of strings over a parametrized set of generators. The generators used are:

$$f_\alpha(x) = \frac{\alpha x}{(\alpha - 1)x + 1} \tag{6.3}$$

$$g_\alpha(x) = \begin{cases} f_\alpha(2x)/2 & x \leq \frac{1}{2} \\ \\ f_\alpha(2x-1)/2 + 1/2 & x > \frac{1}{2} \end{cases} \tag{6.4}$$

Where α is a real-valued parameter in the range $(0, \infty)$. Graphs of $f(x)$ and $g(x)$ for the parameter values $\alpha = 3$ and $\alpha = \frac{1}{3}$ appear in Figure 6.1. All the functions located are elements of the group of continuous bijections of the unit interval, $\mathcal{U}$. The data warp encoded by $f_3(x)$ distorts data toward the right side of the unit interval (toward one); $f_{1/3}(x)$ distorts data to the left (toward zero). The function $g_3(x)$ pulls data into the middle of the unit interval while its inverse, $g_{1/3}(x)$, pushes data toward the edges of the interval. The degree to which each of these functions warps the data is controlled by how far $log(\alpha)$ is from zero - the more distant from zero, the stronger the action of the function.

In addition to being group based, the representation is a *gene expression* representation (Ashlock and Ashlock (2012)). The representation has a length n and consists of two linear structures: the first is a string of length n over the four letter alphabet $\{f, F, g, G\}$, and the

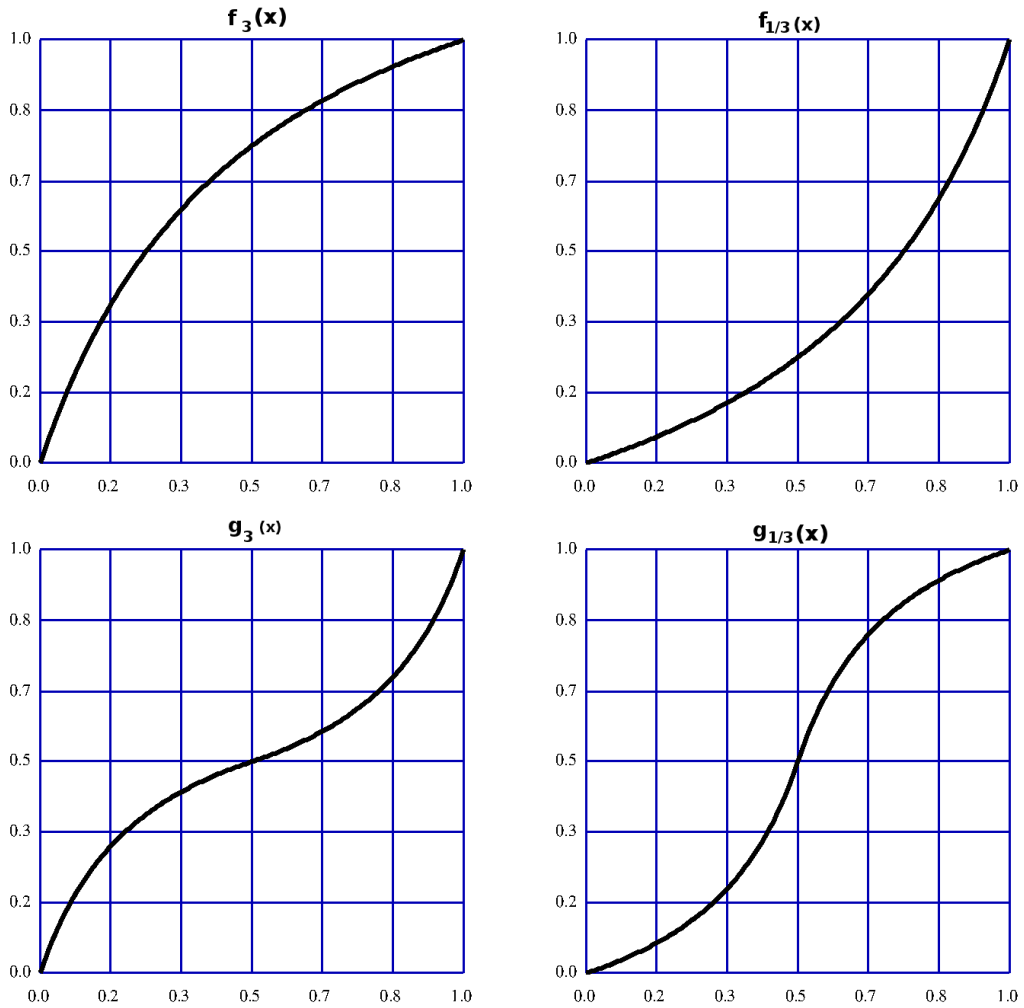


Figure 6.1: Example of the four types of functions used as a basis for the evolvable warp language. The top pair of functions shift the interval right or left, the bottom pair move the distribution toward or away from the center of the interval.

second is a list of n real parameters in a range $[-\tau, \tau]$. This pair of strings represents a left-to-right composition of elements in $\mathcal{U}$. Elements in $\mathcal{U}$ are specified by the corresponding letter and real parameter at the same index in the linear structure. If the real parameter is interpreted as γ_i we set $\alpha_i = e^{\gamma_i}$ and interpret the letters as follows: the upper case letters are both decoded to the identity function $h(x) = x$; this is an "unexpressed" portion of the gene expression representation. The letter f decodes to $f_{\alpha_i}(x)$ and the letter g decodes to $g_{\alpha_i}(x)$.

The choice to use the gene expression technique for evolving data warps rests on the fact that the representation is highly epistatic (see Definition 2.5). The practical effect of using a gene expression representation is that evolution can insert or delete active group elements in a chromosome via mutations that change the upper case/lower case status of a letter. This means that the number of single-mutation neighbours of a given chromosome becomes much larger. This increases the connectivity of the space, making it more compact and, in effect, removing many local optima (the chromosomes that were local optima become objects that can be improved via a single mutation). This compression of the search space compensates, to some degree, for the fragility caused by epistasis. This is similar to the results obtained by using null operations, but has a key difference. When a gene is unexpressed, it behaves the same as a null operation in that instance of an encoding; the key difference is that while a gene is unexpressed, it still retains the solution data of the expressed gene. Therefore, if the gene is turned back on, it will still contain the relationship between the generator function and the real parameter evolved to match with it.

The data warp encoded by a chromosome is the left-to-right composition of all the functions specified. The application of an exponential function to the numerical parameter is deliberate. The identities concerning inverses of the generators, in the next section, show that a balanced exponential interval samples functions and their inverses evenly.

