METHODS FOR REDUCING SEARCH AND EVALUATING FITNESS FUNCTIONS IN GENETIC ALGORITHMS FOR THE SNAKE-IN-THE-BOX PROBLEM

by

JOSHUA DEEN GRIFFIN

(Under the Direction of Walter D. Potter)

ABSTRACT

This thesis explores the use of Genetic Algorithms in approaching the Snake-In-The-Box problem in dimension 8. It discusses methods for improving the solutions found by reducing the representation space for the problem. It presents a representation scheme called Frequency-Based Transition Reassignment (FBTR), which creates a unified interpretation of individuals to prune the search space. FBTR is compared to a standard transition-based representation to determine its effectiveness. It is also compared to a canonical representation that was presented by K. J. Kochut in 1996 which is a different technique meant to reduce the search space. In addition, this thesis introduces the concept of a snake blocker and identifies methods for dealing with snake blockers. These methods are evaluated for their impact on the GA as a whole.

Furthermore, fitness functions are explored in great detail and a variety of components to supplement the length of the longest snake in the chromosome, are suggested and evaluated. These components include tightness, skin density, selective skin density, and target distribution.

INDEX WORDS: Snake-In-The-Box, Genetic Algorithms, Hypercube, Snake Problem, Frequency-Based Transition Reassignment, FBTR, Tightness, Skin Density, Selective Skin Density, SSD, Target Distribution, Snake Blockers, Restricted Random Initialization, RRI, Restricted Heuristic Initialization, RHI, Canonical Representation

METHODS FOR REDUCING SEARCH AND EVALUATING FITNESS FUNCTIONS IN GENETIC ALGORITHMS FOR THE SNAKE-IN-THE-BOX PROBLEM

by

JOSHUA DEEN GRIFFIN

B.S., The University of Georgia, 2003

A Thesis Submitted to the Graduate Faculty of The University of Georgia in Partial Fulfillment of the Requirements for the Degree

MASTER OF SCIENCE

ATHENS, GEORGIA

2009

METHODS FOR REDUCING SEARCH AND EVALUATING FITNESS FUNCTIONS IN GENETIC ALGORITHMS FOR THE SNAKE-IN-THE-BOX PROBLEM

by

JOSHUA DEEN GRIFFIN

Major Professor:	Walter D. Potter
Committee:	Michael A. Covington Daniel M. Everett

Electronic Version Approved:

Maureen Grasso
Dean of the Graduate School
The University of Georgia
December 2009

DEDICATION

This thesis is dedicated to my wife, Rossitza. I never cease to find new ways to be impressed with her. Her shining example of determination and hard work has served as a much needed frame of reference for the completion of this thesis. Her encouragement is invaluable to me in every aspect of my life and I feel truly blessed that she was willing to marry me… twice.

In addition, my parents, Teal and Scott, should share this dedication. None of this would have been possible had they killed me at a young age, as I have no doubt they were tempted to do… many times. I can only hope that this stack of paper will serve as a justification for all of the years of parenting they have had to endure, and those that are still to come. Thank you both for being there to love me and to support me all my life.

ACKNOWLEDGEMENTS

I would like to thank Klaus Gross, a coworker and a friend that has been nice enough to read rough drafts of this thesis to provide a much needed technical perspective on the content. In addition, I would like to thank my wife for being extremely tolerant of my long battle to complete this work. Also, many thanks go to my parents for their support throughout my college career.

TABLE OF CONTENTS

Page

CHAPTER 1

AN INTRODUCTION TO THE SNAKE-IN-THE-BOX PROBLEM

1.1 BACKGROUND

The Snake-in-the-Box problem, referred to here as simply the **snake problem**, was first discussed by W. H. Kautz in 1958. A **snake**, at its most basic level, is a sequence of integers. The snake problem centers on finding the longest possible snakes that can be created from a fixed set of integers. In order for a sequence of integers to qualify as a snake, they must adhere to two rules, which will be discussed in detail later. These two rules that qualify a sequence as a snake make finding long snakes a difficult problem.

As background to the snake problem, it is important to understand how binary numeral representation can be used to represent integers. **Binary numeral representation** is a base-2 system that uses only two digits, 0 and 1, to express numerals. In contrast, numerals in a base-10 system are expressed using ten digits from 0 to 9. **Places** are the ordinal positions which the digits occupy in the numeral, and each place has a place value. **Place values** provide a contextual meaning for the digits in the numeral and increase from right to left. In a base-10 system, place values are powers of ten. For example, the ones place has a place value of 10^0 and the hundreds place has a value of 10^2. The total value for a numeral is the sum of the place value multiplied by the digit for every position in the numeral. For base-10, the total value for a set of digits is determined using the following function:

$$\sum_{place=0}^{n-1} digit_{place} \times 10^{place}$$

Here, *place* is the ordinal position in the numeral, from right to left, $digit_{place}$ is the digit occupying that position, and *n* is the number of places in the numeral. For example, the value of the numeral 105 is $\left(5\times10^{0}+0\times10^{1}+1\times10^{2}\right)=125$. The base-2 system works in the same manner, but the place value at each position is a power of two instead of ten. The function for getting the total value from a base-2 numeral is:

$$\sum_{place=0}^{n-1} digit_{place} \times 2^{place}$$

Applying this function to the binary numeral 1101 yields $\left(1\times2^{0}+0\times2^{1}+1\times2^{2}+1\times2^{3}\right)=13$.

1.2 THE PROBLEM

Snakes were originally suggested as a solution to problems in analog-to-digital systems [Kautz58]. For their utility in real world applications, snakes are an important area of research and have applications in electrical engineering (analog-to-digital conversion), coding theory (data integrity checking), combination locking schemes, the simplification of disjunctive normal form, and computer network topologies (Kautz networks) [Kautz58, Klee67].

In digital applications, numbers are represented in binary numeral representation and each digit is called a bit. In analog-to-digital systems, information must be converted from an analog state to a digital representation so that it can be utilized. An example of this type of system is a linear encoder, shown in Figure 1.1. A linear encoder has a sensor that moves along a reading surface. In an optical linear encoder, the reading surface displays a black and white pattern. When a reading is taken by the sensor, the black portions of the pattern are typically

converted into 1-bits and the white portions into 0-bits. The result is a binary numeral that the digital application uses as an indicator of position. As the sensor moves along the reading surface, the pattern changes, and so does the binary value.

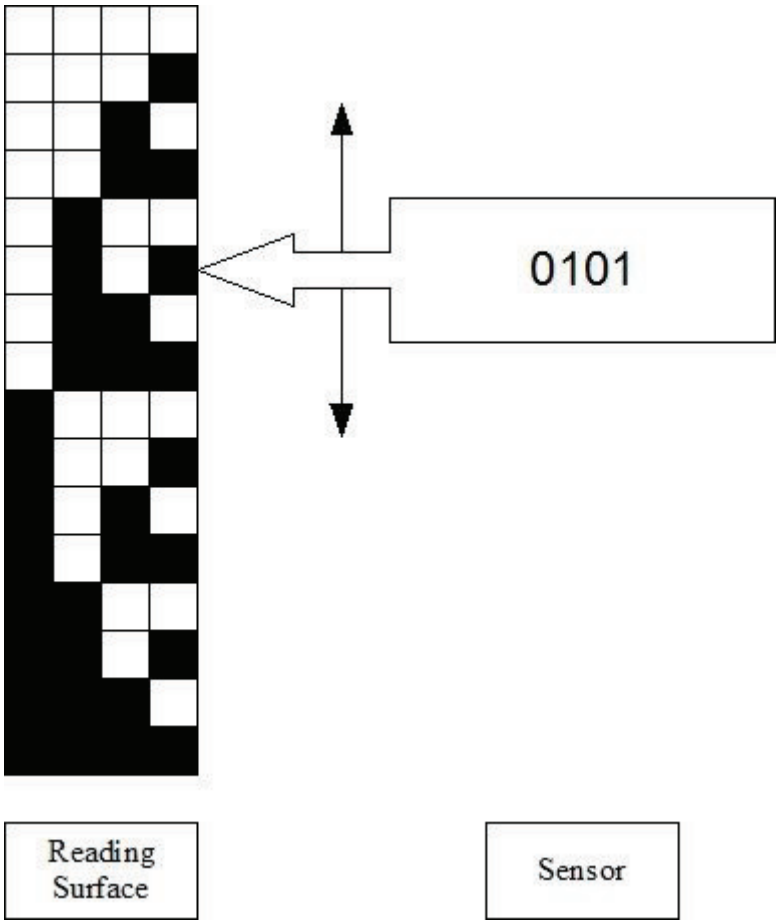

Figure 1.1 – An example of a linear encoder in action. As the sensor moves vertically, in relation to the reading surface, the pattern changes, and so does the resulting binary interpretation.

Unfortunately, reading the encoded value is not a perfect process. If the sensor mistakenly interprets a white square as a black square, or vice versa, the resulting digital number will be incorrect. If the first white square in Figure 1.1 were to be mistakenly interpreted as a black square instead, the resulting digital representation would be 1101. Even though this

reading only differs from the correct reading of 0101 by a single bit, the application will register the position of the reader as being much lower on the reading surface. Because the application has no way to register the error has occurred, it carries on its calculations with the inaccurate reading. To combat this problem, snakes were suggested as a means of detecting this type of error and, possibly, correcting it [Kautz58].

Since read errors of this nature are rare, the goal is to detect an error when only one bit has been misread. The general concept is that read errors should not result in major misinterpretations of the sensor's position. The only way to ensure that the sensor will not accidentally register far from its actual location is to place constraints on the numbers that can be used on the reading surface.

One constraint is that the application should never mistake the sensor's position by more than one encoding location. Therefore, the numbers that are not next to each other on the reading surface must differ by at least two bits. That way, a one bit error will not cause a significant problem. Another constraint is that numbers that are next to each other on the reading surface must only differ by one bit. This constraint allows the application to determine that a read error has occurred. If the current reading differs from the previous reading by more than one bit, then the current reading would be inaccurate. Furthermore, waiting for the next reading may provide enough information to correct the misread value. Figure 1.2 shows a linear encoder where the values used meet these two conditions. Adhering to these constraints makes the linear encoder robust to single bit reading errors. However, the number of values that can be used on the reading surface is greatly diminished.

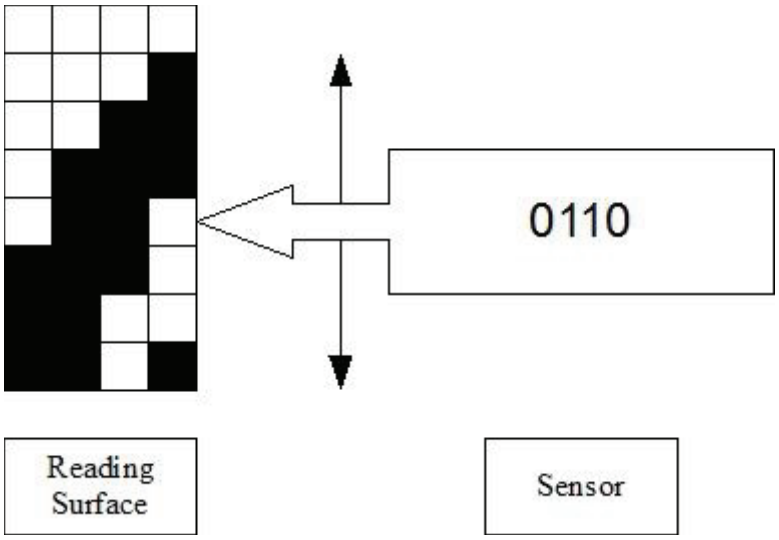

Figure 1.2 – A linear encoder where the reading surface uses only numbers that meet the two constraints described above.

The two constraints outlined in the previous paragraph also define the rules for determining if a sequence of numbers is a snake. As shown with linear encoders, the constraints meant to increase the robustness of an application against incorrect sensor readings also limit size of the encoders reading surface. To keep the reading surface as long as possible, longer snakes are needed. The snake shown above is the longest snake that can be encoded in only four bits. In general, the number of bits used to encode the values can be increased, thereby creating a larger set of numbers to pull from. However, increasing the number of bits also makes finding snakes more difficult. As a result, finding the longest snakes that can be represented in a set number of bits is of great interest.

1.3 HYPERCUBES

A **hypercube** is a tool for conceptualizing how all of the numbers that can be expressed in a fixed number of bits relate to each other. The number of bits is called the **dimension**, and a

hypercube for a given dimension, *d*, is called a *d*-dimensional hypercube. A *d*-dimensional hypercube consists of 2^d vertices, where each **vertex** is a numeral composed of binary digits and $2^d \times \frac{d}{2}$ edges, where an **edge** connects each pair of numerals that differ by only one digit. The number of bits that are different between two binary numerals is referred to as their **Hamming distance**. The vertices of the hypercube that have a Hamming distance of one and, therefore, have an edge between them, are called **neighbors**. Figure 1.3 shows a 1-dimensional hypercube, which has two vertices and one edge. The edge represents the place with a place value of $2^0=1$. When the digit is 0, the total value of the numeral is 0. When the digit here is 1, the total value of the numeral is 1. This edge can also be referred to as **transition 0**, because that indicates the power to which two is raised at that place. A transition simply indicates which place changes between two vertices. A **transition class** is the set of all transitions that have the same place value. There are *d* transition classes in every *d*-dimensional hypercube.

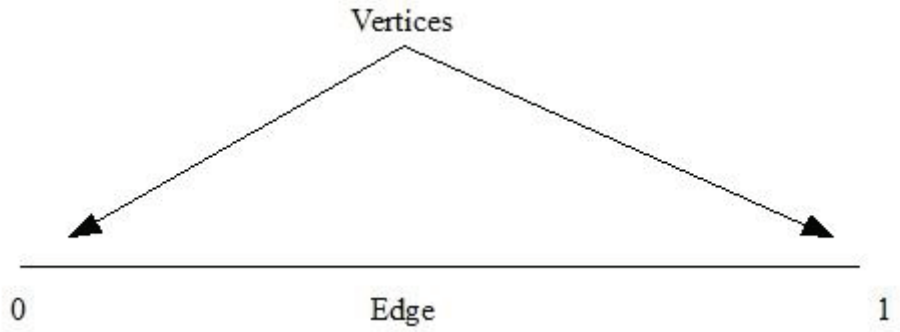

Figure 1.3 – A 1-dimensional hypercube. Only two values can be represented using one bit, 0 and 1.

A 2-dimensional hypercube has four vertices and four edges, as illustrated in Figure 1.4. With each new dimension, the vertices of the hypercube get one new edge and the number of vertices in the hypercube doubles. In addition to transition 0, a 2-dimensional hypercube also

has a transition 1 for the edges where the place value is 2^1. The 2-dimensional hypercube can be thought of as two 1-dimensional hypercubes that are joined together by a new transition. Figure 1.5 shows a 3-dimensional hypercube and Figure 1.6 shows a 4-dimensional hypercube. These figures serve as illustrations of how the number of vertices doubles for each added dimension.

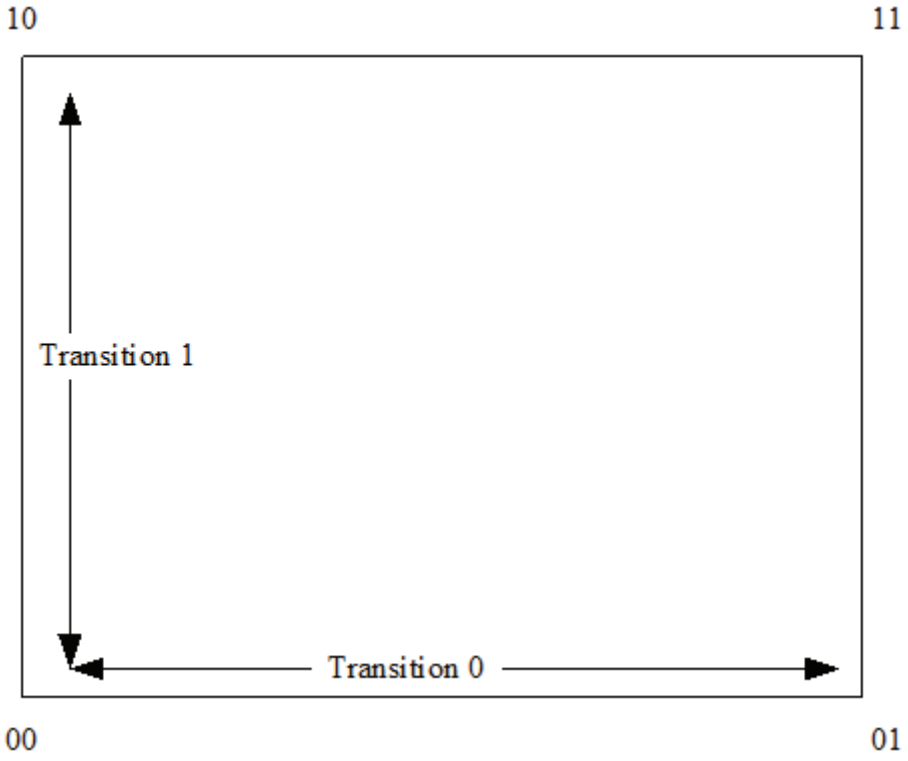

Figure 1.4 – A 2-dimensional hypercube. There are four vertices and four edges in a 2-dimensional hypercube. The horizontal edges represent the transition class for transition 0 and the vertical edges represent the transition class for transition 1.