6.2.1 Properties of the Representation Elements

Elementary algebraic manipulation permits the proof of the following identities:

$$f_{\frac{1}{\alpha}}(x) = f_{\alpha}^{-1}(x) \tag{6.5}$$

$$g_{\frac{1}{\alpha}}(x) = g_{\alpha}^{-1}(x) \tag{6.6}$$

$$f_{\alpha}(f_{\beta}(x)) = f_{\alpha\beta}(x) \tag{6.7}$$

$$g_{\alpha}(g_{\beta}(x)) = g_{\alpha\beta}(x) \tag{6.8}$$

Equations 6.5 and 6.6 show why the real parameters are encoded in a range $[-\tau, \tau]$ and then run through the exponential function. The symmetric range mean that each basis function in the representation also has its inverse in the representation, which both satisfies axiom 3 of a subgroup and provides symmetry. Without this encoding trick, the represen-

tation would be biased toward functions that normalize to one side of the interval or that prefer either compression to the middle or compression away from the middle.

Equations 6.7 and 6.8 demonstrate that the two basis functions form, for all values of α, subgroups of $\mathcal{U}$ that are structurally equivalent to the multiplication group of the positive real numbers. The two functions do *not* commute with one another and, for that reason, generate a non-commutative subgroup of $\mathcal{U}$ that encompasses a rich subset of $\mathcal{U}$. This also means that the order of composition of the generators affects which data warp they specify.

The role of τ is one that interacts with the length of the representation to permit control over how finely the represented elements of $\mathcal{U}$ are spaced out from one another. The larger the value of $|log(\alpha)|$, the greater the curvature of the encoded element with $\alpha > 1$ yielding convex f_α and versions of g_α that move objects to the center. τ thus controls the range of possible space-deformation one element of the representation can encode. When τ is small, one must compose several functions to get an extreme result. The parameter n controls how many functions are available to be composed. Small τ and large n represent fine "spacing" of the representable functions. Large τ and small n represent wide "spacing".

The gene expression representation, treating the capital letters as identity functions that do not transform data at all, provides two potential advantages. First, it permits the representation to select the number of elements of $\mathcal{U}$ actually used to normalize data, up to a maximum of n. Second, the use of a gene expression representation is known to encourage search. The gene expression feature can be disabled with a software switch to permit assessment of its impact.

Example 6.1 *Suppose* $n = 4$ *and that the two linear structures are* $fgFf$ *and* $(0.5, -0.8, 0.9, 0.1)$. *Then the encoded elements are* $f_{1.648}(x)$, $g_{0.741}(x)$, $h(x) = x$, *and* $f_{1.11}(x)$. *The encoded data warp is*

$$w(x) = f_{1.648}(g_{0.741}(h(f_{1.11}(x))))$$

$$= f_{1.648}(g_{0.449}(f_{1.11}(x)))$$

A plot of the encoded function is shown in Figure 6.2.

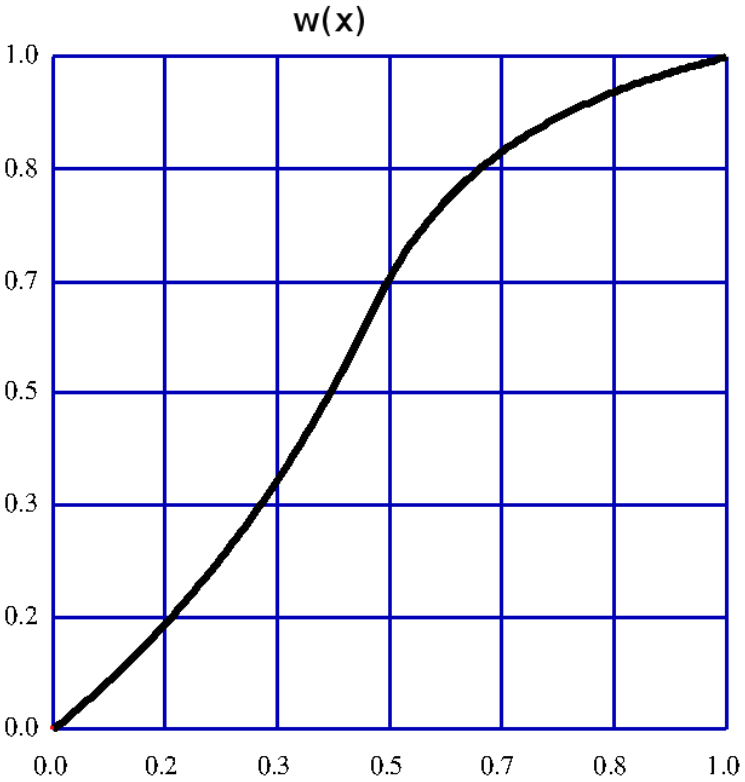


Figure 6.2: An example of the type of data warp that can be encoded with the representation presented in this study.

Another desirable feature of the representation is that, while it approximates the inverse of the CDF of the data, it is easy to retrieve the approximate PDF. If, for a given representation, the real parameters are all negated and the encoded functions composed in the reverse order then the resulting function is the approximate CDF. This follows from Equations 6.5 and 6.6 and group identity $(a \circ b)^{-1} = b^{-1} \circ a^{-1}$. Since the CDF will be coded as the composition of a sequence of generators, each of which has a simple derivative, the PDF arises from an iterated chain rule that will involve computing $n^2 + n$ functions with complexity similar to a single generator. This computation would be performed only once after evolution has completed and only if the PDF is required, making little difference to runtime.

6.3 Experimental Design

The evolutionary algorithm operates on a population of data warps with a default size of 1000. The default gene length is 15 and the gene expression mechanism is used by default. "Default" means that this value is used unless a given experiment specifically states a different value. A single run of the evolutionary algorithm generates an initial population by filling in values uniformly at random. Single tournament selection is used, with a default tournament size of seven. Offspring are produced by copying the parents, performing crossover on the copies, selecting a number of point mutations uniformly at random between one and a specified maximum and performing those mutations. The default maxi-

mum number of mutations is 3.

The choice of one point crossover is made because the representation is highly epistatic (see Definition 2.5). The last (leftmost) function applied has the largest influence on the shape of the normalization function; the first (rightmost) function has the least. The impact of functions more to the right side of the gene is influenced by the functions to the left. This situation, in which the meaning of later alleles is strongly dependent on earlier ones, creates the epistasis. This means that operators with more crossover points will disrupt co-evolved structures, making one point crossover the correct choice for this representation.

The default number of mating events in a run of the evolutionary algorithm is 200,000. The value of $\tau = 1$ was used throughout the experiments, relying on chromosome length to control the granularity of the representation of data warps. Summary fitness statistics are saved periodically, typically so that 100 equally spaced reports are made. All experiments used the fitness function specified by Equation 6.2.