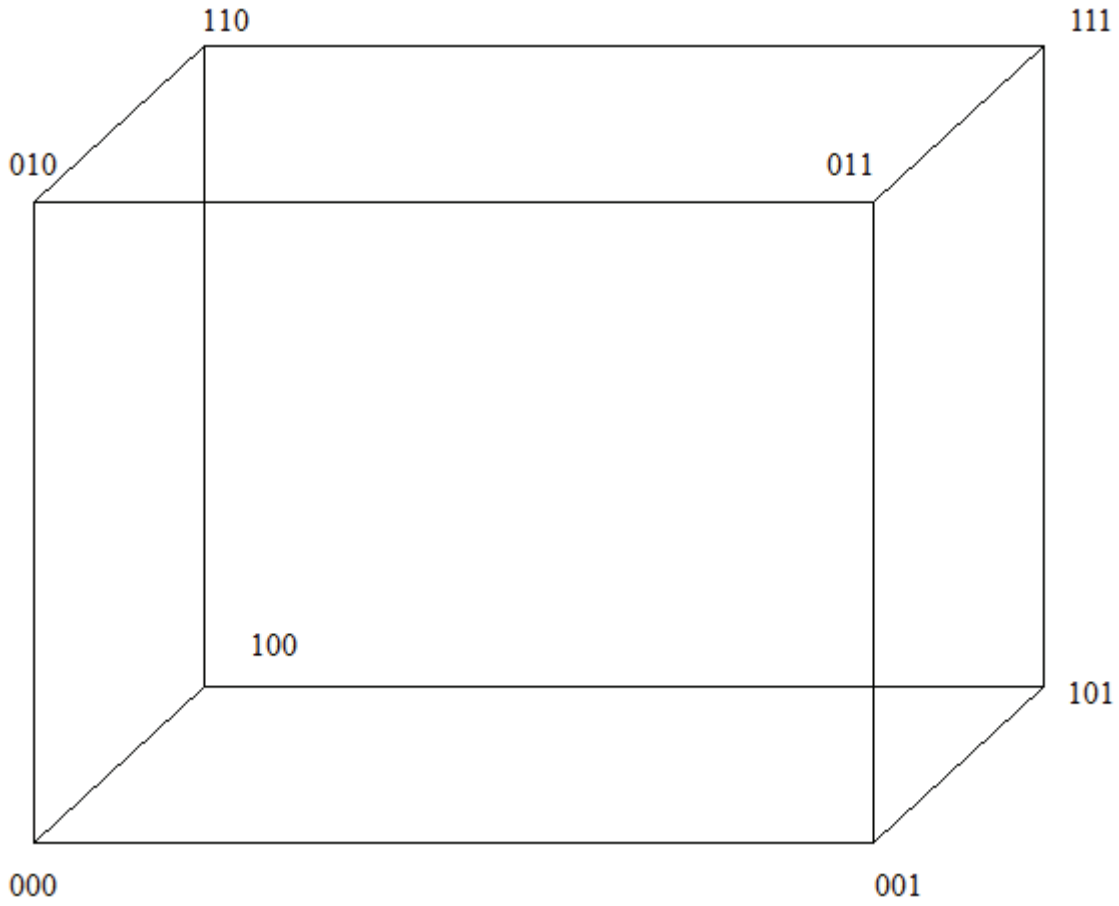

Figure 1.5 – A 3-dimensional hypercube which has eight vertices and twelve edges.

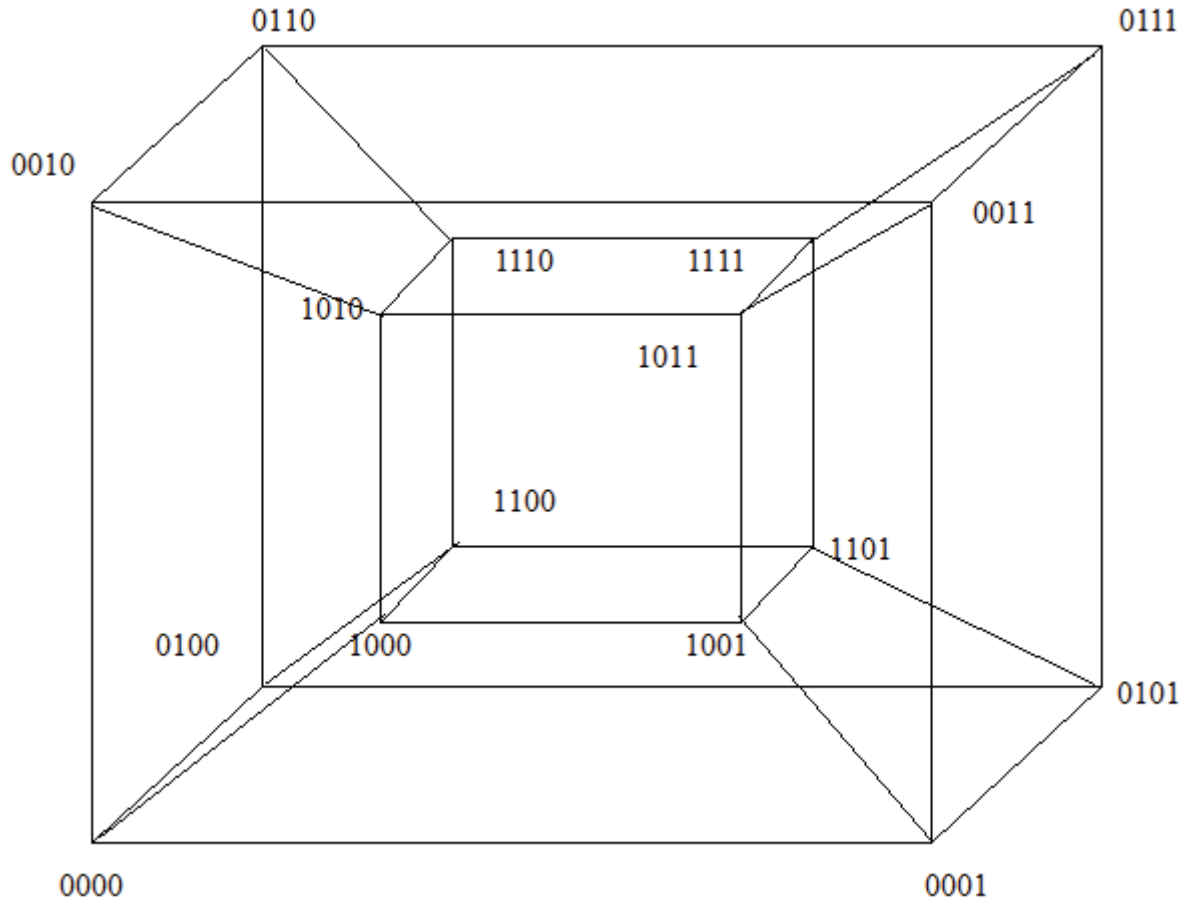

Figure 1.6 – A 4-dimensional hypercube, which has sixteen vertices and thirty-two edges.

1.4 THE SNAKE-IN-THE-BOX

A snake is a sequence of integers. When mapping these integers to their binary representation in a hypercube, a snake becomes a non-cyclical path traversing the edges of the hypercube. Figure 1.7 shows a snake imposed on a 3-dimensional hypercube and illustrates how the hypercube acts as the "box" referenced in the snake problem's name.

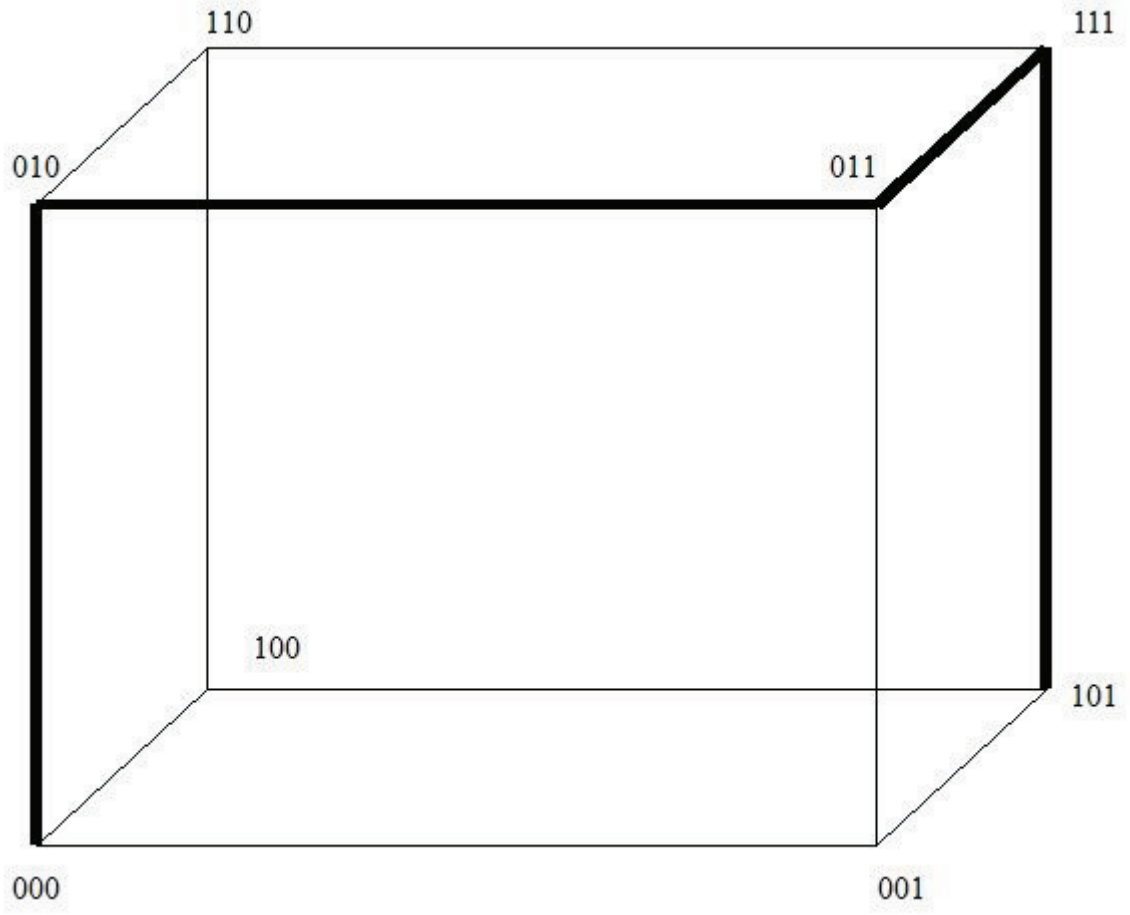

Figure 1.7 – A snake of length 4 in a 3-dimensional hypercube.

The path of a snake can be discussed either in terms of its vertices or its transitions. The snake in Figure 1.7 is comprised of the sequence of vertices 0, 2, 3, 7, and 5. The length of the snake shown here is 4 because the length is measured in terms of the number of transitions, not the number of vertices. For notational convenience, the sequence may be referred to as (*v*0, 2, 3, 7, 5), where the *v* indicates that the numbers are vertices. The **head** of a snake is the vertex at the beginning of the sequence, and the **tail** of the snake is the last vertex of the sequence. The path (*v*0, 2, 3, 7, 5) assumes that the head of the snake is *v*0 and that the tail of the snake is *v*5. However, there is no requirement that *v*0 be the head of the path, and it is perfectly valid to

interpret the snake in reverse as (v5, 7, 3, 2, 0). This also means that (v0, 2, 3, 7, 5) and (v5, 7, 3, 2, 0) are the same snake.

When the path is viewed in terms of transitions, the snake traverses transitions 1, 0, 2, and then 1 again. This will be referred to using the notation (t1, 0, 2, 1), where t indicates that the numbers are transitions instead of vertices. Here again v0 has been assumed as the head of the snake, but it would also be acceptable to start from the other end as (t1, 2, 0, 1). It is worth noting that the transition-based notation does not reference any specific vertices. Traversing a transition is not a directional process; it goes both ways. Therefore, the vertex based path (v0, 2, 3, 7, 5) is merely the result of applying the transition sequence (t1, 0, 2, 1) starting with vertex v0. If a new head is selected, the same transitions can be applied. The result will be a new sequence of vertices, but the transition sequence will remain the same. Figure 1.8 shows the result of applying the same transitions but choosing v6 as the head of the snake, creating the vertex sequence (v6, 4, 5, 1, 3). Therefore, the transition sequence representation is a generalized version of all snakes which follow the same sequence of transitions. Since there are 2^d vertices in a hypercube for dimension d, there can be 2^d valid heads for every transition sequence, making the transition based representation an expression for many different sequences of vertices.

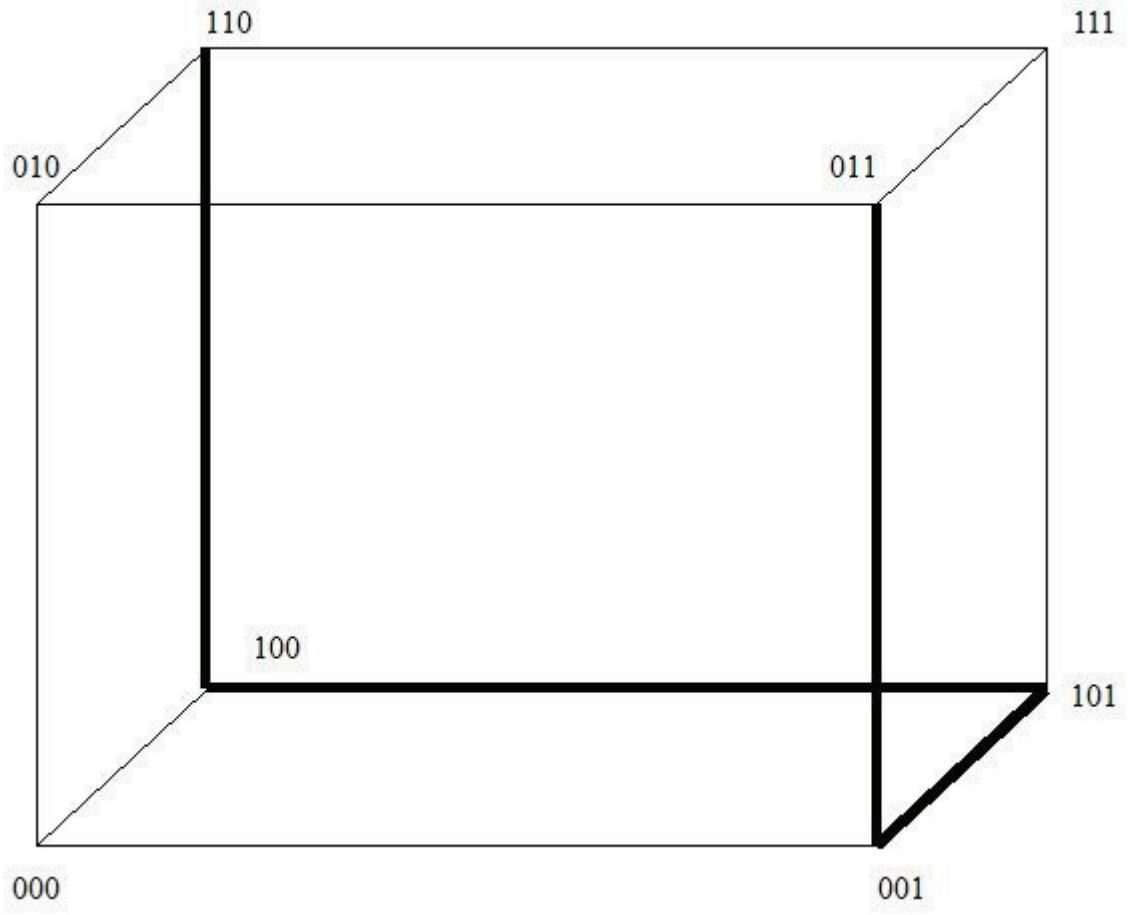

Figure 1.8 – A snake of length 4 in a 3-dimensional hypercube using the same transitions as the snake in Figure 1.7, but starting at a different vertex.

To formalize the rules discussed previously, a sequence of vertices is a snake if and only if, for each vertex v_i, $HammingDistance(v_i, v_{i-1})=1$ and $HammingDistance(v_i, v_{i+1})=1$, for $1 < i < m$ where m is the length of the sequence. And, for each vertex v_i, there does not exist v_j such that $HammingDistance(v_i, v_j) \leq 1$ for $j < i-1$ or $j > i+1$. Here $HammingDistance(v_i, v_j)$ returns the Hamming distance of the two vertices v_i and v_j, which is a reflexive operation. When applied to the hypercube, these rules dictate that the path cannot pass though a vertex that already neighbors another vertex on the path. Therefore, in Figure 1.7, when the path extends from $v0$ to $v2$ to $v3$, it may not then visit vertex $v1$ because $v1$ already neighbors a vertex on the

path, $v0$. All the vertices of the hypercube that neighbor a vertex on the path, but are not part of the path, are referred to as the snake's **skin**. Figure 1.9 shows an example of a snake and its skin in a 3 dimensional hypercube. The rules also exclude the possibility of the path moving from $v0$ to $v2$ and then back to $v0$ again.

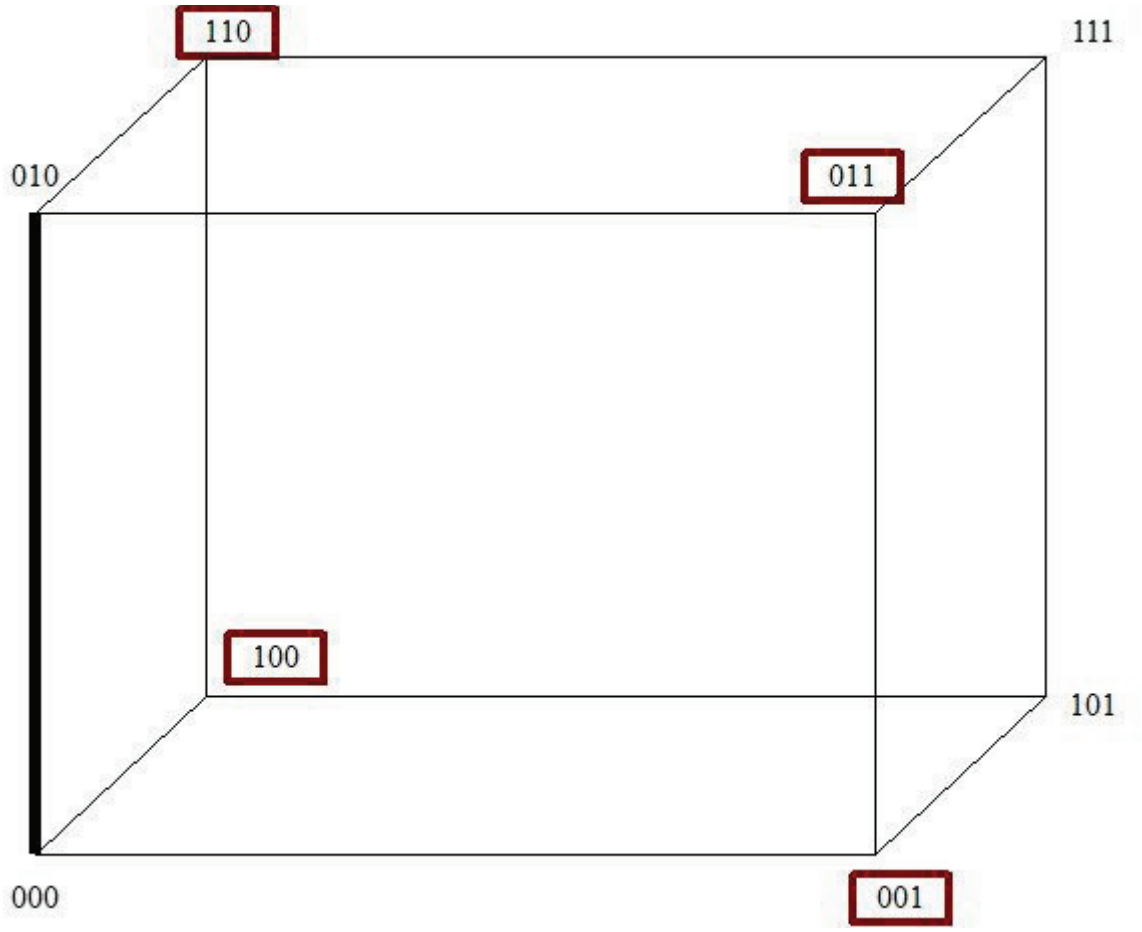

Figure 1.9 – A 3-dimensional hypercube with a two vertex snake, ($v0$, 1). The vertices {$v2$, $v3$, $v4$, $v5$} are shown highlighted here to signify that they are part of the snake's skin.