6.3.1 Data Sets

This section describes the data sets used in experiments.

Data Set D_1

This data set is a collection of real values, slightly skewed to the smaller end of its range. It has 30 members with the following values:

$\{$0.35, 0.4, 0.50, 0.60, 0.70, 0.75, 0.8, 0.85, 0.9, 0.95, 1.0, 1.05, 1.1, 1.15, 1.2, 1.25, 1.6, 1.7, 2.0, 2.5, 3.0, 3.5, 4.0, 4.5, 5.0, 5.5, 6.0, 7.0, 8.0, 9.0$\}$

D_1 was used for an initial parameter setting study.

Data Sets $D_2 - D_7$

The next six data sets D_2-D_7 were generated with the statistical package **R** (R Core Team (2013)). The rbeta function was used to generate these data sets. This function is called with three parameters, first the number of data points desired and then the two shape parameters for the beta distribution (Evans et al. (2000)). The parameters used were $rbeta(100, 2, 8)$ for D_2, $rbeta(100, 10, 1)$ for D_3, $rbeta(100, 0.5, 0.5)$ for D_4, $rbeta(1000, 2, 8)$ for D_5, $rbeta(1000, 10, 1)$ for D_6, $rbeta(1000, 0.5, 0.5)$ for D_7. These parameter settings yield several different shapes of distributions, both unimodal and bimodal, with three sets of size 100 and three of size 1000.

6.3.2 Experiments Performed

The first experiment performed looked at the impact of gene length and the use of gene expression. Gene lengths of $n = 6$, 9, 12, 15, 18, and 21 were used with and without gene expression. Data set D_1 was used in this experiment and 120 replicates were used to facilitate comparison.

The second experiment performed a parameter setting study on data sets D_2-D_7, followed by an attempt at a "best" warp by greatly increasing gene length. Population size was selected from 100, 1000, 10000, with the maximum number of mutations being 3 or 5, and gene length being 15 or 45. After these 12 initial runs, the best combination of parameters for each data set was used along with a larger gene length: n =135 for D_2-D_4

and n =90 for D_5-D_7. 10 separate runs of 200000 mating events each were performed for every set of parameters, with gene expression turned on at all times. The decision to use gene expression at all times was motivated by the results of the first experiment.

6.4 Results and Discussion

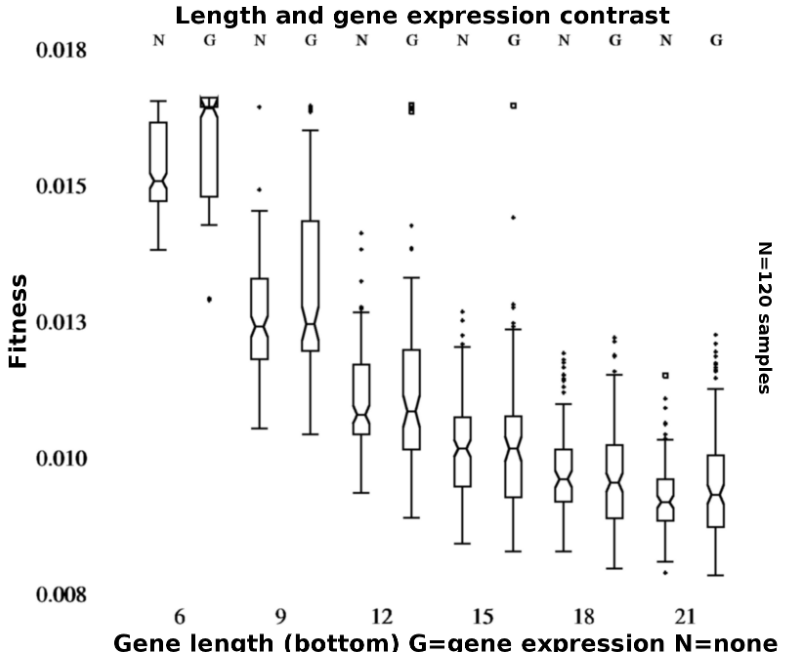


Figure 6.3: Shown are box plots for the parameter study treating length and use of gene expression for test data set D_1.

The results of the experiment testing the impact of gene length and use of gene expression are shown in Figure 6.3. The gene expression representation is significantly worse only for length $n = 6$, for all other lengths the performance difference was not significant. In all cases, however, the best (lowest) value for the objective function was located by one

of the runs with gene expression turned on. This suggests that even the search problem for data set D_1, which is small and possesses a relatively simple structure, favors exploration over exploitation. The poor performance of gene expression for length $n = 6$ is likely due to insufficient gene length. With only 6 alleles available, any unused alleles would likely have had a significant negative effect on the quality of the data warp.

In every case, in the initial length-setting experiments shown in Figure 6.3, lengthening the representation yielded a statistically significant reduction in the objective function, which is a type of error measure. The time required to perform fitness evaluation is directly proportional to the length of the representation and so there is a clear time-quality trade-off.

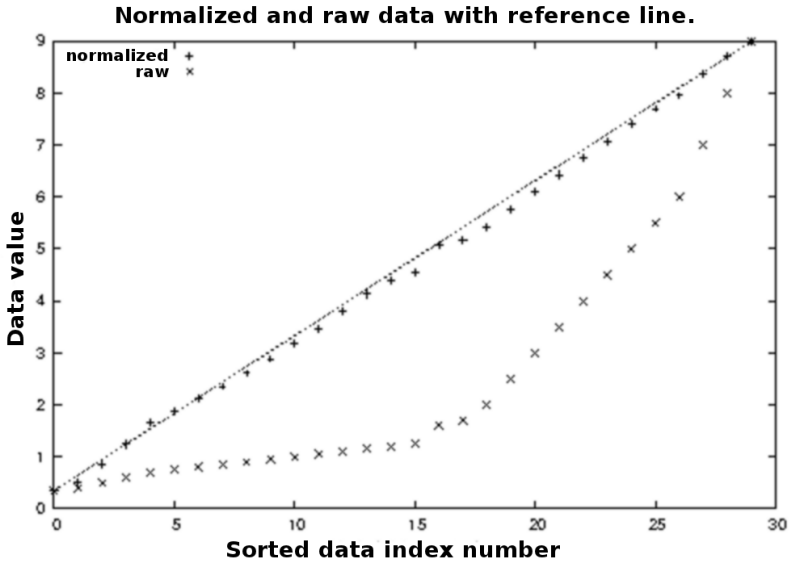


Figure 6.4: This plot shows the data from data set D_1 plotted in its raw form and after normalization by the evolved data warp **1.000g, -0.726f, -0.757g, -0.984g, 0.255f, 0.992f, -0.564g, 0.749f**. The data warp was evolved with length $n = 12$ but four unexpressed loci are not shown.