CHAPTER 2

GENETIC ALGORITHMS AND THE SNAKE PROBLEM

2.1 BACKGROUND

The snake problem consists of finding the longest possible snakes in a given dimension, and this thesis concentrates on searching in dimension 8. As illustrated in the context of linear encoders, longer snakes are more desirable because there is a strong correlation between a snake's utility and its length [Klee67]. When used in linear encoders, longer paths provide better resolution for measurement. However, the principal can be generalized to include all domains in which snakes are useful.

Searching for long snakes is a non-deterministically polynomial (NP) problem [Bitterman04]. The hypercube, which serves as a representation of the available search space, doubles in number of vertices for every increment in dimension. As the size of the search space expands, finding long snakes becomes an increasingly difficult problem. In fact, the search space becomes so large that exhaustive search for dimensions higher than 7 is considered infeasible with current technology [Kochut96, Casella05].

The sizes of the longest snakes in dimensions 1 through 7 have been found using exhaustive search and can be found in Table 2.1 and the current records for dimensions 8 and higher are listed at http://www.ai.uga.edu/sib/records/index.php. Unfortunately, there exists no mathematical proof to confirm whether or not a snake of a particular length is the longest possible snake in a given dimension. Therefore, the only way to prove that there are no snakes

larger than a given example is to do an exhaustive search. Current technology prohibits such a search in dimensions greater than 7, so the longest snake in dimensions 8 and greater will remain an open problem until technology is able to perform an exhaustive search on those dimensions or theory produces a method of proving that the known examples are the longest snakes possible.

Table 2.1 – All of the known longest snakes for dimensions 1 through 7. These snakes were confirmed to be the longest snakes by iterative search of all possible snakes in each dimension.

Dimension	Longest Snake
1	1
2	2
3	4
4	7
5	13
6	26
7	50

The process for exhaustively searching dimension 7 for solutions is described in [Kochut96]. In its entirety, the experiment took over a month of processing and was carried out by five SUN Microsystems SparcCenter 1000's, each having two processors.

As iterative search methods fail, acceptance for heuristic-based approaches increases. In particular, evolutionary computation has been applied to the snake problem with significant success. In fact, applications have been developed to search for snakes using Genetic Algorithms, Ant Colony Optimization, and Stochastic Hill Climbing [Potter94, Hardas05, Casella05].

2.2 INTRODUCTION TO GENETIC ALGORITHMS

In the early 1970s, Genetic Algorithms (GAs) were popularized by John Holland for their ability to mimic the concepts of Darwinian evolution [Holland75]. GAs have been applied

successfully to problems in planning, design, control, classification, function approximation, regression, music composition, and data mining [Engelbrecht02].

Darwinian evolution is a theory that concerns how the environment influences the phenotypes of a species. **Phenotypes** are observable expressions of genes. For example, eye color is a phenotype that is the result of the expression of some gene. The **chromosome** of an individual is an organized structure containing many genes. The individuals of a species each have slightly different genes and, therefore, express different phenotypes.

One of the key components of Darwinian evolution is natural selection. **Natural selection** postulates that phenotypes which improve an individual's chance of survival and reproduction in an environment are more likely to be passed on to future generations. Individuals with phenotypes that are particularly well suited for the environment in which they live, are more likely to be successful than those lacking such advantages. Individuals with higher success in an environment are said to have better **fitness**. Over the course of several generations, the individuals of a species with greater fitness tend to create more offspring than those with poorer fitness. As a whole, the population has more genetic material from individuals with favorable phenotypes, leading to an increased percentage of the population expressing those phenotypes. The effect is that populations become better suited for their environments over time.

GAs use this process as a template to find good solutions to a problem. In Darwinian evolution, a population is composed of many individuals and each individual is more or less fit in the environment. For a GA, the **individuals** are candidate solutions for the problem. The collection of all possible candidate solutions is called the problem's **search space**. The candidate solutions have varying levels of fitness to the problem being solved. The measurement used to map a candidate solution to a fitness value is called a **fitness function** and is an

important component for the success of the GA. Further, a candidate solution is composed of **partial solutions**, which can be thought of as genes that code for specific phenotypes, and has a **representation**, which is likened to the chromosome in living organisms.

In order to find a good solution to a problem, a GA mimics the evolutionary process. To begin, the population of the GA is initialized with a group of individuals, referred to as the first **generation**. The individuals of the first generation are paired up and produce offspring to create the next generation. This process then repeats for many generations. Individuals with better fitness values are given preferential treatment when selecting parents to create offspring. This preferential treatment simulates natural selection in biological systems.

While a GA uses a relatively simple algorithm that operates by creating new generations from the current generation, it relies on many disparate elements and mechanisms that work in harmony to produce good individuals. Mapping the problem to the GA metaphor is not always a straightforward process, and there are a number of decisions that have to be made. Each of these elements will be discussed in detail in later sections.

2.3 EXPLORING THE SEARCH SPACE

The search space for a problem can be thought of as a complex landscape of hills and valleys. Here, hills represent areas in which the solutions are good with the peaks of the hills being the best solution the hill has to offer. In contrast, the valleys represent poor solutions to the problem with the worst solutions at their lowest points. In the search space, some hills are taller than others. At the top of the tallest hill is the point in the search space that represents the best possible solution. This solution is referred to as a **global maximum**. All of the candidate

solutions that reside at the tops of the other hills are referred to as **local maxima**, because they are still the best solutions in their area, but are not the best solution possible.

The GA begins with a population of individuals. This initial population is the first generation. Each individual is a candidate solution to the problem with a measurable fitness and, collectively, they act as a survey of many different points in the search space. Some individuals will be better than others. The good individuals can be thought of as standing on one of the hills in the solution space, while the individuals with lower fitness values are located in one of the valleys.

The GA creates a new generation from the genetic material of the initial generation using operators that are designed to mimic those processes found in biological systems. The new generation will be a different survey of the search space, with the new individuals standing in different places. The goal of the search is to find the global maximum; therefore, the candidate solutions with the best fitness values are usually given some preference for being included in the creation of the next generation. The hope is that the individuals that were standing on the hills in the previous generation will create some offspring that are also standing on hills and, perhaps, are on a higher hill than their parents. In essence, the hope is that the individual as a whole is composed of good partial solutions that can be mixed with good partial solutions of other individuals to create better and better candidate solutions.

If the GA is effective, the individuals will start to concentrate search on one hill that seems to be better than the rest. The genetic material in the population as a whole will tend to become less heterogeneous. When this happens the GA, is said to **converge** because the search goes from being a survey of many different points in the solution space to being concentrated in a specific area. The concentrated search is an effort to find the top of the hill. Unfortunately,

there is no way for the GA to know whether or not it concentrated its search on the right hill. If it did not, then it will likely find a local maximum but, having concentrated on searching the wrong area, will miss the global maximum.

2.4 APPLICATION TO SNAKE PROBLEM

Many different approaches have been used in an attempt to find long snakes. Theoretical approaches have been used as in [Rajan99]. The use of exhaustive search was successful in finding all snakes in dimensions less than 8, including the work done in dimension 7 in [Kochut96]. A variety of heuristic based approaches have been used including Ant Colony Optimization in [Hardas05] and Stochastic Hill Climbing in [Casella05]. GAs have already been successfully applied to the snake problem, but what about the snake problem makes GAs a good approach for finding long snakes?

To determine if a GA may be a good candidate for a problem there are several characteristics of the problem that need to be considered. To start with, GAs are a heuristic search technique, meaning that they are not guaranteed to find an optimal solution to the problem. If the search space is small enough to be searched iteratively, then an iterative search should be used to ensure that an optimal solution is discovered. Unfortunately, exhaustive search is not an option for the snake problem in dimensions higher than 7.

Additionally, potential solutions for the problem need to be easy to evaluate and grade. GAs repeatedly select good answers from a collection of potential solutions and then attempt to generate better solutions. The process is iterative and many potential solutions are analyzed in the course of the process. If checking the solution takes a long time, the GA's generate-and-check methodology will take too long to make it effective. Furthermore, for the solutions to be

gradable there need to be solutions that are obviously better than others based on some measurement. If, in the general case, most solutions are not gradable, the GA will be unable to converge. Individuals for the snake problem are both easy to evaluate as solutions and are gradable.

Furthermore, there should be a reasonable expectation that good partial solutions will lead to good solution candidates. For the snake problem, this seems to be a very reasonable assumption because any subsection of a snake is also a snake.

2.5 REPRESENTATION

The preliminary consideration in GA design is the fact that the individuals require some form of chromosomal representation. The **chromosomal representation** is a method of representing a candidate solution to the problem being solved. The representation should incorporate all information necessary for an optimal solution and should be fitting to the domain. However, this does not mean that the representation must be able to express *every possible* solution. Instead, the representation need only be capable of representing the optimal solution. This distinction would typically involve preventing candidate solutions that are invalid because they violate some constraint of the problem. For the snake problem, there are methods of constraining the search because redundancy exists.

The set of all solutions that can be described using a representation, in conjunction with any constraints placed on the representation, will be referred to here as the **representation space**. The representation space is a subset of the entire search space for the problem. Choosing a representation with a representation space that is smaller than the search space is acceptable provided the representation is capable of expressing the optimal solution to the problem. This

may be possible if, for example, the solutions in the search space are highly redundant. In this case, a representation may be used which limits or eliminates such redundancy. The distinction between the search space and a representation space is important for reasons that will become evident later.

For the snake problem specifically, representations come in two flavors: vertex-based and transition-based. Vertex-based representations build a chromosome from vertices of the hypercube. Therefore, each candidate solution is a sequence of numerals that correspond to the vertices or the hypercube.

In contrast, transition-based representations build a chromosome from the edges of the hypercube. Transition-based representations are also sequences of numerals, but the numerals correspond to the transitions in the hypercube. In other words, the numbers indicate which bit place changes between the previous vertex and the current vertex. Recall that the head chosen for a transition sequence only affects where the sequence begins in the hypercube but does not change whether the resulting vertex sequence is a snake or not. By taking advantage of this property of the hypercube, transition-based representations have a much smaller representation space than vertex-based representations. This is because, for each unique candidate solution in a transition-based representation, there would exist 2^d candidate solutions in a vertex-based representation all starting at different vertices, where d is the number of dimensions. By not needing to represent each of these 2^d candidate solutions separately, the transition-based representation has fewer candidate solutions to consider and, therefore, a smaller representation space.

Furthermore, in a transition-based representation, only valid transitions are being searched. A **valid transition** is a transition that corresponds to an edge in the hypercube. An

invalid transition occurs whenever two consecutive vertices in a series are not neighbors. For example, *v*0 and *v*3 differ have a Hamming distance of 2. If these two vertices were to appear next to each other in a vertex-based representation, there would be an invalid transition between them because they do not share an edge in the hypercube. A sequence of vertices cannot be a snake if it contains an invalid transition. In a vertex-based representation, invalid transitions are possible, whereas transition-based representations only use valid transitions. Removing all candidate solutions with invalid transitions further reduces the representation space for a transition-based representation versus a vertex-based representation.

In addition, individuals may either be fixed or variable length. For the snake problem, fixed length individuals are typically used. Fixed length individuals tend to simplify crossover operators and fitness functions, which will be discussed later. Furthermore, variable length individuals can have other drawbacks aside from added complexity. Without methods to manage the length of the individuals, they may tend to grow in size over the course of several generations without a corresponding payoff in the lengths of snakes contained in those individuals.

This thesis focuses on a transition-based representation with a fixed length of 110 because it has the potential to represent a snake of length 110 should one exist in dimension 8. Within the chromosome of an individual, there may exist several snakes of varying length.

2.6 INITIALIZATION

Once a representation has been established, the first generation of individuals must be initialized to provide the chromosomal pool from which subsequent generations will be built. One technique used in this thesis will be referred to as Restricted Random Initialization (RRI). RRI initializes an individual by the algorithm in Figure 2.1. RRI's name signifies that the

transition being added at the current position is restricted from being from the same transition class used in either of the last two positions. This restriction avoids creating individuals with snake blockers. **Snake blockers** are two special transition patterns that serve as barriers to block snakes. There are two types of snake blocker, one is when the same transition appears twice in a row and the other is when the same transition occurs twice with only a single different transition in between. The two types of snake blockers will be discussed in full later.

The second technique is referred to as Restricted Heuristic Initialization (RHI) and is outlined in Figure 2.2. RHI uses a heuristic to guide the addition of transitions to the end of the individual. RHI starts by attempting to build a snake one transition at a time. As long as there are transitions available that will not violate the rules for a snake, it will pick one of those transitions. The probability for selecting a transition is weighted based on a factor called skin density, which will be an important concept later. The use of a weighted probability is a heuristic used to create longer starting snakes. After RHI runs out of available transitions, it finishes the individual's chromosome with the same algorithm that RRI uses.

```
function RRI (integer[] individual, integer dimension){
        integer length = individual.getLength();

        individual[0] = individual[1] = get_random_integer_less_than(dimension);

        while (individual[0] == individual[1]){
                individual[1] = get_random_integer_less_than(dimension);
        }

        // here the first two integers in the array have been
        // initialized to two different transitions

        for( index = 2; index < length; ){
                individual[index] = get_random_integer_less_than(dimension);

                if(individual[index] != individual[index-1] and
                        individual[index] != individual[index-2]){
                        index++;
                }
        }

        // here the entire array has been initialized to transitions
        // that don't conflict with the previous two in the sequence

}
```

Figure 2.1 – Pseudocode for Restricted Random Initialization.

```
function RHI (integer[] individual, integer dimension){
        integer length = individual.getLength();

        individual[0] = individual[1] = get_random_integer_less_than(dimension);

        while (individual[0] == individual[1]){
                individual[1] = get_random_integer_less_than(dimension);
        }

        // here the first two integers in the array have been
        // initialized to two different transitions

        index = 2;
        available_transitions = get_available_transitions(individual);

        while(index < length){
                if(available_transitions.getLength() > 0){
                        individual[index] = pick_from_available(available_transitions);
                        available_transitions = get_available_transitions(individual);
                } else {
                        individual[index] = get_random_integer_less_than(dimension);
                }

                if(individual[index] != individual[index-1] and
                        individual[index] != individual[index-2]){
                        index++;
                }
        }

        // here the entire array has been initialized to transitions
        // that don't conflict with the previous two in the sequence
        // the start of the individual is a snake of some nominal
        // length

}
```

Figure 2.2 – Pseudocode for Restricted Heuristic Initialization.

2.7 FITNESS FUNCTIONS

Fitness functions are arguably the most important part of a GA [Engelbrecht02]. The role of a fitness function is to indicate how good a candidate solution is at solving the problem. In order for a fitness function to be appropriate for the problem, it should consistently map better solutions to better fitness values. If the fitness function does not accurately map an individual to an appropriate fitness, the GA may fail to converge altogether. Therefore, careful selection of an appropriate fitness function is essential.

Some projects that focus on GAs as an approach to the snake problem utilize the length of the longest snake as the sole element in calculating fitness. Undoubtedly, the length of the longest snake is an important component in that it makes it easy to compare how good the candidate solutions are when they contain snakes of different lengths. Several alternatives have been suggested to help distinguish between individuals with snakes of equal length. One alternative is to take other snakes that may be present in the chromosome into account. This approach was used in [Bitterman04]. Another alternative is to quantify the quality of the longest snake. One way to do this is to judge how well the snake utilizes the space it occupies in the hypercube. Whenever a vertex is added to a snake, most of the neighbors of that vertex become part of the snake's skin and, therefore, cannot be used as part of the path later. One measurement, referred to here as **tightness**, is the number of vertices that are left unoccupied by either the snake or the snake's skin. This measurement is used in both [Casella05] and [Tuohy07]. Another measurement, referred to here as **skin density**, looks at the vertices that are part of the snake's skin and counts how many of their neighbors are part of the snake. When a skin vertex neighbors more vertices on the path, it is said to be denser. Skin density is used as a

component of work done in [Hardas05]. This thesis experiments with both tightness and skin density as components of the fitness function.

Furthermore, some work has been done using the frequency with which transition classes are used in the chromosome. The basic idea is to test whether the long snakes traverse some transition classes more than others. If long snakes typically use transition classes with roughly the same frequencies, then the distribution of those frequencies can be used to select for genetic material with the right distribution. Further discussion on the motivation and application of this hypothesis will be provided later.

2.8 SELECTION

The key mechanism of GAs is the iterative creation of new, and hopefully better, generations using the genetic material from the current generation. Before the next generation can be created, parent individuals must be selected from the current population to be used in the reproduction process. Parents are selected two at a time. They are used to create offspring for the next generation and then placed back into the population where they are eligible to be selected again. This process repeats until the next generation is full, which is usually when it reaches the same size as the current generation.

Selection operators are methods for determining which individuals from the current generation should be paired to produce offspring for the population of the next generation. This project compares random, rank-based, roulette wheel, and tournament selection operators.

Random selection picks two individuals from the population at random to use as parents for the next generation. Because random selection does not take the fitness of the individuals

into account, it is unlikely to be a high achieving selection method. Regardless, it serves as a good baseline from which to judge the performance of the other selection operators.

In roulette wheel selection, the probability of selection is directly proportional to the fitness of an individual in relation to the other individuals in the population. Once the fitness of each individual has been evaluated, the finesses are added together to get a total sum for the population. Selecting an individual entails randomly generating a number between 0 and the total fitness sum for the population. The fitness values for each individual are subtracted from the randomly generated number until it falls below zero. The individual that causes the value to fall below zero is selected as a parent. Figure 2.3 shows the probability of being selected for each individual in a population of four. Individual 3 dominates the graph because its fitness value is significantly greater than those of the other three. The result is that Individual 3 is much more likely to be selected as a parent than the other members of the population.