Figure 6.4 shows a plot of data set D_1 in raw form and after normalization. This figure

shows that the normalized distribution, while not uniform, is quite close to uniform. The data warp used for normalization in this case was the best of ten replicates using the default parameters with gene expression turned on.

The results of the parameter study from the second experiment are displayed in Table 6.1, and Figure 6.5 contains plots of the data sets before and after normalization. Note that while the runs performed with largely increased gene length were intended to provide the best solutions, this was not always the case. This suggests that the trade-off of computation time for quality was surpassed, and the algorithm became overloaded with gene loci. There is a clear preference for smaller population sizes, and the change between 3 or 5 maximum number of mutations seemed to have negligible effect.

Table 6.1: Parameters used to obtain the best fit warp function on data sets D_2-D_7.

Data Set	Population Size	Max. Mutations	Gene Length
D_2	100	3	135
D_3	1000	3	45
D_4	100	5	135
D_5	100	3	90
D_6	100	5	90
D_7	100	5	45

Once the best fit solutions were obtained, the warps were applied to a different data set and the resulting plots were observed. Once such plot appears in Figure 6.6 alongside a Q-Q plot generated from the same data sets. The visualization produced from the data warp tracks very closely to the points specified by the discrete Q-Q plot, but has the obvious benefit of being smooth and continuous across the interval.

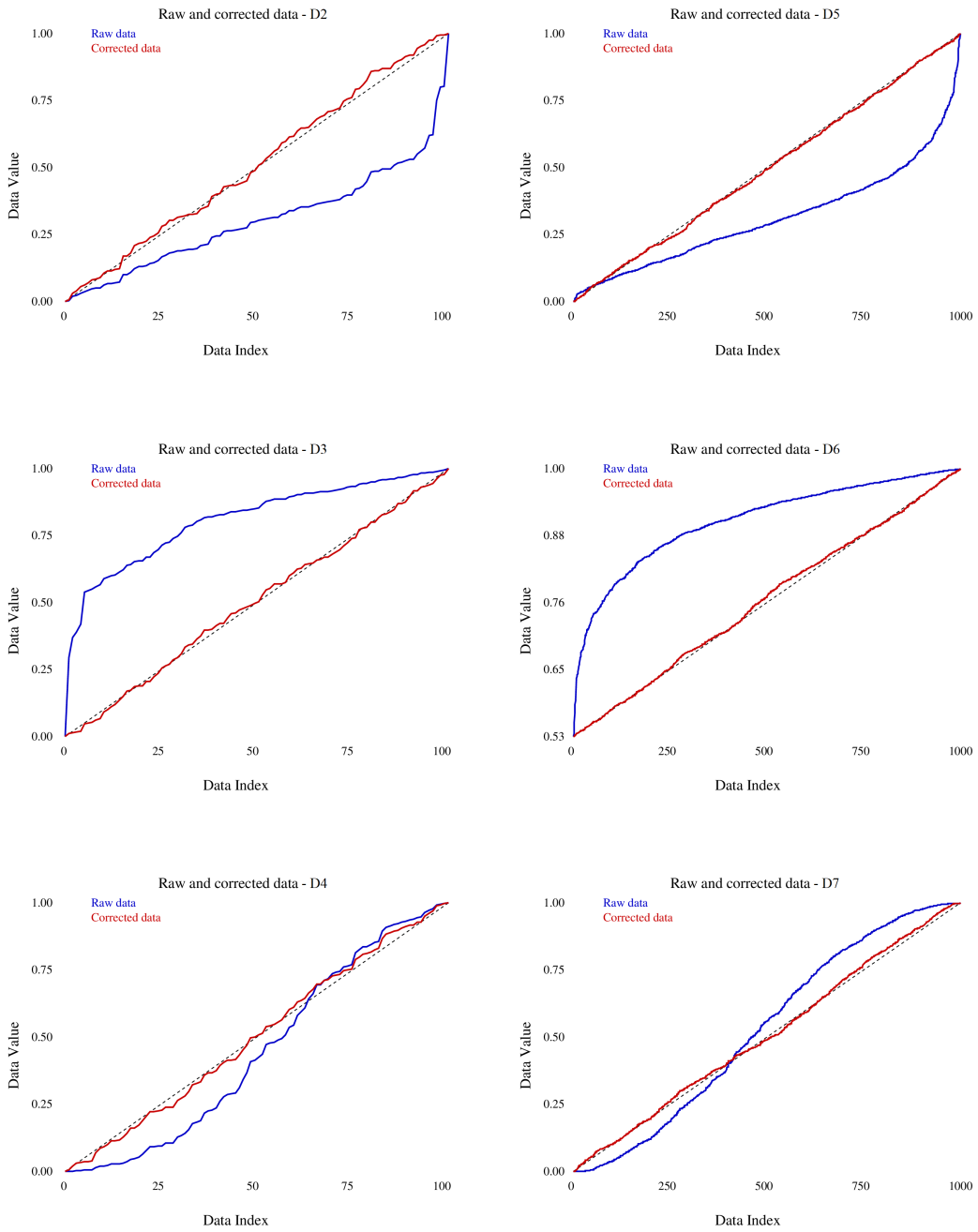


Figure 6.5: Data sets D_2-D_7 before (blue) and after (red) normalization with the best-fit evolved data warps for the respective sets.

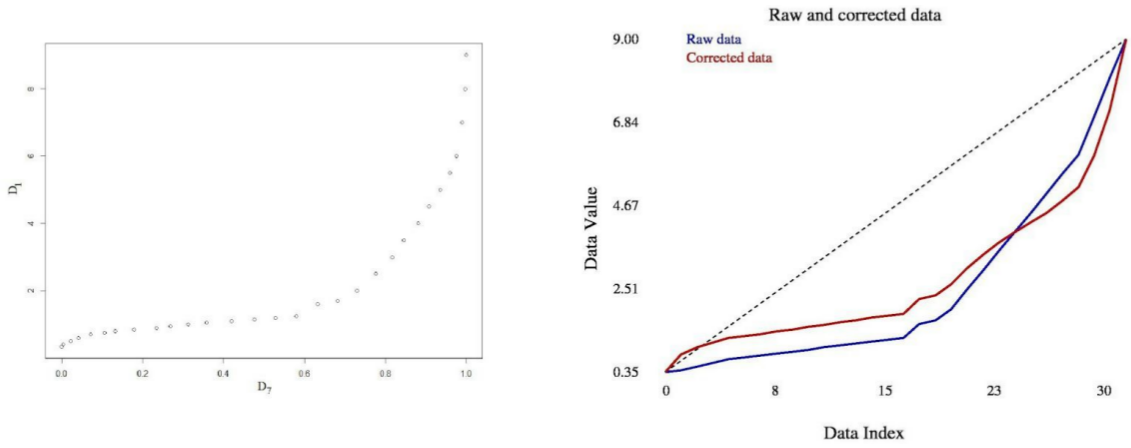


Figure 6.6: Data sets D_1 and D_7 being compared by Q-Q plot (left) and by data warps (right, red line).

6.5 Conclusions and Next Steps

The representation presented here has been shown to be able to effectively find a normalizer that approximates the inverse of the CDF of both unimodal and bimodal data sets. It has also been shown that this representation is capable of producing a form of data comparison which is similar to, and an improvement on, Q-Q plots. An initial parameter study showed that longer genes generally obtain better results - but this is not surprising as these genes encode more fitting parameters. Further study found eventual diminishing returns, although the relationship between gene length and values of τ has yet to be examined. The gene expression technique, while it did not yield a large impact on average performance, was used when finding all the best results. It may also have a stronger impact when using larger gene lengths, when it is less important for every gene loci to be expressed. This suggests it may be worth retaining in future experiments.

6.5.1 Relationship to Genetic Programming

With gene expression engaged, the representation used to evolve data warps is searching a space of between 0 and n compositions of parametrized bijections of the unit interval for those that space out a data set as evenly as possible. Given that we are evolving a function by choosing both real parameters and one of two possible functional forms which could be regarded as unary operators, the representation presented here might be regarded as a type of genetic programming (Koza (1992, 1994); Banzhaf et al. (1998)). It is, however, a linear representation with a fixed maximum length.

Since one goal of genetic programming is to evolve special-purpose functions, it might be profitable to define a new type of genetic programming called *unary genetic programming* of which the space warps are an example. Any group-based representation where the generators are functions from a set to itself could be placed in this category, creating a type of genetic programming that is far neater and with simpler inheritance rules than the original sort.

The evolvable data warp representation might alsohttps://www.overleaf.com/project/60f5c527c12f3 be profitably *incorporated* into the existing field of genetic programming. Tunably biased data warps would form a class of unary operators that could be chosen for their problem-specific character. In addition, the individual functions f_α and g_α are potentially valuable additions to the library of functions used in genetic programming. With proper normalization, they can make small modifications to the range or domain of other functions.

6.5.2 Target Compilation of Final Results

Once a data warp is evolved, identities 6.7 and 6.8 permit substantial automatic simplification of its form. In particular, blocks of f or g functions can be condensed into a single f or g function and unexpressed loci can be deleted. In addition, after this simplification it is almost trivial to automatically generate compilable code, e.g. a C++ or Java function, that encodes the computations performed by the data warp. If data warps are to be used in other code, this form of simplification is potentially quite valuable and can be incorporated into the class for the data warp representation. In applications, the creation of target-compiled source for a particular data warp would also yield substantial speed advantages.

6.5.3 An Alternate Use for Bijections of the Unit Interval

In this study, data warps are used to re-parametrize a data set. Imagine, instead, that a data warp is applied to the independent variables of an rectangular optimization domain. Then the warps would re-parametrize the search space. An example of such a re-parametrization in one dimension is shown in Figure 6.7. Having a representation that encoded point coordinates and their associated warps would permit an optimization strategy in which members of the evolving population re-parametrized the space dynamically. Those that discovered data warps that make difficult-to-find global optima larger would gain a selective advantage.

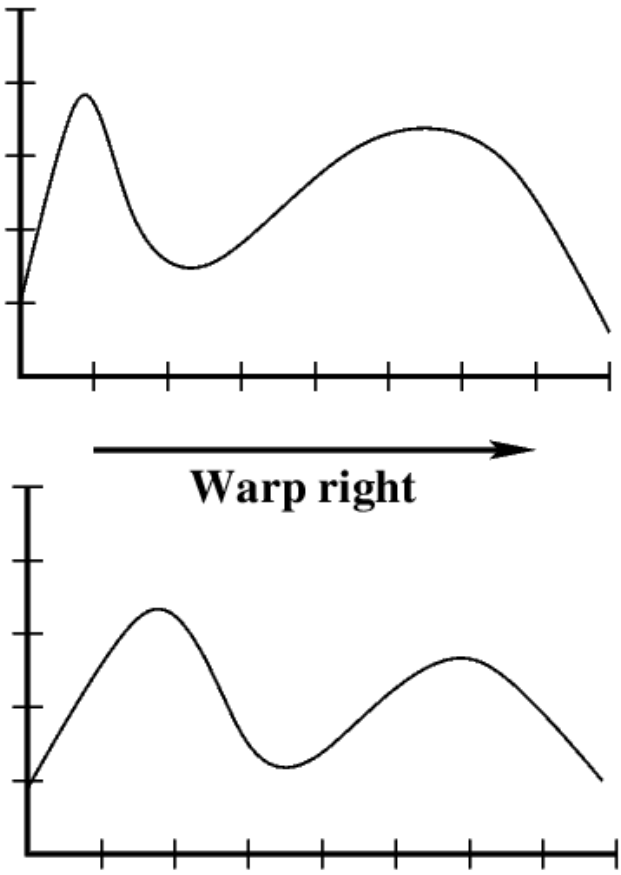


Figure 6.7: An example of using a data warp to modify a fitness landscape. Notice that the warp chosen enlarged the basin of attraction of the global optimum.

Chapter 7

Conclusions and Next Steps

This thesis explored a novel style of representing a problem to a computer. It introduced the idea of using generators of a subgroup to build representations, and paired them with evolutionary computation to test their efficacy. The restriction to this particular application was in an attempt to manage the size of the thesis, as it quickly became apparent that this new area of research has an enormous number of possibilities. The thesis then evolved from simply trying something new to being a guiding first look into using group theory to construct representations for computational intelligence. Any one study contained herein had sufficient additional areas to explore that they could have been expanded and examined in greater detail to become a large research project in their own right.

However, the goal was to show that using group theory to construct representations for evolutionary computation is a good idea in general, and can likely be easily extended to other families of algorithms, rather than simply detailing one instance where it worked.

The result of the pursuit of this goal is a thesis which includes a variety of groups used to solve problems in very different areas, from basic additive vector groups to bijections of simplices in $\mathbb{R}^n$, and from real optimization to evolved art, most of which has appeared in peer-reviewed publications (listed in Chapter 1). This all contributed to further exploring the representation problem in evolutionary computation, and opened a door to a very large realm of new possible representations to provide even more opportunities to study the effects of different representations on algorithm behaviour.