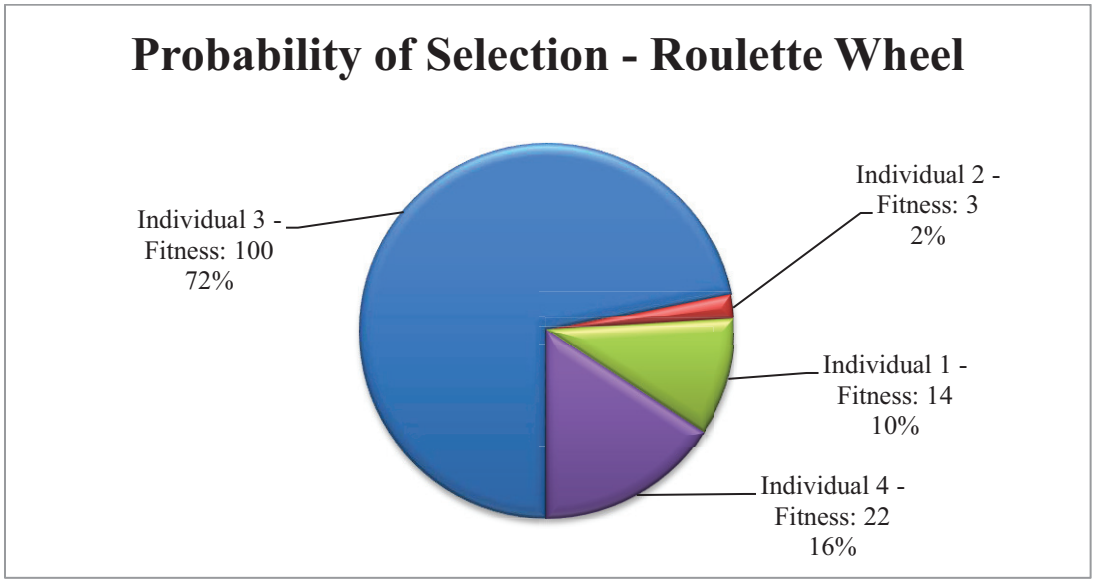

Figure 2.3 – The probability that each of four individuals in a population will be selected to be a parent when using roulette wheel selection. Individual 3 has a much higher probability of being selected because its fitness value is significantly greater than the fitness values of the other three individuals.

Rank-based selection assigns a probability of selection for an individual based on its ordinal position in the population once sorted according to fitness, where the individual with the lowest fitness is assigned a rank value of 1 and the individual with the highest fitness is assigned a rank value equal to the population size. To select an individual, a goal value is generated between 0 and the sum of the values assigned to the population. Next, the rank values of the individuals are subtracted from the goal value one at a time. The individual that causes the goal value to fall below zero is selected as a parent. Figure 2.4 shows the probability of selection for a four individual population. Notice that Individual 3 has a much higher fitness value than the other three members of the population. It is a property of rank based selection that individuals with fitness values that are significantly higher than the other members of the population do not necessarily dominate the reproduction process.

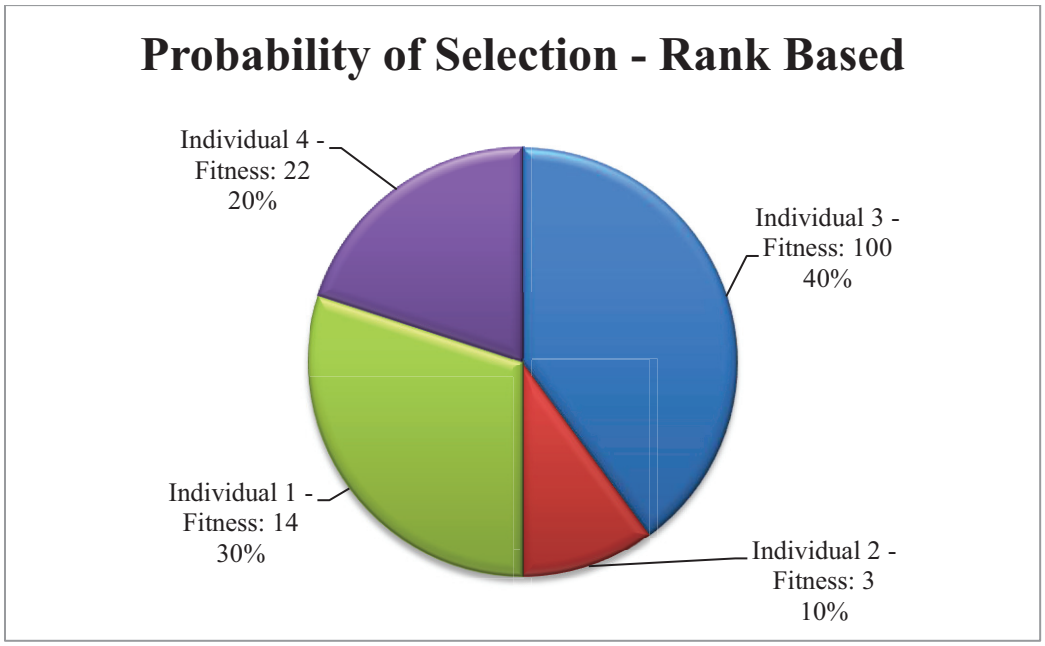

Figure 2.4 – The probability that each of four individuals in a population will be selected as a parent when using rank-based selection.

In tournament selection, mates are selected based on highest fitness from a group of *k* individuals chosen at random from the population. The probability that any one individual will be selected to participate in reproduction is more difficult to calculate with tournament selection, but is a function of the population size, the value of *k*, and the fitness of the individual.

2.9 REPRODUCTION

Once two parents have been selected from the population, reproduction operators are used to produce offspring. Reproduction operators, also referred to as crossover operators, are methods of taking individuals from the current generation, mixing their genetic information, and creating new individuals for the next generation. Techniques for creating offspring using reproduction operators vary and some representations work better with certain types of reproduction. This project works with single point, double point, triple point, and uniform crossovers.

In single point crossover, a point is randomly selected between the start and the end of the individuals. For a fixed length representation, only one point needs to be selected. The operation creates two children. The first child starts with the genetic material from the first parent, before the randomly selected point, and ends with the genetic material from the second parent, after the selected point. In the second child, the front portion comes from the second parent and then end portion comes from the first parent. Figure 2.5 shows an example of single point crossover. Double and triple point crossovers operate on the same principle, but select two and three points, respectively. Figure 2.6 and Figure 2.7 show examples of double and triple point crossover.

In uniform crossover, genetic material is swapped at a collection of points generated at random. To select the points that will be swapped, a mask is created that is the same length as the parents. A mask is an array used to hold the information about which bits will be swapped. The mask starts out as an array of zeros. At each point in the array, the zero may be switched to a one based on some probability. When crossover occurs, the positions in the mask that are marked with zeros pass straight through to the children, parent 1 to child 1 and parent 2 to child 2, while the positions in the mask that are marked with ones are then swapped with the other child, parent 1 to child 2 and parent 2 to child 1. Figure 2.8 shows an example of uniform crossover broken into two stages for clarity.

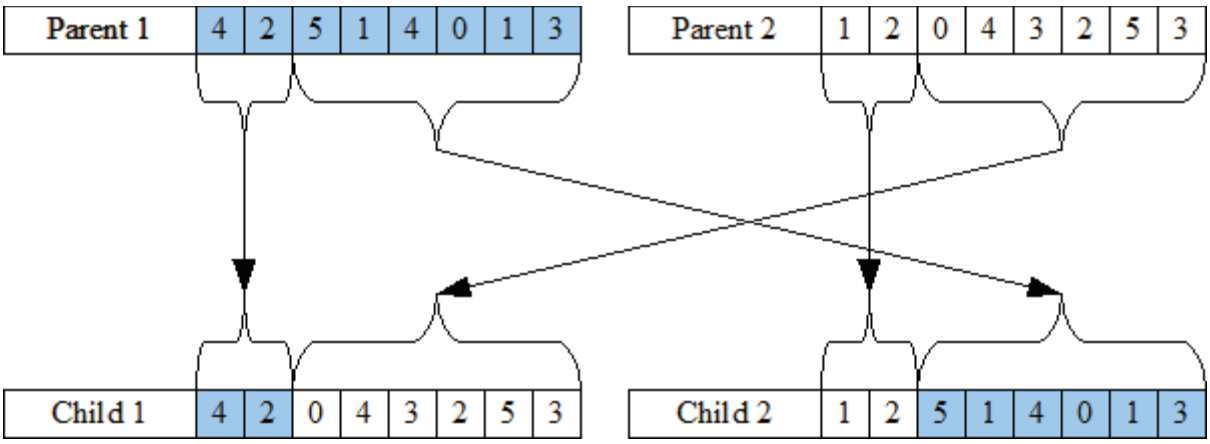

Figure 2.5 – An example of single point crossover.

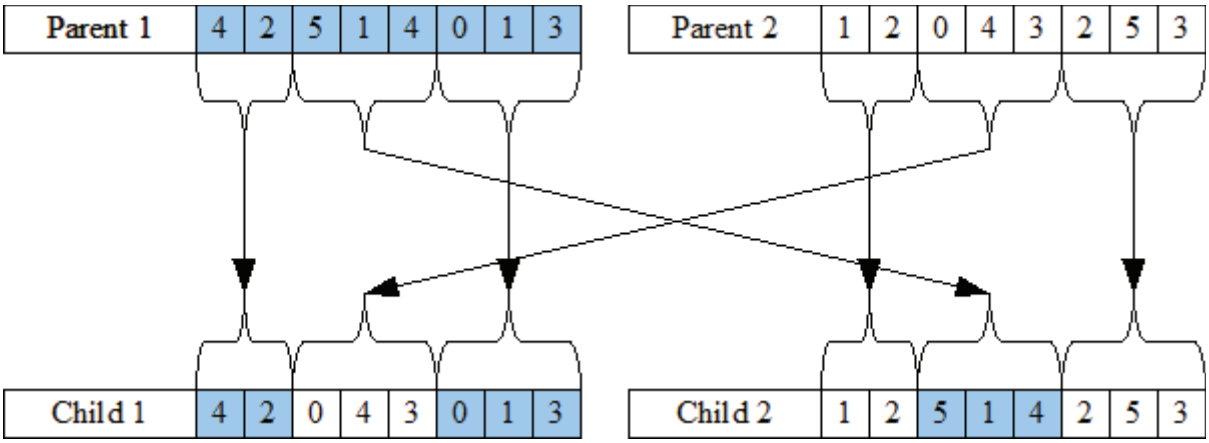

Figure 2.6 – An example of double point crossover.

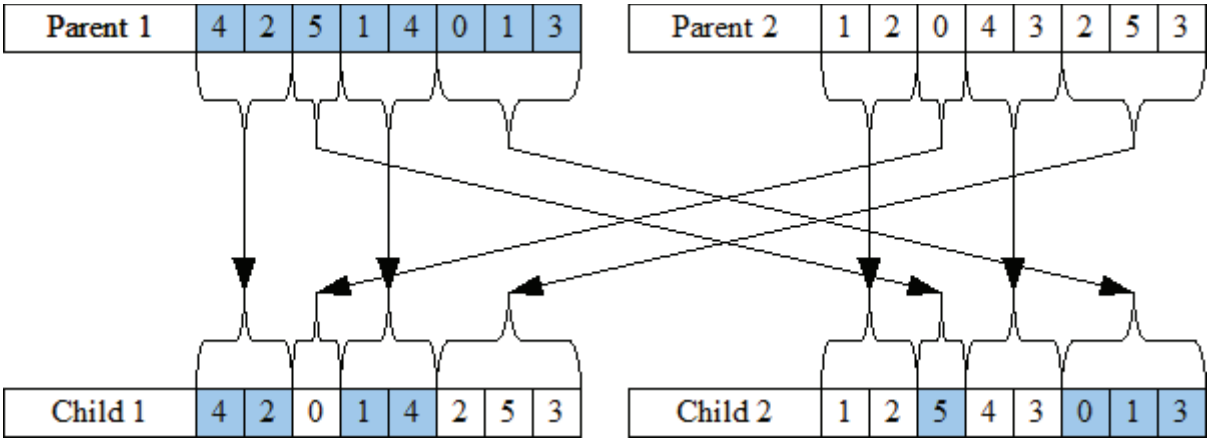

Figure 2.7 – An example of triple point crossover.

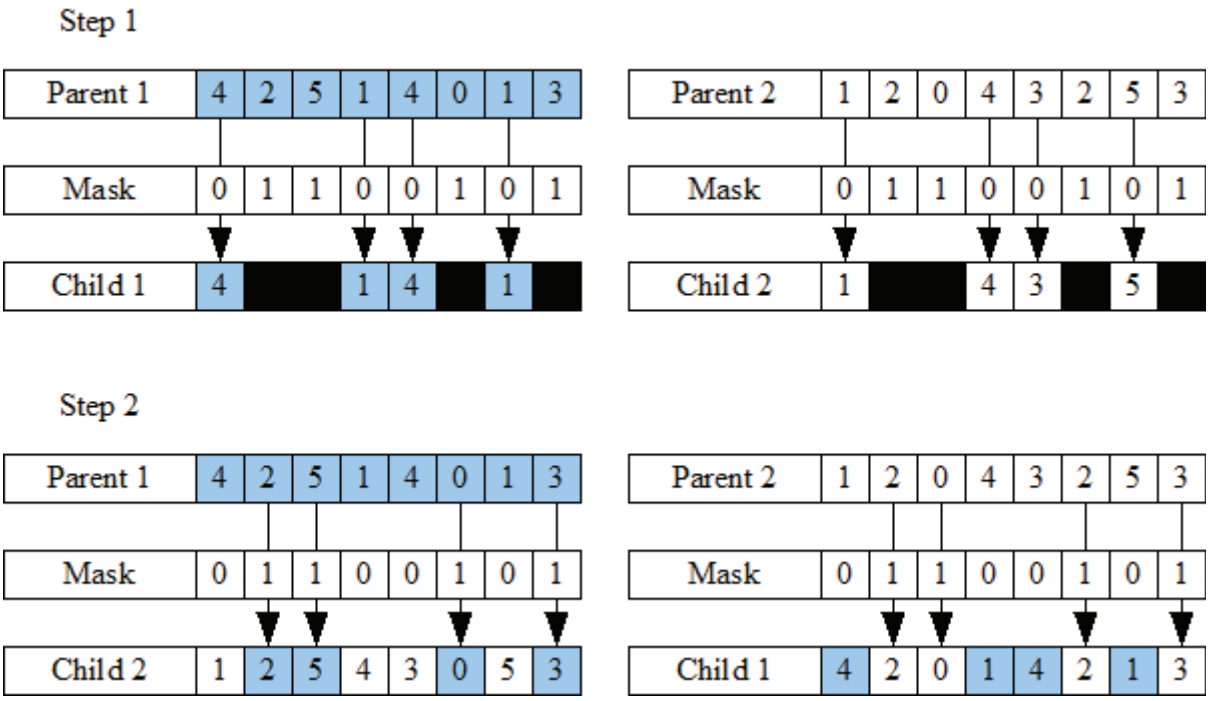

Figure 2.8 – An example of uniform crossover, it is shown here as a two step process.

2.10 MUTATION

After the children have been produced, they are subject to mutation. Mutation operators work on a single individual and take place after reproduction has occurred. With a probability that has been decided in advance, a mutation operator changes the chromosome of the child. These operators are a mechanism for introducing genetic diversity that may not be represented in the population. The changes made to the chromosome may be beneficial or may be harmful to the individual's fitness. This project uses two different mutation schemes: random mutation and a snake specific mutation operator commonly referred to as Xor mutation.

In random mutation, a gene in the individual is replaced with a different gene. This is a low cost operation and is simple to implement. In the case of the snake problem and a transition-based representation, this means replacing one transition with another transition. Figure 2.9 shows an example of random mutation.

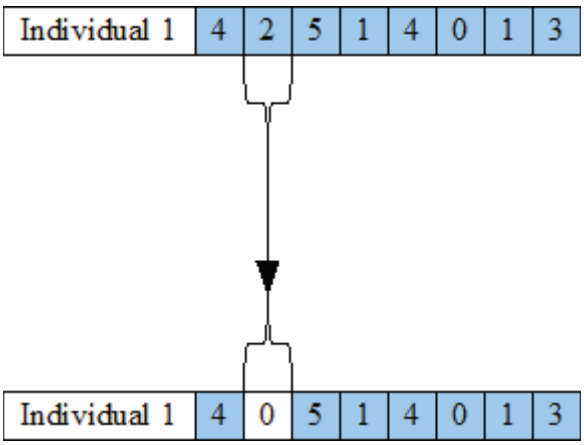

Figure 2.9 – An example of random mutation.

Xor mutation takes advantage of the properties of the hypercube to introduce changes to the chromosome. As the hypercube illustrates, there are two paths of length two between vertices of the hypercube with a Hamming distance of two. Figure 2.10 shows these two paths between $v0$ and $v3$, one traveling through $v1$ and one traveling through $v2$. Xor mutation replaces one of these two transition routes with the other. The result is that two neighboring transitions in the individual have been transposed. Figure 2.11 shows an example of Xor mutation. Xor mutation gets its name from the fact that using the exclusive or operator (xor) on the vertices in one path will result in the replacement vertex for the other. For example, the first path in Figure 2.10 is ($v0$, 1, 3). Applying the xor operator, shown here as the symbol ∨, yields 00 ∨ 01 ∨ 11 = 10. Therefore, the vertex that is needed to replace $v1$ and yield the alternate route is $v2$.

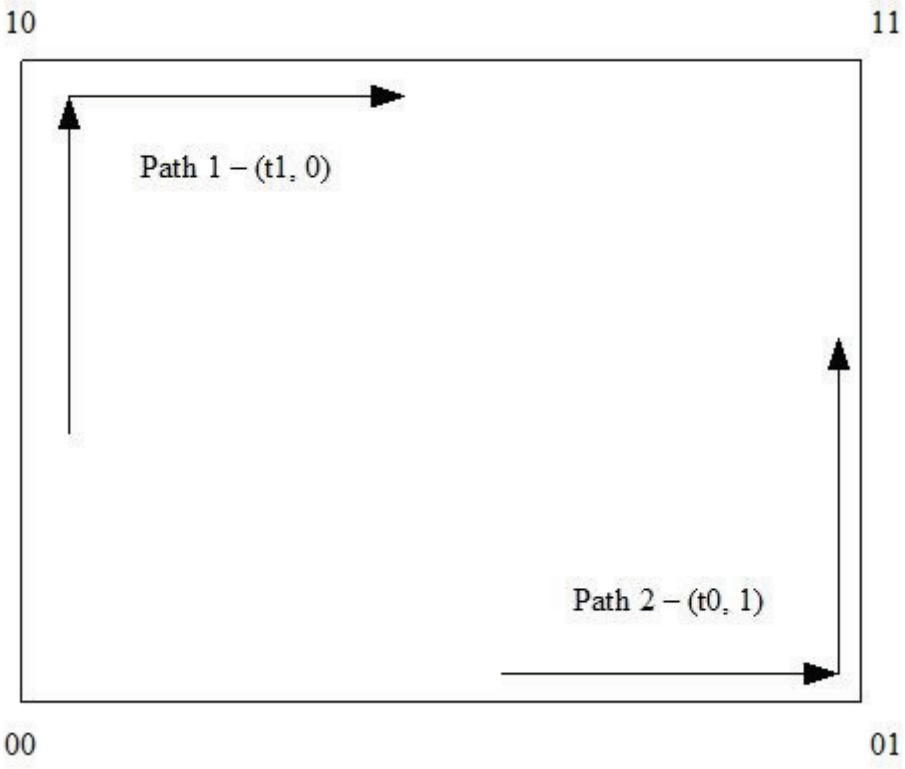

Figure 2.10 – The two potential paths from *v*0 to *v*3 of length 2.