It is my hope that this initial foray into group-based representations for computational intelligence is not simply the beginning of potentially years of interesting topics of study for myself, but also an illuminating look into the possibilities for others to construct their own representations in a similar manner. The possibilities in this area of research seem truly endless, and I am sincerely excited to see what can be produced when more creative minds turn their focus in this direction.

7.1 Future Work

The individual studies contained in this thesis include some ideas on future work that may arise from that particular project, such as modifying the WUC representation used in Chapter 3 to be a restarting-recentering algorithm, or the possibility of using the data warps from Chapter 6 to re-parameterize a given search domain. There is an entire area of research examining structured algorithmic initialization, using point packings, related to

Chapter 5 (Ashlock and Graether (2016); Stoodley et al. (2018)). However, there are areas of potential research which arise from this group of projects as a whole.

7.1.1 Exploring Different WTRs

The first instance of a WTR in this thesis was used in Chapter 3 and is referred to as the WUC representation (detailed in Section 3.2). In theory, there are an infinite number of bijections of the set of simplices in $\mathbb{R}^n$, any of which could be used as an operation in a representation. In the study introducing a representation using bijections of simplices (Ashlock et al. (2013)), the uncenter operation was not used. Instead, operations that scale the sides lengths of all the sides of a simplex incident on a particular vertex were used, referred to as "grow" and "shrink". It is likely that particular bijections will be useful in "designer" representations.

A natural example of a variation is a representation that used only the center operation, which is referred to as the CSR and is explored in Chapter 4. This representation completely loses the exploratory power of the WUC representation, as objects of interest must lie within the initial simplex, but is highly exploitative. This type of representation could also be hybridized with the original WUC representation by requiring a suffix of the gene to consist only of centering operations, preceded by any operation available. This would allow the WUC to behave normally for some time before forcing a terminal exploitative search.

7.1.2 Exploiting Epistasis

Epistasis is defined in Definition 2.5 in Section 2.2.2. As noted in the justification of the use of one point crossover, the group elements farther right in an instance of an epistatic representation often have diminished effect on the resulting solution. This means that one can automatically decompose any search problem. Start with a relatively short words over the group, yielding a small search space. When the population loses diversity, make the first several loci of the currently best structure in the population mandatory for all population members and evolve only suffixes to this initial segment. If the length of the portion of the structure being evolved is held constant, the search becomes focused in the portion of space represented by the initial fixed segment, and searches that subset at higher resolution.

This scheme represents a managed transition from exploration to exploitation and would permit the successive search of a chain of smaller spaces in place of a large space (full length from the beginning). The expectation is a reduction in the number of fitness evaluations required as well as an increase in final quality.

7.1.3 Shaping Group-based Representations

Definition 7.1 *In evolutionary computation, a* **shape** *is any restriction on how a representation may be instantiated.* **Shaping** *is the practice of introducing such restrictions upon a representation.*

Sections 7.1.1 and 7.1.2 include the idea of shaping to an extent, but there are many

ways to experiment with shapes alone. For example, in any of these group based representations, a very simple restriction would be to avoid placing an element and its inverse beside one another as this wastes space. A similar example, in Chapter 6, when space warp generators of the same type (either f or g) are encoded in succession, their real parameters are simply multiplied yielding a single instance of the generator (see Identities 6.7 and 6.8). Since one generator could essentially do the same job as two generators encoded in succession, a restriction on having repeats of the same generator in succession could again prevent wasted space. In this case, however, the restriction placed on the real parameter they are encoded with may need to be simultaneously loosened, as the product of the real parameters which results from having two generators encoded in succession can potentially yield a single real parameter which is outside the range of this initial restriction. Loosening this restriction would then be necessary for a single generator to replicate the behaviour of two. Therefore, it is obvious that shaping can be helpful in avoiding waste in a representation, but must be applied carefully.

In the past, shaping was used in (Ashlock and Ashlock (2014); Ashlock et al. (2014)) on finite state automata used as game playing agents for Iterated Prisoner's Dilemma. Both studies used eight predetermined shapes during co-evolution of the game playing agents to asses their changes in behaviour. In the latter, the shapes were specifically examined for their ability to produce exploitative strategies. In both cases, the shaping of the game playing agents had distinct effects on the behaviour and strategies implemented by those agents, and some entirely novel strategies were observed.

Shaping was also used in (McEachern and Ashlock (2014)) on side effect machines for DNA classification. For side effect machines, shapes consisted of static transition diagrams with evolution being allowed to change the labels. In this study, the authors again designed their own shapes for use during evolution. However, they also explored the idea of taking the shape of an evolved, high fitness side effect machine, and allowing evolution to produce new side effect machines using that shape. Using the transition structure of a high fitness side effect machine but permitting evolution to change the labels, and further restrict the transitions, yielded substantially more fit side effect machines than running evolution for a longer time with a random initial population. The discovered restrictions on transitions drawn from evolution were better than even the designed restrictions. These techniques could also be beneficially applied to WTRs.

The WUC representation operations come in inverse pairs and there are operations which do not stack in any way, they just cancel one another out. This gives a very simple shape: restrict genes to avoid having adjacent operations that are inverses of one another, as is mentioned above. Much more powerful shaping is possible. As an example, suppose that the gene is restricted to have all its uncenter operations before all of its walk operations before all of its center operations. This would have the effect of making the simplex grow – if it is going to – then walk at its maximum size, and finally home in on its final value. The shape means that the WUC actions in such a shape restriction have three phases with fairly clear meanings for a human observer. The uncenter-walk-center shape is one possible shape, and one that has a clear interpretation, but other shapes may work better. Requiring

at least half the actions be centering actions (and assigning bad fitness to chromosomes that violate this restriction) is a different type of shape that may also be profitable for the WUC representation.

Of course, any group-based representation may be shaped by restricting the pattern of use of the generators, making the space warp and WUC representation shapes examples of an initial proposal of a much broader set of possibilities.

7.2 Conclusion

Beyond these specific examples of areas of potential new research related to topics covered in this thesis, there is also the much broader scope of constructing new representations based on generators of a group. A strong attempt was made in this thesis to showcase the wide range of possibilities for new representations. Entirely new groups can be used to generate entirely new representations to be used on entirely new problems, all completely disjoint from the topics discussed herein. So while this thesis attempts to convince any reader of the efficacy of the few different techniques developed and discussed, it also attempts to convince any reader and indeed any researcher in this area that the possibilities for new, efficient, and effective representations which arise from group theory are limited only by one's imagination.

References

B. Addis, M. Locatelli, and F. Schoen. Disk packing in a square: a new global optimization approach. *INFORMS Journal on Computing*, 20(4), 2008.