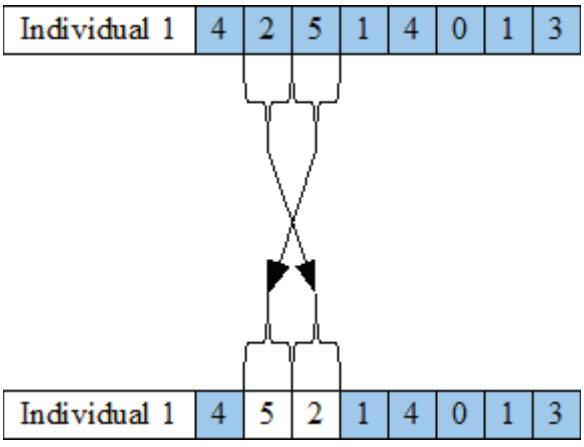

Figure 2.11 – An example of Xor mutation. The end effect is that two consecutive transitions are transposed in the result.

2.11 ELITISM

Elitism is used to ensure that the best individuals are always present to influence future generations. Elitism is when a group of the best individuals from the current population is copied directly into the next generation, bypassing any reproduction or mutation operators. In theory, the best individuals are composed of the best partial solutions. Therefore, retaining the individuals with best fitness is more likely to result in better offspring. The negative aspect of elitism is that it has the potential to drive populations toward local maxima over the course of several generations. All GA experiments run for this thesis used elitism and copied the top ten individuals from the current generation to the next. Since all of the experiments were run with a population size of 500 individuals, this translates to 2% of the population.

2.12 CHALLENGES

GAs are a good alternative for problems that have search spaces that are too large for iterative analysis. However, excessively large search spaces also hinder GA performance. In the snake problem, one of the most difficult issues is the size of the search space.

Although http://www.ai.uga.edu/sib/records/index.php lists the longest known dimension 8 snake as being 98 transitions long, this snake has not been published as of the writing of this thesis. The longest published snake known in dimension 8 is 97 transitions long. Mapping the transitions to vertices creates a sequence of 98 vertices. In dimension 8 there are 256 vertices total. Therefore, the number of vertex-based individuals of length 98 that can be expressed using 256 vertices is:

$$256^{98} \approx 1.01 \times 10^{236}$$

By adding the constraint that each vertex can only be used once the total number of vertex-based individuals of length 98 that can be expressed using 256 vertices is:

$$P(256,98) = \frac{256!}{(256-98)!} \approx 4.62 \times 10^{226}$$

For a transition-based representation, the number of length 97 individuals that can be represented is:

$$8^{97} \approx 3.97 \times 10^{87}$$

While the use of a transition-based representation significantly reduces the number of individuals that can be represented, there are still an enormous number of candidate solutions in dimension 8.

In addition to the size of the search space, there are many more short snakes than there are long snakes. When using length as a fitness indicator, this poses a problem for the GA because it will encounter many different individuals with equal lengths. If length is used as the sole indicator of fitness, these individuals will be indistinguishable. The result is that all the individuals are viewed equally and the search is more random. In order to provide a more guided search, distinctions need to be made between the relative fitness values of individuals of equal length.

This thesis looks to address both of these challenges. A variety of techniques are explored for reducing the overall representation space for the problem. Furthermore, fitness functions are analyzed in great detail to evaluate which indicators work the best for describing the fitness of an individual, and testing methods of supplementing length as the sole indicator.

CHAPTER 3

ADDRESSING THE SIZE OF THE SEARCH SPACE

3.1 BACKGROUND

The search space for dimension 8 is restrictively large for conventional search techniques, but excessively large search spaces also hinder GA performance. One contributing factor to the size of the search space is the fact that each transition sequence is a generalization of 2^d vertex sequences in a d-dimensional hypercube, each having the same sequence of transitions but starting at a different vertex. For this reason and because they only represent valid transitions, transition sequence representations have a smaller representation space than vertex-based representations.

Despite these advantages, there is still a great deal of redundancy in a transition-based representation. In order to discuss how these redundancies might be addressed, the concept of transition reassignment needs to be introduced. **Transition reassignment** is when all of the transitions from one transition class are swapped with the transitions of another transition class. Figure 3.1 shows an example of transition reassignment. The figure also shows a transition map which indicates which transition classes changed and to what new value.

When two individuals can be made to have the same transition sequence by performing one or more transition reassignments for the whole chromosome, they are said to be in the same **equivalence class**. Because the transition reassignments were applied uniformly across the individual in Figure 3.1, these are two instances of the same equivalence class. When two

transition sequences from the same equivalence class are mapped to the hypercube, the vertices at any two positions will have the same Hamming distance for the two vertex sequences. In effect, the transitions will have the same shape or a reflection of the same shape. Figure 3.4 shows this in context and can be compared to the snakes in Figure 1.7 and Figure 1.8.

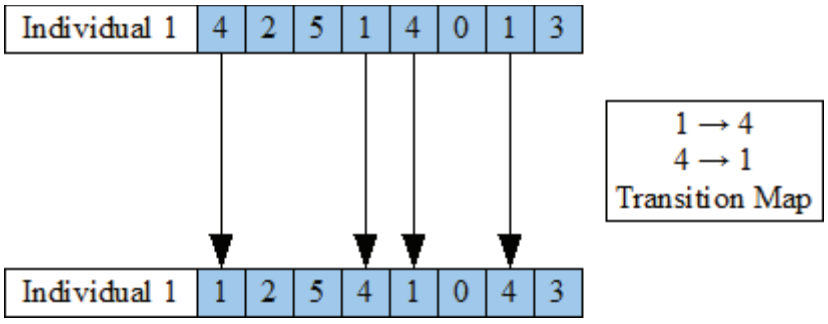

Figure 3.1 – An example of transition reassignment. *t*1 is being reassigned to *t*4 and vice versa.

Furthermore, when the transitions are introduced for the first time in sequential numeric order, from left to right, the individual is said to be **canonical**. Figure 3.2 shows a dimension 3 equivalence class and how each member of the equivalence class maps to its canonical equivalent. Figure 3.3 shows three individuals from different equivalence classes. Figure 3.4 shows two snakes from the same equivalence class as represented in the hypercube.

Equivalence classes are important because if two individuals are in the same equivalence class, they will both have the same number of snakes with the same lengths and in the same positions in the chromosome. In addition, performing transition reassignment on a snake will always result in a snake in the same equivalence class, as long as the transition reassignment is performed in a uniform manner across the whole snake. Because snakes in the same equivalence class can be generated from one another, there is no need to search for all snakes in the equivalence class. Having one is as good as having all of them. Therefore, a good approach to

dealing with the size of the search space is to limit the representation to only express one snake from each equivalence class.

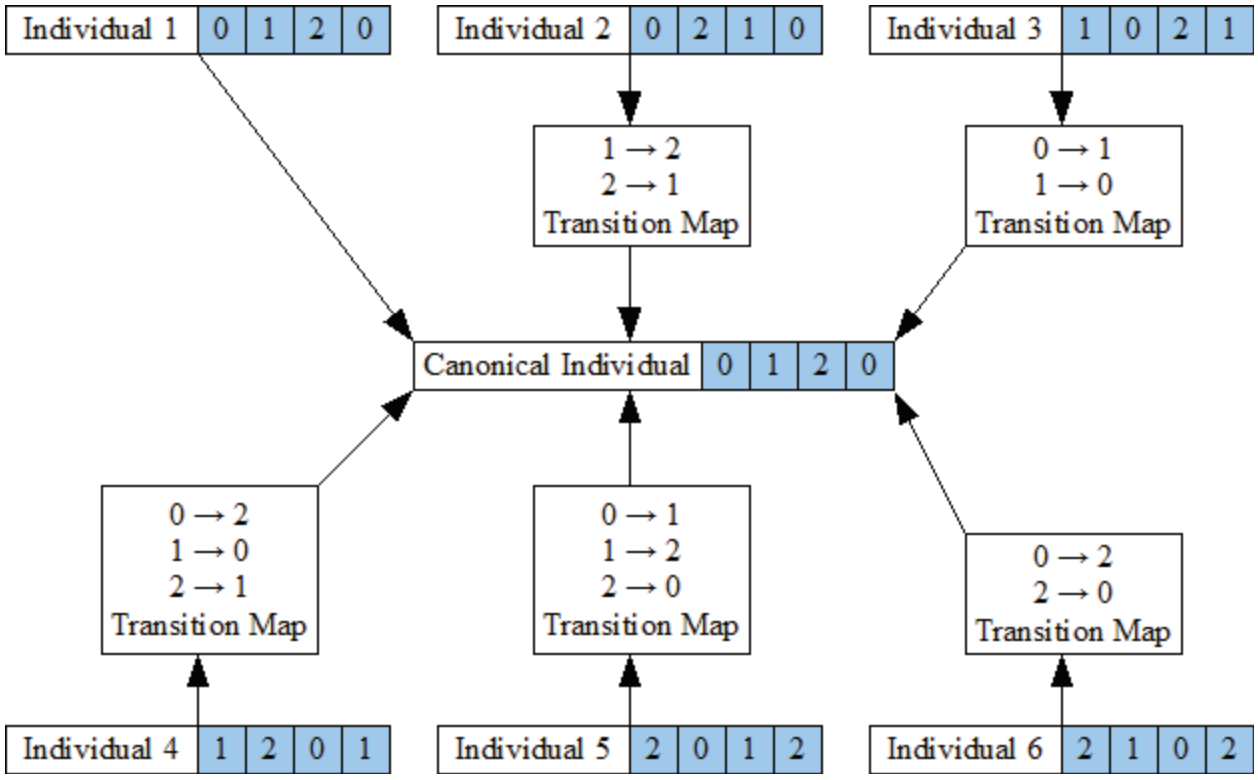

Figure 3.2 – An equivalence class with six individuals in dimension 3. Each of the six individuals is shown with the transition mapping necessary for transition reassignment to the canonical form in the center.

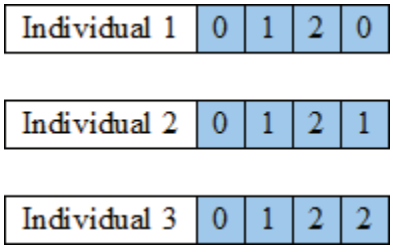

Figure 3.3 – Each of these three individuals is in a different equivalence class. All are shown in canonical form. In Individual 1, the transition pattern is ABCA, which differs from the patterns for Individual 2, ABCB, and Individual 3, ABCC.

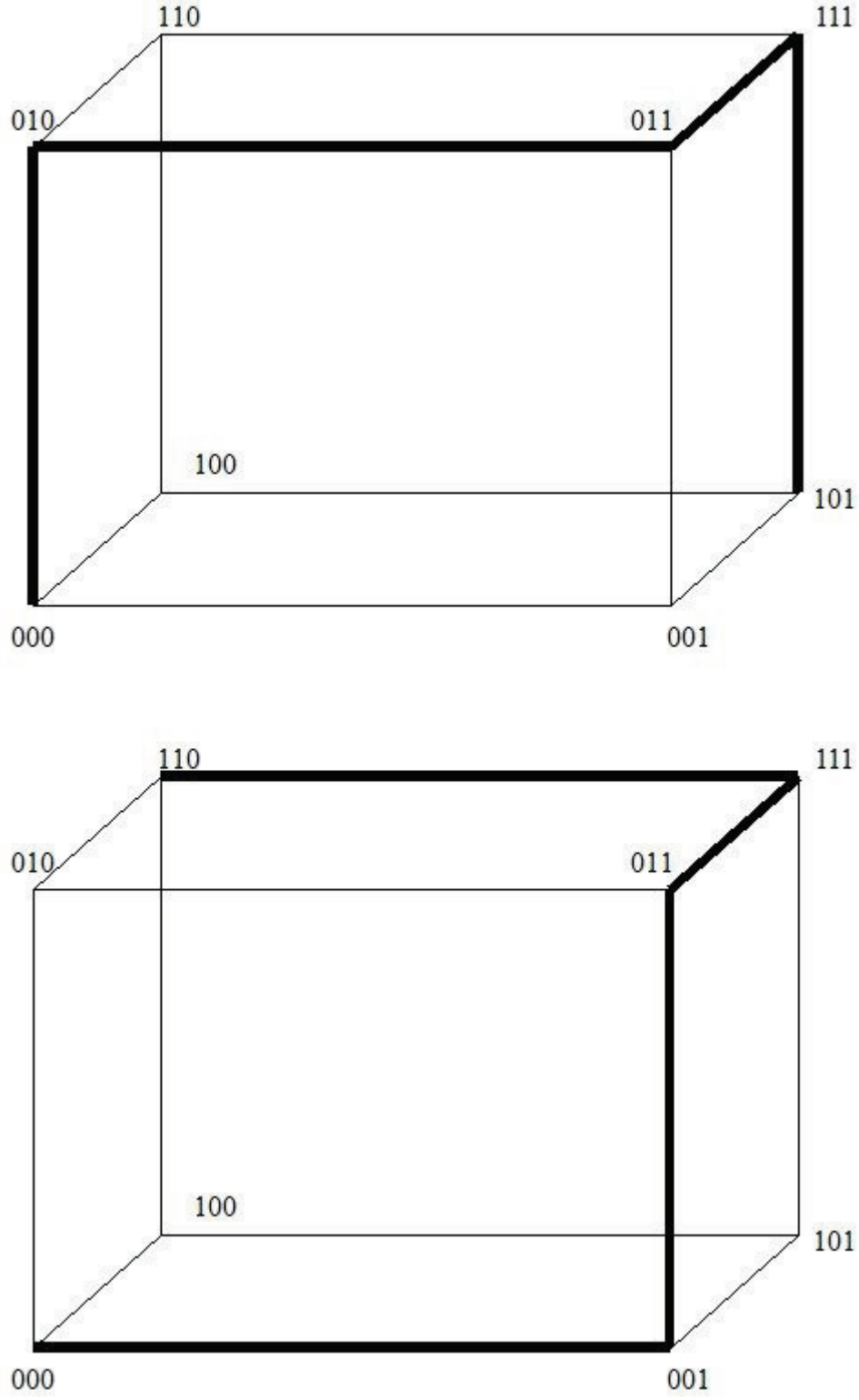

Figure 3.4 – A comparison of two snakes from the same equivalence class in the hypercube. Both snakes start at $v0$. The first snake is ($t1$, 0, 2, 1) and the second is ($t0$, 1, 2, 0).

3.2 CANONICAL REPRESENTATION

For each snake using d transition classes, $d!$ snakes exist in the same equivalence class in dimension d [Kochut96]. Therefore, even with the utilization of a transition-based representation, there exist $d!$ equivalent individuals for each individual being analyzed, making it easy to waste time analyzing many equivalent versions of the same individual without making definitive progress.

In [Kochut96], the process of exhaustively searching for dimension 7 is described. The work took over a month and was carried out on five SUN Microsystems SparcCenter 1000's, each having two processors. Only canonical solutions were considered. The paper defines a canonical solution as starting at the $v0$ with an initial transition of $t0$, with each subsequent transition class not yet represented in the chromosome being assigned to the next unused transition of least significance. In other words, each solution introduced the transitions in consecutive order: $t0$, $t1$, $t2$, $t3$, $t4$, $t5$, and, finally, $t6$ [Kochut96].

Searching only canonical solutions allowed the search space to be pruned significantly because it provided a meaningful baseline, in which transition classes were interpreted uniformly across all individuals. Therefore, it was only necessary to visit one individual from each equivalence class.

While this approach was appropriate for an exhaustive search of dimension 7, it is unclear whether it will perform as well in a GA. The individuals are reassigned to their canonical equivalent based on the chromosome as a whole, i.e. starting from the first transition in the individual. Therefore, the chromosome is canonical, but if the head of the longest snake is not also the first transition in the chromosome, then the snake may not be represented canonically.

3.3 FREQUENCY-BASED TRANSITION REASSIGNMENT

As an alternative to using the canonical representation presented in [Kochut96], this thesis offers Frequency-Based Transition Reassignment (FBTR). As the name implies, FBTR performs transition reassignment on individuals based on the frequency with which the transition classes appear in the chromosome. In other words, the most frequently occurring transition class is mapped to $t0$, the second most frequently to $t1$, and so on until $t7$ for dimension 8. When conflicts are encountered, they are resolved by assigning the next available transition as the replacement for the transition class occurring in the leftmost position on the chromosome.

To help understand why this might be a justifiable approach, consider the two individuals in Figure 3.5. When comparing these two individuals, one is faced with the non-trivial question: How similar are these individuals? From the figure it is easy to see that they differ by only a single transition. It is tempting to conclude that they are very similar, but the problem of quantifying that similarity is difficult. If the transitions were to be compared one at a time, the two individuals have more transitions in common than not. But now, consider Figure 3.6 in which the two individuals have been reassigned to their canonical representations. Now, seven of the fourteen transitions in the individuals are different. Are the individuals now more different than they were before the transition reassignment?

Individual 1	1	0	2	3	4	0	3	1	2	0	3	4	2	3
Individual 2	3	0	2	3	4	0	3	1	2	0	3	4	2	3

Figure 3.5 – Two 5-dimensional individuals.

Individual 1	0	1	2	3	4	1	3	0	2	1	3	4	2	3
Individual 2	0	1	2	0	3	1	0	4	2	1	0	3	2	0

Figure 3.6 – The two dimension 5 individuals from Figure 3.5 in canonical representation.

Unfortunately, there are no easy answers to these questions. However, the questions are important to ask because the purpose of using a canonical representation is to reduce the search space. In this particular situation, using canonical representation hinders our ability to notice that these individuals are similar. While the chromosome as a whole is being interpreted in a uniform manner, the snakes within the chromosome are not necessarily canonical.

Figure 3.7 shows the result of applying FBTR to the snakes from Figure 3.5. While the results are an improvement, these examples are not necessarily indicative that FBTR will perform better more often than a canonical representation. The primary mechanism for minimizing the effects of equivalence classes that canonical representation utilizes is to standardize the interpretation of the transition classes at the beginning of the chromosome. In contrast, FBTR seeks to standardize the interpretations of the most frequently occurring transition classes. So, in the canonical representation, *t*0 always means the transition found at the beginning of the chromosome, and in FBTR, *t*0 always means the most frequently occurring transition class. By shifting to a standardized interpretation that is not focused on a particular position in the chromosome, the interpretation is more meaningful at more positions in general.

Individual 1	3	1	2	0	4	1	0	3	2	1	0	4	2	0
Individual 2	0	1	2	0	3	1	0	4	2	1	0	3	2	0

Figure 3.7 – The two dimension 5 individuals from Figure 3.5 after applying FBTR.

3.4 SNAKE BLOCKERS

Snake blockers are transition patterns that act as barriers to block snakes. There are two types of snake blocker. The first is when a transition occurs twice in a row. The result of following the same transition twice is that the path doubles back on itself, which violates the rules for a snake. Figure 3.8 shows how this looks in the hypercube. The second type of snake blocker is when the same transition occurs twice with only a single transition in between. This results in the path visiting a skin vertex, which also violates the rules for a snake. Figure 3.9 shows this type of snake blocker in the hypercube.

Snake blockers can be introduced through the operators of the GA. Both reproduction operators and mutation operators are capable of producing snake blockers. Figures 3.10 and 3.11 show the introduction of snake blockers using single point crossover and uniform crossover, respectively. Figures 3.12 and 3.13 show the introduction of snake blockers using random mutation and Xor mutation, respectively.

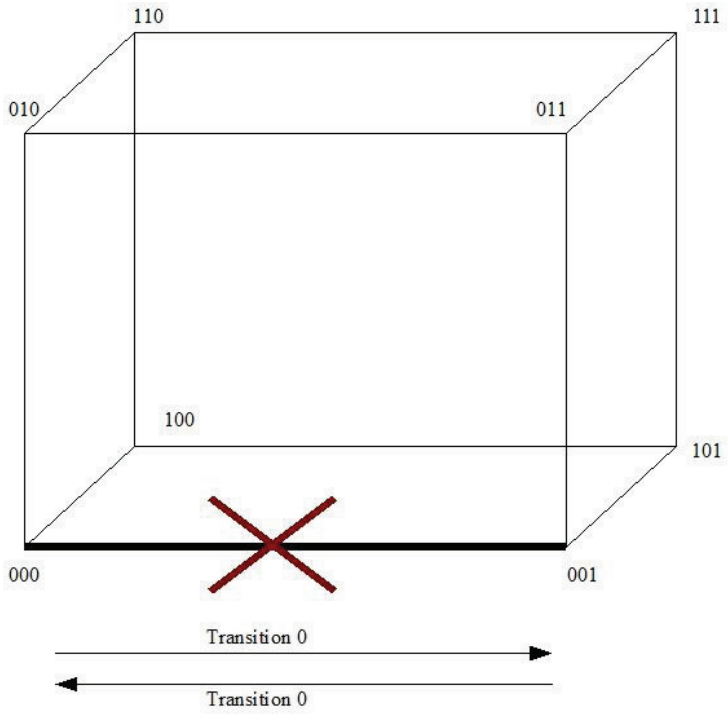

Figure 3.8 – The result of following the same transition twice. Starting from *v*0 following *t*0 twice results in going back to *v*0.

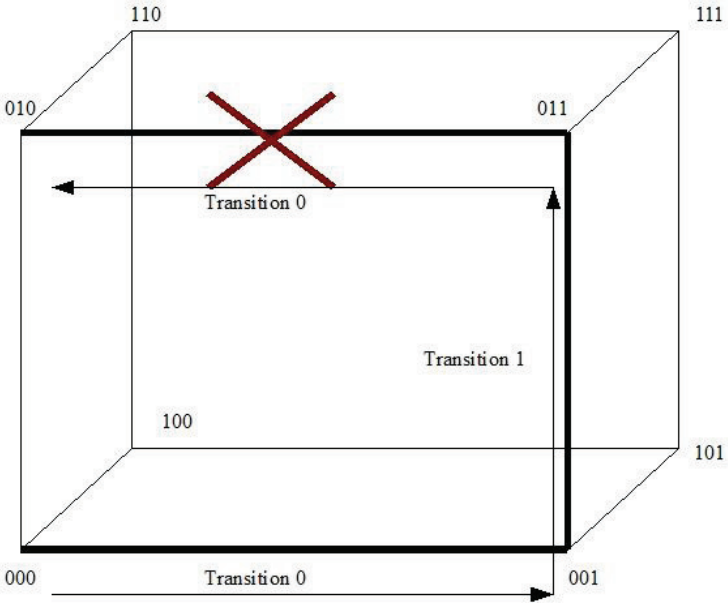

Figure 3.9 – The result of following the same transition twice with only one intermediate transition. Starting from *v*0 and following *t*0 then *t*1 and then *t*0 again results in the path transitioning to a skin vertex.

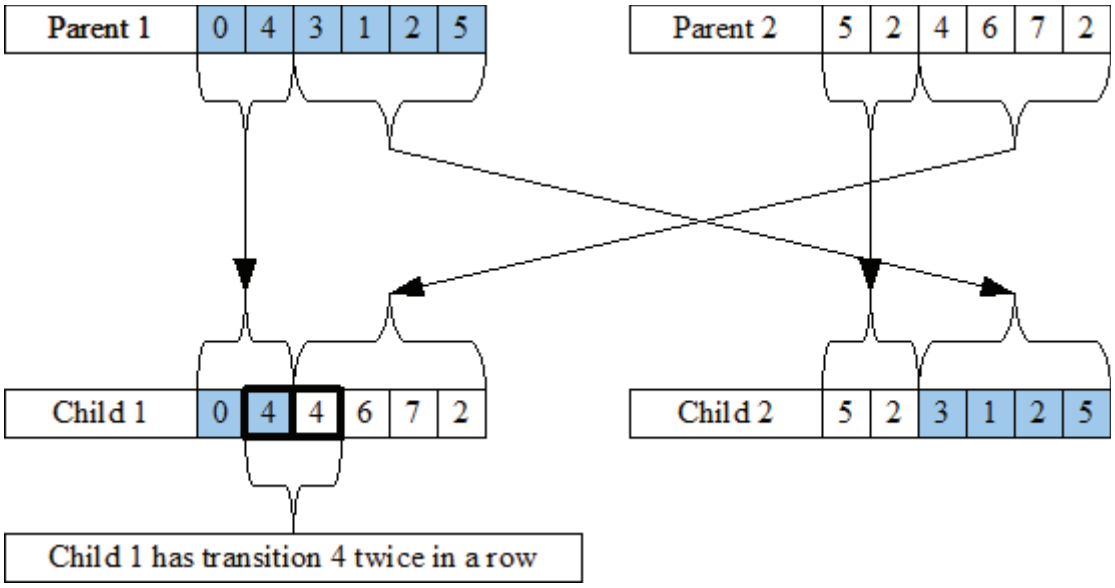

Figure 3.10 – An example of a snake blocker being introduced by single point crossover.

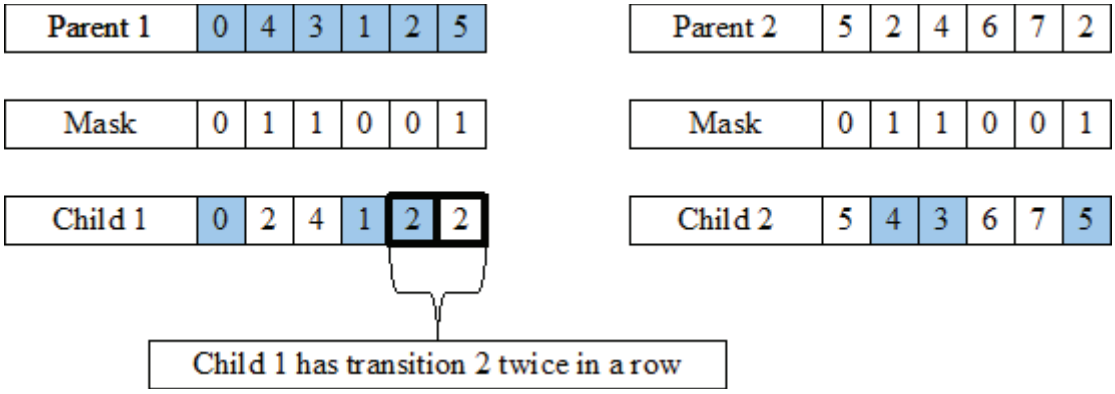

Figure 3.11 – An example of a snake blocker being introduced by uniform crossover.

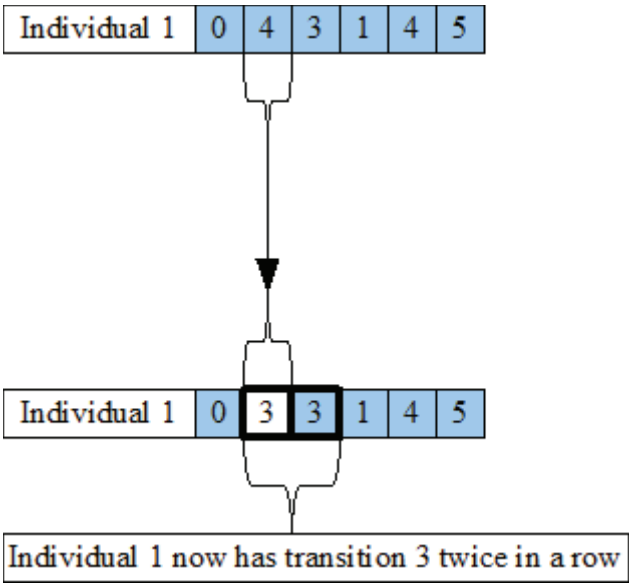

Figure 3.12 – An example of a snake blocker being introduced by random mutation.

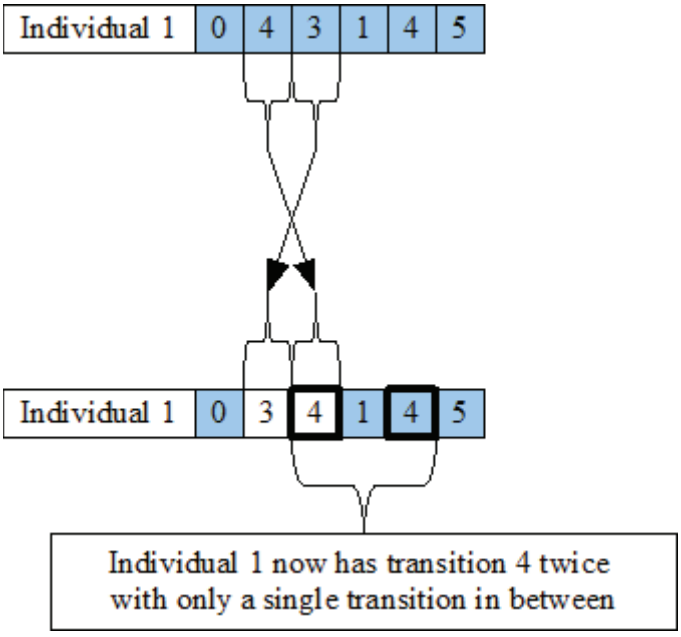

Figure 3.13 – An example of a snake blocker being introduced by Xor mutation.

There are two primary reasons why snake blockers should be addressed. The first one is that they adversely affect the performance of the GA. An argument could be made that snake

blockers are simply a byproduct of the mechanisms of the GA and that the mechanisms should be left alone to sort the problem out. The reasoning being that, by allowing the snake blockers to remain, the fitness of the individual will suffer and the snake blockers will eventually be removed from the gene pool. Unfortunately, the introduction of snake blockers is likely to be far too prevalent.

To illustrate, given two sequential transitions, t_i and t_{i+1}, which are the transitions immediately preceding the point of crossover after which two new transitions will be added, t_{i+2} and t_{i+3}, the probability that t_{i+2} or t_{i+3} will create a snake blocker is $\frac{2d+(d-2)}{d^2}$ for dimension d. In dimension 8, crossover performed at a given point is likely to result in a snake blocker $\frac{2(8)+(8-2)}{8^2}=\frac{22}{64}=34.375\%$ of the time. With snake blockers being introduced over one third of the time a crossover operation is performed, it is unlikely that the GA will be able to sufficiently filter affected individuals.

The second reason to remove snake blockers is that it greatly reduces the search space. The number of transitions that would not generate a snake blocker at any given point in the path past the first two transitions is $d-2$, where d is the current dimension. At the first position, t_1, there are d potentially valid transitions. At the second position, t_2, there are $d-1$ valid transitions left to prevent two identical transitions in succession, $t_1=t_2$. From the third position on, the previous two positions, t_1 and t_2, are occupied by transitions that must be avoided in the current position, leaving $d-2$ options, $t_k \neq t_{k-1}$ and $t_k \neq t_{k-2}$. Allowing snake blockers increases this back to d for every position because the preceding two positions are not taken into account, which changes the effective search space from $d(d-1)(d-2)^{(m-2)}$ to d^m, where m is the length of the individual. The effect on the representation space is significant:

$$8\times7\times6^{95} \approx 4.7\times10^{75}$$

Rather than leave the GA to handle the snake blockers, a second option is to replace the problematic transition with a valid one. This could be accomplished by randomly selecting a replacement transition and swapping out the transgressor with the replacement. While this would be a relatively quick method of resolving the problem, it has the potential to affect the individual's fitness in a negative way. When moving across the chromosome from left to right, the first transition, t_i, where $t_i=t_{i-1}$ or $t_i=t_{i-2}$, has the potential to be the head or the tail of the longest snake in the chromosome. If this is the case, replacing the transition with a different transition could result in the reduction of the size of the snake. The result of this could be devastating to the individual's fitness and prevent it from participating in the creation of the next generation. In effect, this may be no better than leaving the snake blocker where it is.

Instead, experiments done for this thesis use a novel algorithm called Snake Blocker Removal Algorithm (SBRA). The SBRA is designed to have no impact on the snakes in the chromosome. SBRA moves across the snake from left to right searching for snake blockers. When a snake blocker is located, it performs transition reassignment from the location of the snake blocker to the end of the chromosome. The result is that the portion of the snake before the snake blocker remains unchanged while the portion after the snake blocker is replaced with a sequence of transitions from the same equivalence class. The pseudocode for this algorithm is shown in Figure 3.14. Note that a snake blocker is more than one transition, but the location of the snake blocker is understood to mean the last transition in the snake blocker sequence. In other words, whether the sequence is (… $t1$, $t3$, $t1$ …) or (… $t1$, $t1$ …), the second $t1$ is considered to be the location of the snake blocker.

In the case of the first type of snake blocker, having the same two transitions twice in a row, it is relatively easy to see how this would work because there is no potential for interaction

between the transitions to the left of the snake blocker with the transitions to the right of the snake blocker. An example of this type is shown in Figure 3.15.

```
// for the purposes of this example, the dimension is assumed to be known

// get_non_conflicting_transition returns a random transition less than the dimension but not
// equal to any of the arguments
function SBRA (integer[] individual){
        integer length = individual.getLength();

        // for efficiency, snake blockers occurring in the first two positions are handled
        // separately
        if(individual[0] == individual[1] || individual[0] == individual[2]){
                individual[0] = get_non_conflicting_transition(individual[1], individual[2]);
        }
        if(individual[1] == individual[2] || individual[1] == individual[3]){
                individual[1] = get_non_conflicting_transition(individual[0], individual[2],
                        individual[3]);
        }

        // here, the chromosome starts with three different transitions

        transition_map = get_default_transition_map();
        // the default transition map is a  mapping of the transitions to themselves
        //  i.e. t0 → t0, t1 → t1, etc.

        for(index = 2; index < length-1; index++){
                if(
        transition_map[individual[index]] == individual[index-1] // (… ti, ti …)
        or
        transition_map[individual[index]] == individual[index-2] // (… ti, tk, ti …)
                ){
                        replacement = get_non_conflicting_transition(individual[index-2],
individual[index-1], transition_map[individual[index]], transition_map[individual[index+1]]);
                        reassign_transitions(transition_map[individual[index]], replacement);

        // reassign_transitions updates the transition_map for the two transitions given as
        // arguments, but it is based on the targets of the mapping, not the starting points
        // if t0 and t1 are given as arguments and the transition map is currently
        // mapping t3 → t0 and t4 → t1, the end result will be t3 → t1 and t4 → t0

                }

        }

        // here, all of the snake blockers have been removed

}
```

Figure 3.14 – Pseudocode for the Snake Blocker Removal Algorithm.

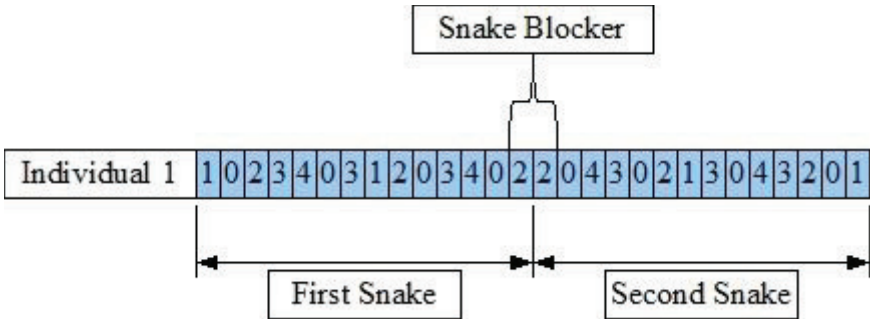

Figure 3.15 – A snake blocker of the first type. The snake blocker prevents interaction between the sequence of transitions to its left with the sequence of transitions to its right.

To remove the snake blocker in Figure 3.15 a replacement transition must be generated. The replacement may be any transition but the ones in the two positions before the second $t2$, $t2$, or the transition immediately following $t2$. For this example, $t0$ and $t2$ are excluded, leaving $t1$, $t3$, and $t4$ as valid selections for a replacement. To be thorough, Figure 3.16 shows the results of picking each of the three valid options. Because the second half of the individual was replaced with a different sequence of transitions from the same equivalence class, the snakes present there have not be disturbed. Figure 3.17 shows that they are all equivalent by mapping each to its canonical representation.

The other case that needs to be handled is a snake blocker of the second type, in which there are two transitions from the same transition class separated by only a single other transition. In this case, the transition in the center of the snake blocker is available for use by both the transitions before the snake blocker and the transitions after the snake blocker. In other words, they have a single transition that they may share. Figure 3.18 shows an example of this type of snake blocker.