D. Ashlock. *Evolutionary Computation for Opimization and Modeling*. Springer, New York, 2006.

D. Ashlock and W. Ashlock. Impact of regulatory genes on optimization behavior. In *Proceedings of the 2012 Congress on Evolutionary Computation*, pages 3372–3379, 2012.

D. Ashlock and W. Ashlock. *A Course in Evolutionary Computation*. Ashlock and McGuinness Consulting, Guelph, ON, 2021.

D. Ashlock and J. Gilbert. A discrete representation for real optimization with unique search properties. In *Proceedings of the IEEE Symposium on the Foundations of Computational Intelligence*, pages 54–61, 2014.

D. Ashlock and J. Gilbert. A greedy, generative, lattice representation for point packing. In *Proceedings of the 2019 IEEE Congress on Evolutionary Computation*, pages 3181–3188, 2019.

D. Ashlock and S. Graether. Conway crossover to create hyperdimensional point packings, with applications. In *Proceedings of the 2016 Congress on Evolutionary Computation*, pages 1570–1577, Piscataway, NJ, 2016. IEEE Press.

D. Ashlock and F. Jafargholi. Evolving extremal epidemic networks. In *Proceedings of the 2007 IEEE Symposium on Computational Intelligence in Bioinformatics and Computational Biology*, pages 338–345, Piscataway NJ, 2007. IEEE Press.

D. Ashlock and B. Jamieson. Evolutionary exploration of generalized Julia sets. In *Proceedings of the 2007 IEEE Symposium on Computational Intelligence in Signal Processing*, pages 163–170, Piscataway NJ, 2007. IEEE Press.

D. Ashlock and B. Jamieson. Evolutionary computation to search mandelbrot sets for aesthetic images. *Journal of Mathematics and Art*, 3(1):147–158, 2008.

D. Ashlock and J. Montgomery. An adaptive generative representation for evolutionary computation. In *Proceedings of the IEEE 2016 Congress on Evolutionary Computation*, pages 1578–1585, 2016.

D. Ashlock and J. Tsang. Evolving fractal art with a directed acyclic graph genetic programming representation. In *Proceedings of the 2015 Congress on Evolutionary Computation*, pages 2137–2144, Piscataway NJ, 2015. IEEE Press.

D. Ashlock, E.Y. Kim, and N. Leahy. Understanding representational sensitivity in the iterated prisoner's dilemma with fingerprints. *Transactions on Systems, Man, and Cybernetics–Part C: Applications and Reviews*, 4(36):464–475, 2006.

D. Ashlock, K.M. Bryden, and S. Gent. Multiscale feature location with a fractal representation. In *Intelligent Engineering Systems Through Artificial Neural Networks*, volume 19, pages 173–180, 2009.

D. Ashlock, E. Shiller, and C. Lee. Comparison of evolved epidemic networks with diffusion characters. In *Proceedings of IEEE Congress on Evolutionary Computation*, pages 781–788, 2011.

D. Ashlock, N. Krisk, and G. Fogel. Functions for the analysis of exploration and exploitation. In *Proceedings of the 2013 Congress on Evolutionary Computation*, pages 2020–2027, 2013.

D. Ashlock, P. Hingston, and C. McGuinness. Evolving point packing in the plane. Accepted to Proceedings of the Australasian Conference on Artificial Live and Computational Intelligence, 2015.

D. Ashlock, S. Gillis, A. McEachern, and J. Tsang. The do what's possible representation. In *Proceedings of the IEEE 2016 Congress on Evolutionary Computation*, pages 1586–1593, 2016.

W. Ashlock and D. Ashlock. Shaped prisoner's dilemma automata. In *Proceedings of the 2014 IEEE Conference on Computatiational Inteligence in Games*, pages 76–83, 2014.

W. Ashlock, J. Tsang, and D. Ashlock. The evolution of exploitation. In *Proceedings of the 2014 IEEE Symposium on Foundations of Computational Intelligence*, 2014.

W. Banzhaf, P. Nordin, R. E. Keller, and F. D. Francone. *Genetic Programming : An Introduction*. Morgan Kaufmann, San Francisco, 1998.

M. Barnsley. *Fractals Everywhere*. Academic Press, 1988.

Ernesto Benini and Andrea Toffolo. Development of high-performance airfoils for axial flow compressors using evolutionary computation. *Journal of Propulsion and Power*, 18 (3):544, June 2002.

G. D. Birkhoff. Proof of the ergodic theorem. *PNAS*, 12(17):656–660, 1931.

É. Borel. Les probabilités dénombrables et leurs applications arithmétiques. *Rendiconti del Circolo Matematico di Palermo*, 27:247–271, 1909.

Markus Brameier. *On Linear Genetic Programming*. PhD thesis, University of Dortmund, 2004.

K. M. Bryden, D. A. Ashlock, S. Corns, and S. J. Willson. Graph based evolutionary algorithms. *IEEE Transactions on Evolutionary Computation*, 10:550–567, 2006.

Edward B. Burger and Robert Tubbs. *Making Transcendence Transparent: An Intuitive Approach to Classical Transcendental Number Theory*. Springer, New York, NY, 2004.

E. K. Burke, G. Kendall, and G. Whitwell. A new placement heuristic for the orthogonal stock-cutting problem. *Operations Research*, 52(4):655–671, 2004.

Thomas Bäck and Hans-Paul Schwefel. An overview of evolutionary algorithms for parameter optimization. *Evolutionary Computation*, 1(1):1–23, 1993. doi: 10.1162/evco.1993.1.1.1.

Hai-Chau Chang and Lih-Chung Wang. A simple proof of Thue's theorem on circle packing. *arXiv: Metric Geometry*, 2010.

Chris Christensen. Polish mathematicians finding patterns in Enigma messages. *Mathematics Magazine*, 80(4):247–273, 2007.

J. H. Conway and D. A. Smith. *On Quaternions and Octonians*. A. K. Peters Ltd., Natick, MA, 2003.

H. T. Croft, K. J. Falconer, and R. K. Guy. *Unsolved Problems in Geometry*. Springer-Verlag, New York, 1991.

S. Das and P.N. Suganthan. Differential evolution: A survey of the state-of-the-art. *Evolutionary Computation, IEEE Transactions on*, 15(1):4–31, Feb 2011. ISSN 1089-778X. doi: 10.1109/TEVC.2010.2059031.

Edwin D. de Jong, Dirk Thierens, and Richard A. Watson. Hierarchical genetic algorithms. In Xin Yao, Edmund K. Burke, José A. Lozano, Jim Smith, Juan Julián Merelo-Guervós, John A. Bullinaria, Jonathan E. Rowe, Peter Tiňo, Ata Kabán, and Hans-Paul Schwefel, editors, *Parallel Problem Solving from Nature - PPSN VIII*, pages 232–241, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg. ISBN 978-3-540-30217-9.

K. Deb, A. Anand, and D. Joshi. A computationally efficient evolutionary algorithm for real-parameter optimization. *Evolutionary Computation*, 4(10):371–395, 2002.

M. Evans, N. Hastings, and B Peacock. *Statistical Distributions*. Wiley, N.Y. New York, 2000.

Günter Ewald. *Geometry: An Introduction*. Thomson Wadsworth, Belmont, CA, 1971.

D B Fogel, E C Wasson 3rd, E M Boughton, and V W Porto. Evolving artificial neural networks for screening features from mammograms. *Artificial Intelligence in Medicine*, 14(3):317–326, November 1998. doi: 10.1016/s0933-3657(98)00040-2.

J. Gilbert and D. Ashlock. Evolvable warps for data normalization. In *Proceedings of the IEEE 2016 Congress on Evolutionary Computation*, pages 1562–1569, 2016.

F. Glover. Future paths for integer programming and links to artificial intelligence. *Computers & Operations Research*, 13(5):533–549, 1986.

P. Grzegorek, J. Januszewski, and L. Zielonka. Efficient 1-space bounded hypercube packing algorithm. *Algorithmica*, 82:3216–3249, 2020.

P. F. Hingston, L. C. Barone, and Z. Michalewicz. *Design by Evolution*. Springer, New York, NY, 2008.

G. S. Hornsby, H. Lipson, and J. B. Pollack. Evolution of generative design systems for modular physicalrobots. *IEEE Transactions on Robotics and Automation*, 19(4):703–719, 2003.

J. A. Hughes, S. Houghten, and D. Ashlock. Recentering and restarting a genetic algorithm using a generative representation for an ordered gene problem. *International journal of hybrid intelligent systems*, 11(4):257–271, 2014.

T. W. Hungerford. *Algebra*. Springer-Verlag, New York, 1974.

Gaston Julia. Mémoire sur l'iteration des fonctions rationnelles. *Journal de Mathématiques Pures et Appliquées*, 8:47–245, 1918.

Dervis Karaboga, Beyza Gorkemli, Celal Ozturk, and Nurhan Karaboga. A comprehensive survey: artificial bee colony (abc) algorithm and applications. *Artificial Intelligence Review*, 42(1):21–57, Jun 2014. ISSN 1573-7462. doi: 10.1007/s10462-012-9328-0. URL `https://doi.org/10.1007/s10462-012-9328-0`.

Davar Khoshnevisan. Normal numbers are normal. *Clay Mathematics Institute Annual Report*, 15:27–31, 2006.

E. Y. Kim and D. Ashlock. *On the design of game playing agents*. Morgan and Claypool, San Rafael, CA, 2017.

John R. Koza. *Genetic Programming*. The MIT Press, Cambridge, MA, 1992.

John R. Koza. *Genetic Programming II*. The MIT Press, Cambridge, MA, 1994.

K. Kunanusont, R. D. Gaina, J. Liu, D. Perez-Liebana, and S. M. Lucas. The n-tuple bandit evolutionary algorithm for automatic game improvement. In *2017 IEEE Congress on Evolutionary Computation (CEC)*, pages 2201–2208, 2017.

A. Linhares. Synthesizing a predatory search strategy for VLSI layouts. *IEEE Transactions on Evolutionary Computation*, 3:147–152, July 1999. doi: 10.1109/4235.771168.

B. Mandelbrot. *The Fractal Geometry of Nature*. W. H. Freeman and Sons, 1983.

A. McEachern and D. Ashlock. Shape control of side effect machines for dna classification. In *Proceedings of the 2014 IEEE Conference on Computational Intelligence in Bioinformatics and Computational Biology*, pages 1–8, 2014.

Silja Meyer-Nieberg and Hans-Georg Beyer. Self-adaptation in evolutionary algorithms. In *Parameter Setting in Evolutionary Algorithm*, pages 47–76. Springer, 2006.

B. Chandra Mohan and R. Baskaran. A survey: Ant colony optimization based recent research and implementation on several engineering domain. *Expert Systems with Applications*, 39(4):4618 – 4627, 2012. ISSN 0957-4174. doi: https://doi.org/10.1016/j.eswa.2011.09.076. URL http://www.sciencedirect.com/science/article/pii/S0957417411013996.

J. A. Nelder and R. Mead. A simplex method for function minimization. *Computer Journal*, 7(4):308–313, 1965.

Ferrante Neri and Ville Tirronen. Recent advances in differential evolution: A survey and experimental analysis. *Artif. Intell. Rev.*, 33(1-2):61–106, February 2010. ISSN 0269-2821. doi: 10.1007/s10462-009-9137-2. URL http://dx.doi.org/10.1007/s10462-009-9137-2.

W. H. Press, S. A. Vetterling, and W. T. Flanney. *Numerical Recipes: The Art of Scientific Computing (3rd ed.)*. Cambridge University Press, Cambridge, MA, 2007.

R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2013. URL http://www.R-project.org/.

Shahryar Rahnamayan, Hamid R. Tizhoosh, and Magdy M.A. Salama. A novel population initialization method for accelerating evolutionary algorithms. *Computers and Mathematics with Applications*, 53(10):1605 – 1614, 2007. ISSN 0898-1221. doi: https://doi.org/10.1016/j.camwa.2006.07.013. URL http://www.sciencedirect.com/science/article/pii/S0898122107001344.

Marian Rejewski. Mathematical solution of the Enigma cipher. *Cryptologia*, 6(1):1–18, 1982.

Claude E. Shannon. A mathematical theory of communication. *Bell System Technical Journal*, 27(3):379–423, July 1948. doi: 10.1002/j.1538-7305.1948.tb01338.

O. M. Shir. Niching in evolutionary algorithms. In G. Rozenberg, T. Bäck, and J.N. Kok, editors, *Handbook of Natural Computing*, pages 1035–1069. Springer, Berlin, Heidelberg, 2012.

M. Stoodley, D. Ashlock, and M. Graether. Data driven point packing for fast clustering. In *Proceedings of the 2018 IEEE Conference on Computational Intelligence in Bioiformatics and Computational Biology*, pages 1–8, 2018.

H. Terashima-Marín, E. J. Flores-Álvarez, and P. Ross. Hyper-heuristics and classifier systems for solving 2d-regular cutting stock problems. In *GECCO '05 Proceedings of the 2005 conference on Genetic and evolutionary computation conference*, pages 637–643, New York, 2005. ACM.

M. B. Wilk and R. Gnanadesikan. Probability plotting methods for the analysis of data. *Biometrika*, 55(1):1–17, 1968.

D. H. Wolpert and W. G. Macready. No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, 1(1):67–82, 1997.