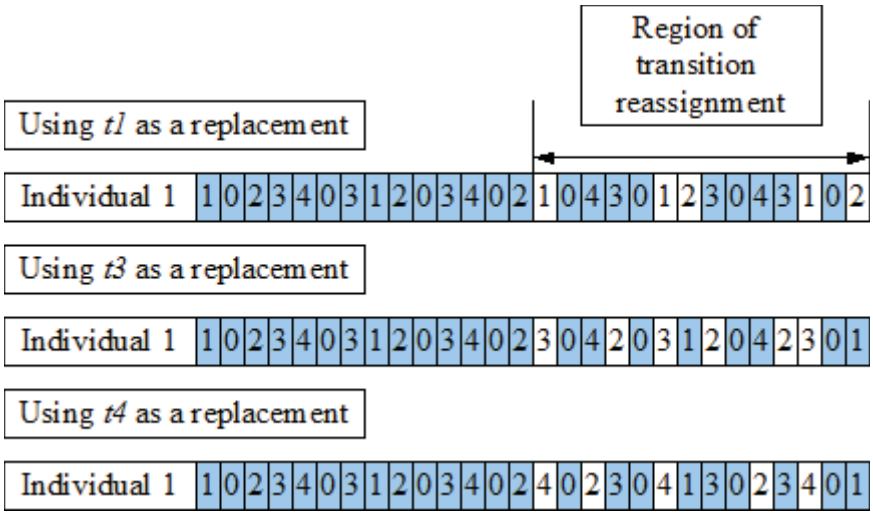

Figure 3.16 – All the possible resolution states for removing the snake blocker in Figure 3.15.

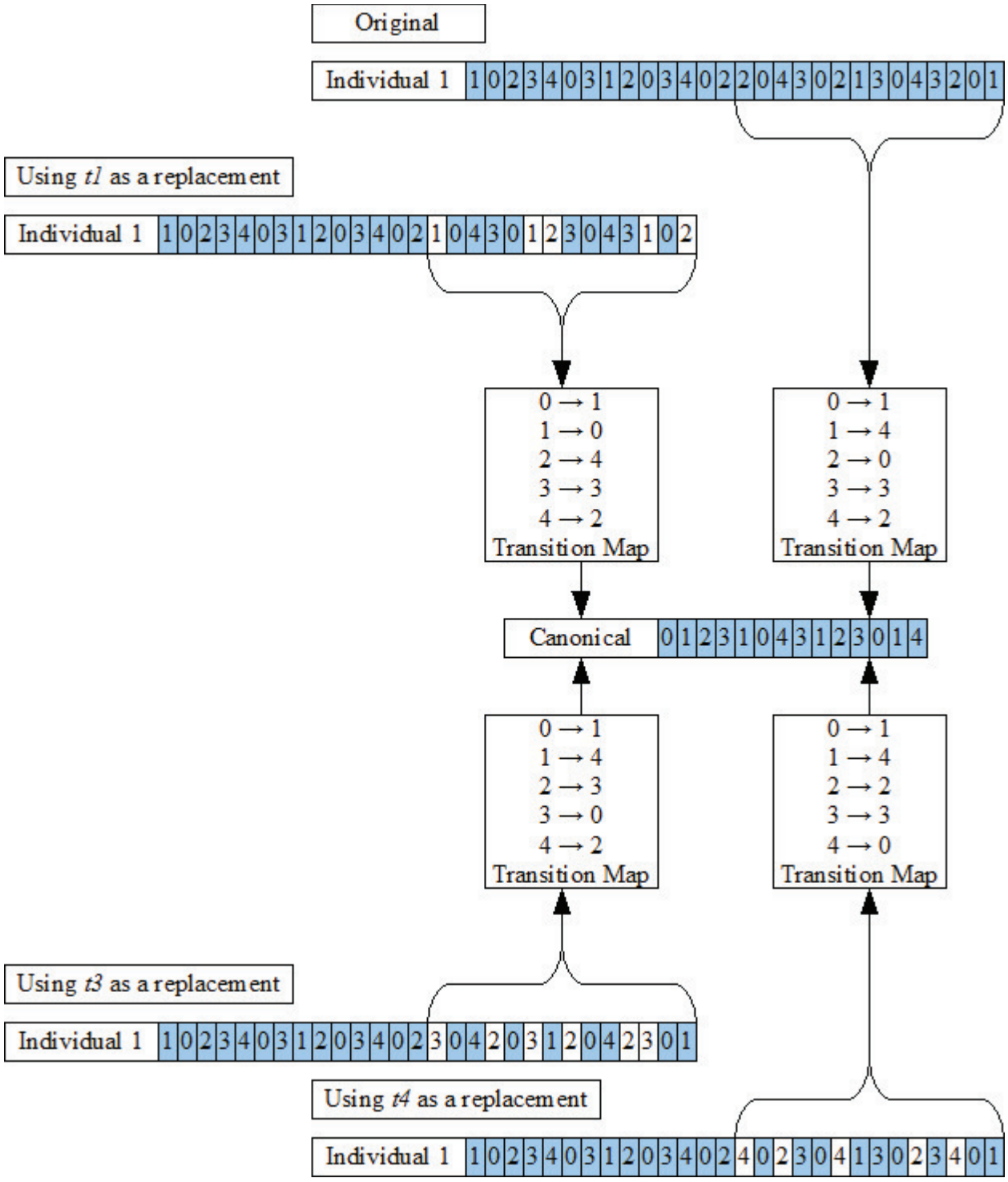

Figure 3.17 – Each of the sections after the transition blocker are compared to prove that they are members of the same equivalence class.

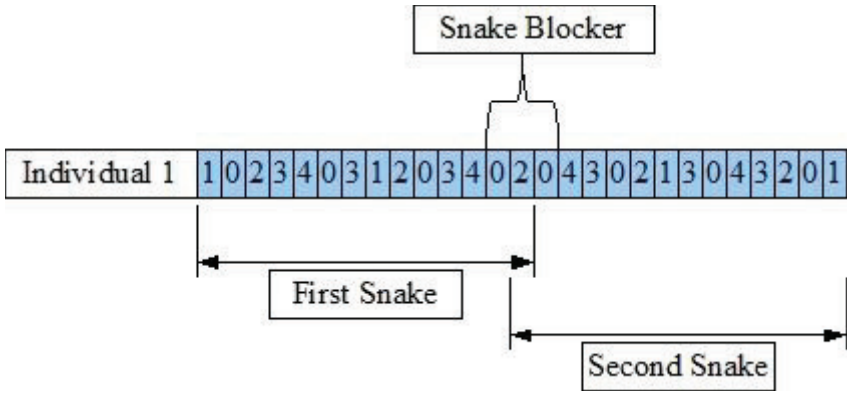

Figure 3.18 – An example of a snake blocker of the second type. The snakes overlap by one vertex.

To remove the snake blocker in Figure 3.18 a replacement transition must be generated. The replacement may be any transition but the ones in the two positions before the second $t0$, $t0$, or the transition immediately following $t0$. For this example, that excludes $t0$, $t2$, and $t4$, leaving $t1$ and $t3$ as valid selections for a replacement. To be thorough, Figure 3.19 shows the results of picking each of the two valid options. Since the transition at the end of the first snake was not changed, the snake before the snake blocker has remained intact. The second snake is also intact because it has been replaced by a different transition sequence from the same equivalence class. Verifying this by mapping it to its canonical representation is left as an exercise for the reader.

This process is applied repeatedly along the length of the chromosome whenever a snake blocker is encountered. As shown, it does not affect existing snakes in the chromosome, but by removing the snake blockers, the total representation space is decreased significantly and there is a chance that the individual will now contain a snake that is longer than it had originally contained.

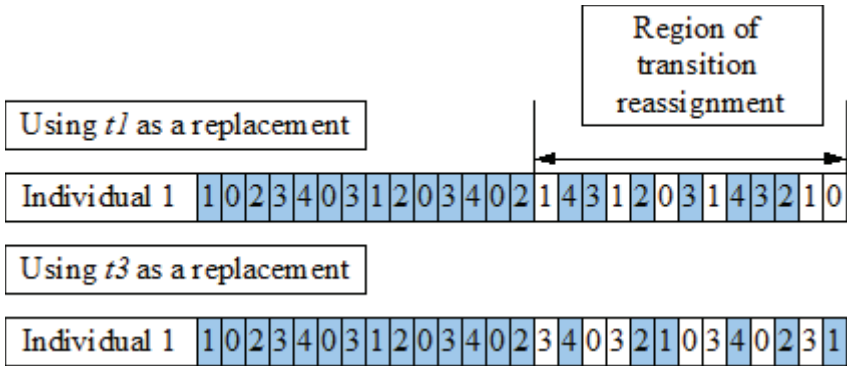

Figure 3.19 – All the possible resolution states for removing the snake blocker in Figure 3.18.

CHAPTER 4

ADDRESSING FITNESS EVALUATION

4.1 COMPACTNESS MEASUREMENTS

A survey of the relevant literature reveals that fitness functions have little variation in the snake problem domain. Seemingly, the most common fitness function is a mapping from the length of the maximum snake present in an individual's chromosome [Potter94, Bitterman04]. The initial inclination is to accept this as an acceptable fitness function; after all, the ultimate goal is to find the longest snake possible. Unfortunately, such a coarsely grained fitness function does not necessarily lend itself well to the evolutionary process as a whole. Fitness based solely on length assigns equal quality to every individual with the same length, even though the genetic material of those individuals may not be equally likely to result in competitive offspring.

Another alternative is to include the concept of tightness. Tightness measures how many vertices in the hypercube are not on the path or part of the snake's skin. If two individuals are of equal length yet one leaves more vertices available, then that individual is said to be tighter. Fitness functions incorporating tightness have an advantage over those utilizing length alone due to an increased level of discrimination of individuals with equal length. That is not to say that fitness functions that use tightness will outperform those using only length, but that adding tightness as a supplemental component may help differentiate between snakes of equal length, which may lead to better results. Furthermore, tightness measures an attribute of the individuals that seems likely to result in better offspring. Tighter individuals occupy less of the hypercube,

leaving more vertices which could be occupied after a GA operation. Unfortunately, tightness is also a measurement that decreases as the size of the snake increases. This property means that tightness must be used as a secondary component to length in the fitness function.

Alternatively, the skin density of the snake could be measured. Skin density is a tally of how many neighbors of the skin vertices are on the path. For each vertex in the snake's skin, the skin density score gets a point for each neighbor that is on the path. For example, if a dimension 3 snake is comprised of two vertices, $v0$ and $v1$, as in Figure 1.9, then the snake's skin would be comprised of $v2$, $v3$, $v4$, and $v5$. Each of the four skin vertices has a single neighbor on the path. Both $v2$ and $v4$ neighbor $v0$, and both $v3$ and $v5$ neighbor $v1$. Therefore, the skin density for this snake would be 4. Skin density has the property of increasing as the length of the snake increases. Therefore, skin density stands a chance of working as a stand-alone fitness measurement.

Skin density is an attractive measurement to include in a fitness function because it provides information about how tightly packed a snake is independent of the snake's length. This is not to say that longer snakes are unlikely to have higher skin density scores, only that there is information contained in the density of the skin vertices that might otherwise be omitted using length alone or tightness, when the tightness scores of two individuals are identical.

Unfortunately, skin density scores are not always more discerning than tightness, which is why experiments conducted for this thesis also use a measurement called Selective Skin Density (SSD) as a supplement to length for evaluation of individual fitness. Using SSD, only skin vertices with more than two neighbors on the path are tallied when calculating the skin density. What makes this measurement more discriminating than the basic skin density

calculation, is that it is capable of giving a finer level of differentiation when comparing individuals.

To illustrate the differences between using tightness, skin density, and selective skin density, the examples in Figure 4.1 will be considered. If tightness were used to compare the three individuals displayed above, S_4^1 and S_4^3 would get scores of 0 because neither leaves any vertices in dimension 4 that are not either on the path or part of the snake's skin. S_4^2 would get a score of 1 because it leaves one vertex, $v13$, unoccupied by either the snake or the snake's skin.

In contrast, when taking the skin vertices into account as listed in Figure 4.2, skin density would score all three individuals equally with scores of 16. Using SSD, in which only skin vertices with at least three neighbors on the path are included, S_4^1 gets a score of 0, S_4^2 gets a score of 3, and S_4^3 gets a score of 6. Interestingly, S_4^3 is the only snake that is not a **maximal** snake in dimension 4, meaning that it is the only snake capable of being extended at one of its ends by another vertex. While this may be coincidental in this example, it serves to illustrate the broader concept that each of these measurements is different. Table 4.1 shows the differences in the rankings assigned to the three individuals based on the measurement used.

Individual 1 - S_4^1 ($v7$, 3, 1, 0, 8, 12, 14) ($t2$, 1, 0, 3, 2, 1)
Individual 2 - S_4^2 ($v7$, 3, 1, 0, 8, 10, 14) ($t2$, 1, 0, 3, 1, 2)
Individual 3 - S_4^3 ($v15$, 7, 3, 1, 0, 8, 10) ($t3$, 2, 1, 0, 3, 1)

Individual 1 Skin Vertices {$v2$, 4, 5, 6, 9, 10, 11, 13, 15}
Individual 1 Free Vertices ∅

Individual 2 Skin Vertices {$v2$, 4, 5, 6, 9, 11, 12, 15}
Individual 2 Free Vertices {$v13$}

Individual 3 Skin Vertices {$v2$, 4, 5, 6, 9, 11, 12, 13, 14}
Individual 3 Free Vertices ∅

Figure 4.1 – Three examples of dimension 4 individuals.

$v2 \leftrightarrow \{v0, 3\}$ $v6 \leftrightarrow \{v7, 14\}$ $v11 \leftrightarrow \{v3\}$
$v4 \leftrightarrow \{v0, 12\}$ $v9 \leftrightarrow \{v1, 8\}$ $v13 \leftrightarrow \{v12\}$
$v5 \leftrightarrow \{v1, 7\}$ $v10 \leftrightarrow \{v8, 14\}$ $v15 \leftrightarrow \{v7, 14\}$
Individual 1

$v2 \leftrightarrow \{v0, 3, 10\}$ $v6 \leftrightarrow \{v7, 14\}$ $v12 \leftrightarrow \{v8, 14\}$
$v4 \leftrightarrow \{v0\}$ $v9 \leftrightarrow \{v1, 8\}$ $v15 \leftrightarrow \{v7, 14\}$
$v5 \leftrightarrow \{v1, 7\}$ $v11 \leftrightarrow \{v3, 10\}$
Individual 2

$v2 \leftrightarrow \{v0, 3, 10\}$ $v6 \leftrightarrow \{v7\}$ $v12 \leftrightarrow \{v8\}$
$v4 \leftrightarrow \{v0\}$ $v9 \leftrightarrow \{v1, 8\}$ $v13 \leftrightarrow \{v15\}$
$v5 \leftrightarrow \{v1, 7\}$ $v11 \leftrightarrow \{v3, 10, 15\}$ $v14 \leftrightarrow \{v10, 15\}$
Individual 3

Figure 4.2 – The skin vertices from each of the individuals from Figure 4.1. The vertices in curly braces are the snake vertices that neighbor each skin vertex.

Table 4.1 – The ranking of each individual according to the fitness measurement used.

Rank	Tightness	Skin Density	SSD
1	S_4^2	S_4^1, S_4^2, S_4^3	S_4^3
2	S_4^1, S_4^3		S_4^2
3			S_4^1

4.2 TARGET FREQUENCY DISTRIBUTION

One of the components analyzed for performance as a fitness measurement was the frequency with which transition classes where present in the chromosome. The motivation for this approach comes from the observation that the best known dimension 8 individual, as shown in [Rajan99], that has already been found, contains a length 50 snake from dimension 7 as a subsequence. Because this length 97 snake starts with a dimension 7 snake, the first 50 transitions do not use transition 7. In fact, of the 97 transitions, only one is $t7$. When the

transition classes are broken down by frequency of occurrence, there is a strong preference for traversing transition 3.

To test if the frequency distribution of an individual is similar to that of the target distribution, the individual is converted into its FBTR representation. This means that the frequency with which transition 0 occurs is greater than or equal to that of transition 1 and so on. The frequencies are then summed together for each transition class to get a total distribution. The distribution for the individual is then scaled and compared to the target distribution. In the example shown in Figure 4.3, the individual has 27 transitions, whereas the target distribution is only 13 transitions. To compare these, the distribution values for the individual are scaled. To get the distribution value for the fitness function, the absolute differences in the transition frequencies are summed. Figure 4.4 shows these calculations.

Individual 1	2	0	3	1	4	0	1	2	3	0	1	4	0	3	2	4	1	0	3	2	1	0	4	1	3	0	2

Target Distribution	*t0*	*t1*	*t2*	*t3*	*t4*	Sum
	4	3	3	2	1	13

Individual Distribution	*t0*	*t1*	*t2*	*t3*	*t4*	Sum
	7	6	5	5	4	27

Figure 4.3 – An example of a dimension 5 individual along with its transition frequency distribution. The target distribution for dimension 5 is also shown.

$$\text{Target Distribution Factor} = \left|\frac{7\times 13}{27} - 4\right| + \left|\frac{6\times 13}{27} - 3\right|$$

$$+ \left|\frac{5\times 13}{27} - 3\right| + \left|\frac{5\times 13}{27} - 2\right| + \left|\frac{4\times 13}{27} - 1\right| \approx 2.67$$

Figure 4.4 – The calculation of the target distribution factor for the individual shown in Figure 4.3.

The idea for using the frequency distribution as a target of the fitness function is designed to test whether the frequency with which transition classes are traversed is an indicator of how long the snakes in those individuals may be. If this does serve as a good indicator in the fitness function, then the only remaining obstacle would be determining what distribution to target.

For the testing done in this project, the distribution from the length 97 individual listed above was used. However, it cannot be assumed that there will be good individuals to serve as examples in the dimension that is being searched. In those situations, it may be possible to extrapolate from lower dimensions. The results of a comparison between the target distributions in dimensions 3 through 8, after being normalized, are shown in Figure 4.5. The overall trend indicates that the best individuals in dimensions 3 through 7, and the best known individual in dimension 8, have a tendency to use some transition classes more than others. Figure 4.6 shows only dimensions 5 through 8 to reduce some of the noise present in Figure 4.5 and Figure 4.7 shows dimensions 7 and 8 for a one-on-one comparison.

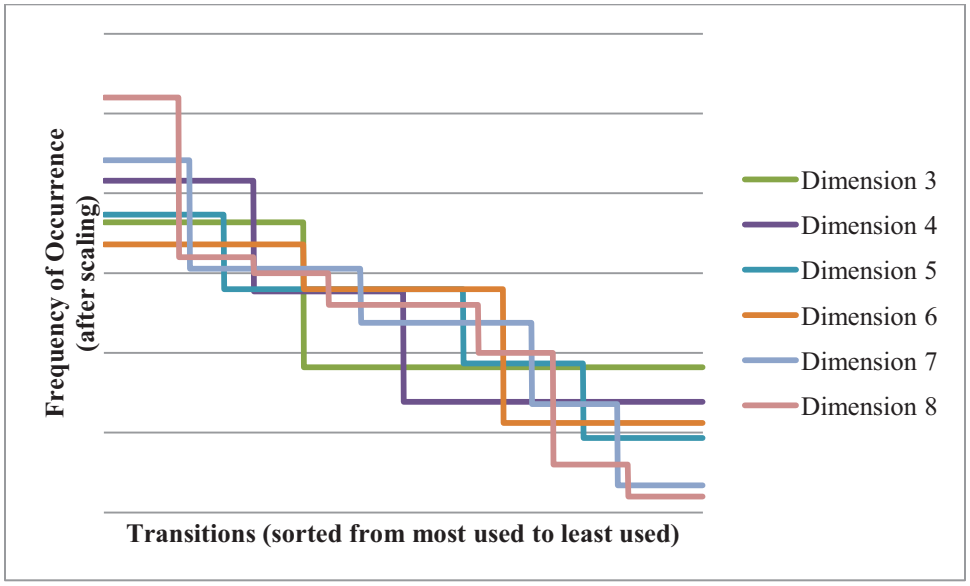

Figure 4.5 – The target distributions for dimensions 3 through 8.

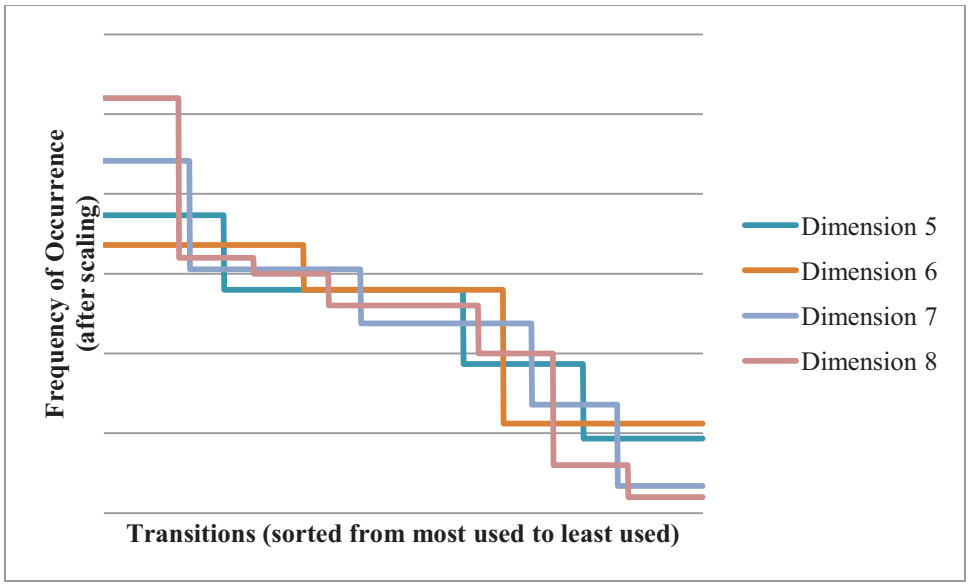

Figure 4.6 – The target distributions for dimensions 5 through 8.

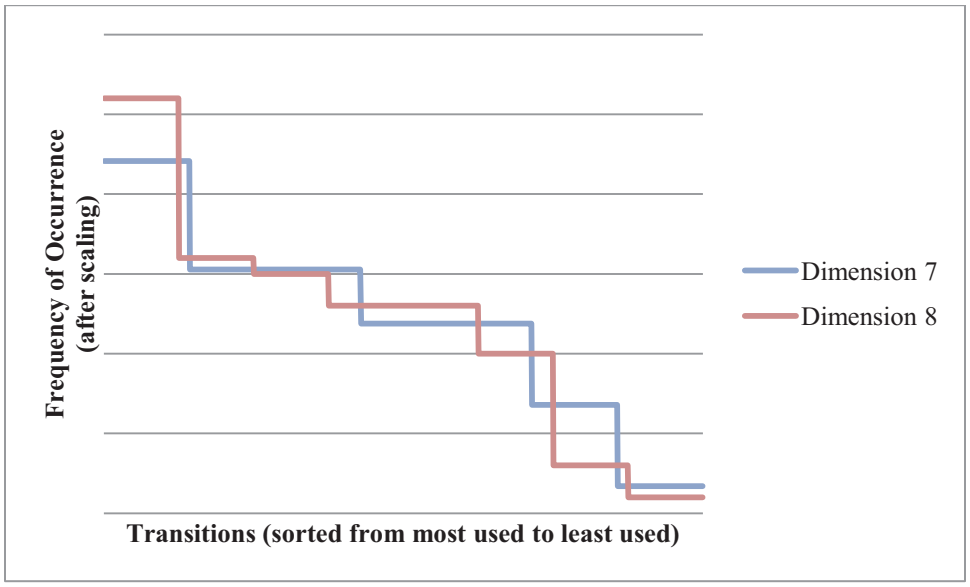

Figure 4.7 – The target distributions for dimensions 7 and 8.

CHAPTER 5

TESTING AND RESULTS

5.1 SETUP

To test the relative merits of the different representations and the parameters that were being used in the GA, experimentation was performed by running a series of head-to-head competitions. The GA was designed to allow multiple, mutually exclusive populations to exist. Each population shared the same crossover, mutation, and selection settings for each run. The GA was set to run until every population stopped making forward progress, measured in terms of maximum or average individual fitness, for ten consecutive generations.

When the GA halted, the results indicated which populations outperformed the others. Those individual configurations that consistently outperformed the others were taken to be superior. Additionally, this scheme ensured that each population was run for an equal number of generations, for a given set of parameters, thereby preventing bias against any population that had a slower convergence rate. Each population was given an equal chance to discover the longest snake for a run.

All populations were of size 500 and all used elitism, moving the top 10 individuals from each generation directly into the next generation. Table 5.1 shows all the available parameters that were tested. Table 5.2 shows all of the fitness functions that were available for experimentation.

Table 5.1 – All of the components and parameters that were available for testing in this thesis.

Component	Options
Representations	Transition-based, Canonical, FBTR
Initialization	RRI, RHI
Snake blockers	Allowed, Removed (SBRA)
Fitness Functions	Length, Tightness, Skin Density, SSD, Target Distribution
Crossover	Single Point, Double Point, Triple Point, Uniform (5%, 10%, 30%)
Mutation	Random, Xor
Mutation Frequency	0.5%, 1%, 5%
Selection	Random, Rank-Based, Roulette Wheel, Tournament

5.2 METRICS

Several key indicators were used to compare the results after running competitions. The three that were chosen as primary indicators are average maximum length, average generations to converge, and average best individual length increase. **Average maximum length** is the average length of the longest snake found across all runs. This measurement indicates, on average, how long the snakes were that were found with a given technique. **Average generations to converge** is the average number of generations the population took to reach the maximum length snake that was found. **Average best individual length increase** is the average number of transitions difference there were between the longest snake in the first generation and the longest snake in the last generation.

Table 5.2 – All of the fitness functions that were used in experimentation. In general, when two components were used, they were given roughly half the total weight of the calculation. When three were used, they were either weighted equally or one of the components was given half the total weight. The fitness codes are designed to indicate which of the three components was weighted more than the other two by placing it first.

Fitness Code	Definition
F_L	Length Only
F_T	Tightness Only
F_{SD}	Skin Density Only
F_{SSD}	Selective Skin Density Only
F_D	Target Distribution Only
F_{L_D}	½ Length, ½ Target Distribution
$F_{L\_SSD\_D\_Equal}$	⅓ Length, ⅓ Selective Skin Density, ⅓ Target Distribution
$F_{L\_SSD\_D}$	½ Length, ¼ Selective Skin Density, ¼ Target Distribution
$F_{SSD\_L\_D}$	¼ Length, ½ Selective Skin Density, ¼ Target Distribution
$F_{D\_SSD\_L}$	¼ Length, ¼ Selective Skin Density, ½ Target Distribution
F_{L_SSD}	½ Length, ½ Selective Skin Density
F_{SSD_D}	½ Selective Skin Density, ½ Target Distribution
$F_{L\_SD\_D\_Equal}$	⅓ Length, ⅓ Skin Density, ⅓ Target Distribution
$F_{L\_SD\_D}$	½ Length, ¼ Skin Density, ¼ Target Distribution
$F_{SD\_L\_D}$	¼ Length, ½ Skin Density, ¼ Target Distribution
$F_{D\_SD\_L}$	¼ Length, ¼ Skin Density, ½ Target Distribution
F_{L_SD}	½ Length, ½ Skin Density
F_{SD_D}	½ Skin Density, ½ Target Distribution
$F_{L\_T\_D\_Equal}$	⅓ Length, ⅓ Tightness, ⅓ Target Distribution
$F_{L\_T\_D}$	½ Length, ¼ Tightness, ¼ Target Distribution
$F_{T\_L\_D}$	¼ Length, ½ Tightness, ¼ Target Distribution
$F_{D\_T\_L}$	¼ Length, ¼ Tightness, ½ Target Distribution
F_{L_T}	½ Length, ½ Tightness
F_{T_D}	½ Tightness, ½ Target Distribution

5.3 SNAKE BLOCKERS

In order to observe the effects of removing snake blockers, tests were run comparing populations in which snake blockers were allowed against populations where snake blockers were being removed using SBRA. Twelve populations were used; half allowed snake blockers and half removed them. After 54 competitions, all six of the populations using SBRA occupied the top six rankings. The maximum length snake found during this experiment was 73.

Table 5.3 – The parameters used to test SBRA.

Experiment Parameters	
Component	Options
Representations	FBTR
Initialization	RRI
Snake blockers	Allowed, Removed (SBRA)
Fitness Functions	$F_{L\_SSD\_D\_Equal}$, $F_{L\_SSD\_D}$, F_{L}, F_{L_SSD}, F_{SSD}
Crossover	Single Point
Mutation	Random, Xor
Mutation Frequency	0.5%, 1%, 5%
Selection	Tournament

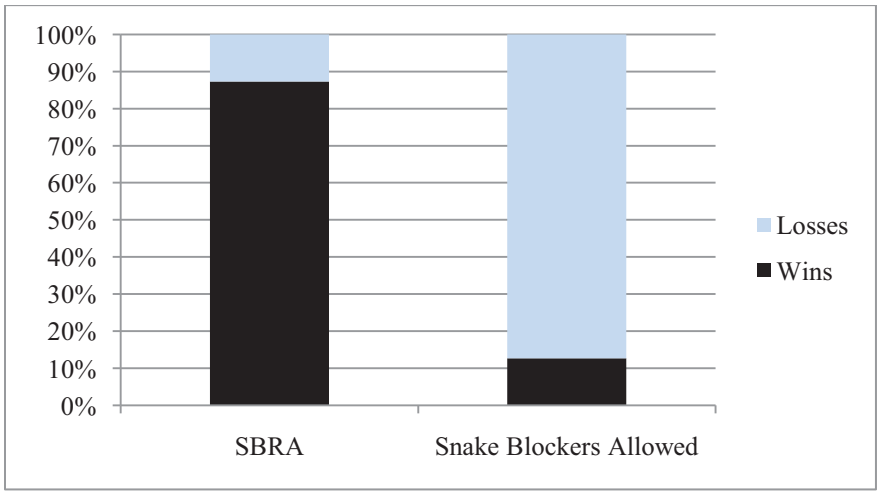

Figure 5.1 – The percentage of competitions won by populations using SBRA versus those populations allowing snake blockers.

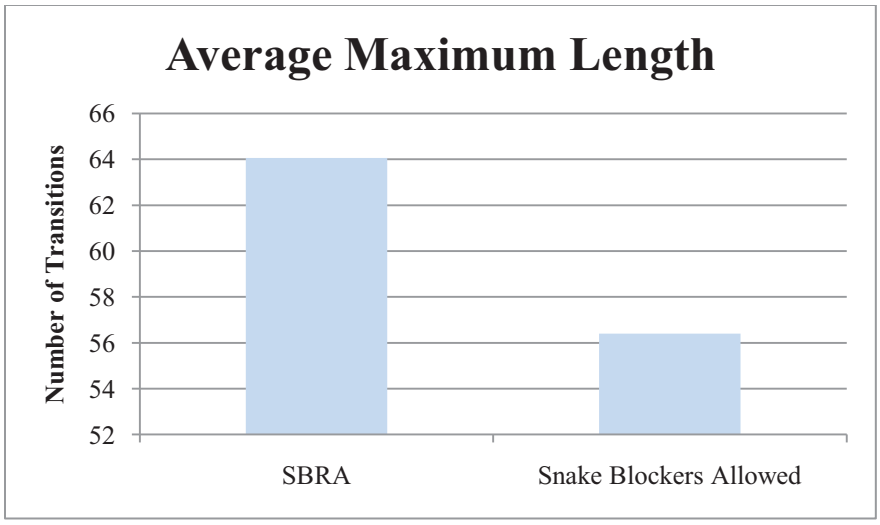

Figure 5.2 – The average maximum length snake found in competitions between populations using SBRA versus those populations allowing snake blockers.

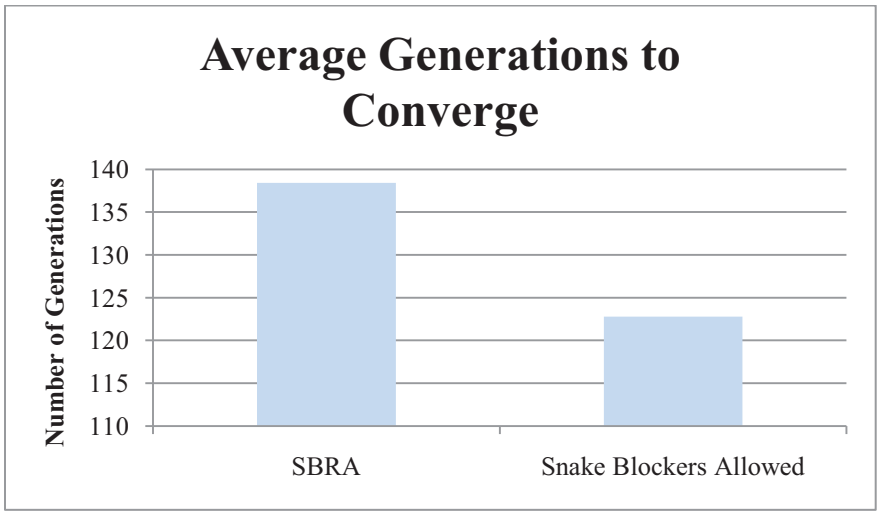

Figure 5.3 – The average generations to converge for populations using SBRA versus those populations allowing snake blockers.

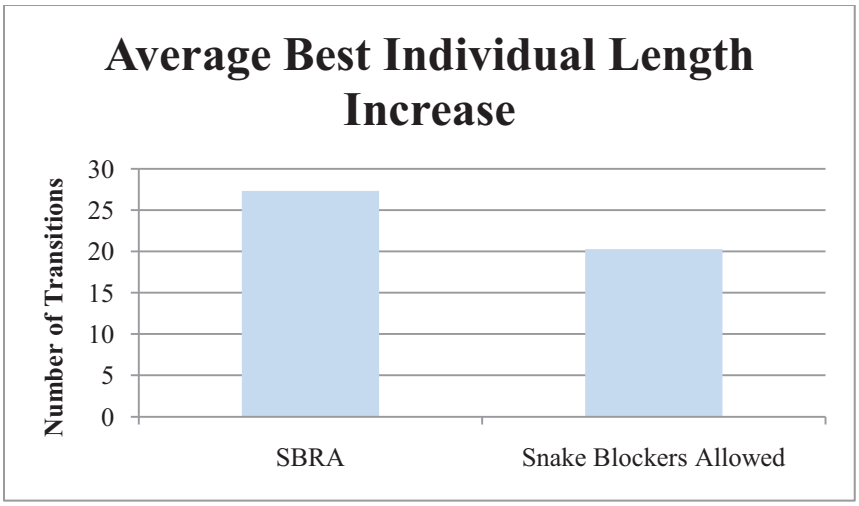

Figure 5.4 – The average increase of the maximum length snake from the first generation until the last in populations using SBRA versus those populations allowing snake blockers.

5.4 REPRESENTATION

Initial testing was done to compare FBTR to a standard transition-based representation. Ten populations were set up to use a transition-based representation and ten populations were set up to use FBTR. After 600 competitions, FBTR won 61.16% of the encounters. Furthermore, on average, FBTR found significantly longer snakes. The maximum length snake found by FBTR for these runs was 74 and the maximum length snake found by the populations using a transition-based representation was 76.

Table 5.4 – The parameters used to compare a transition-based representation to FBTR.

Experiment Parameters	
Component	Options
Representations	Transition-based, FBTR
Initialization	RRI
Snake blockers	Removed (SBRA)
Fitness Functions	$F_{L\_SSD\_D\_Equal}$, $F_{L\_SSD\_D}$, F_L, F_{L_SSD}, F_{SSD}, F_{SSD_D}, $F_{D\_SSD\_L}$, F_{L_D}, $F_{SSD\_L\_D}$, F_D
Crossover	Single Point
Mutation	Random, Xor
Mutation Frequency	0.5%, 1%, 5%
Selection	Tournament

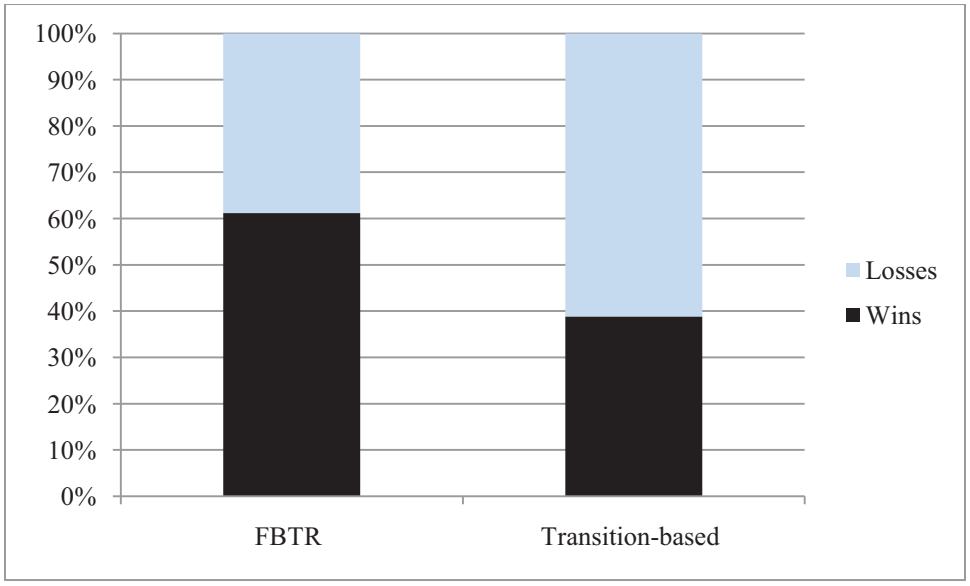

Figure 5.5 – The percentage of competitions won by populations using FBTR versus those populations using a transition-based representation.

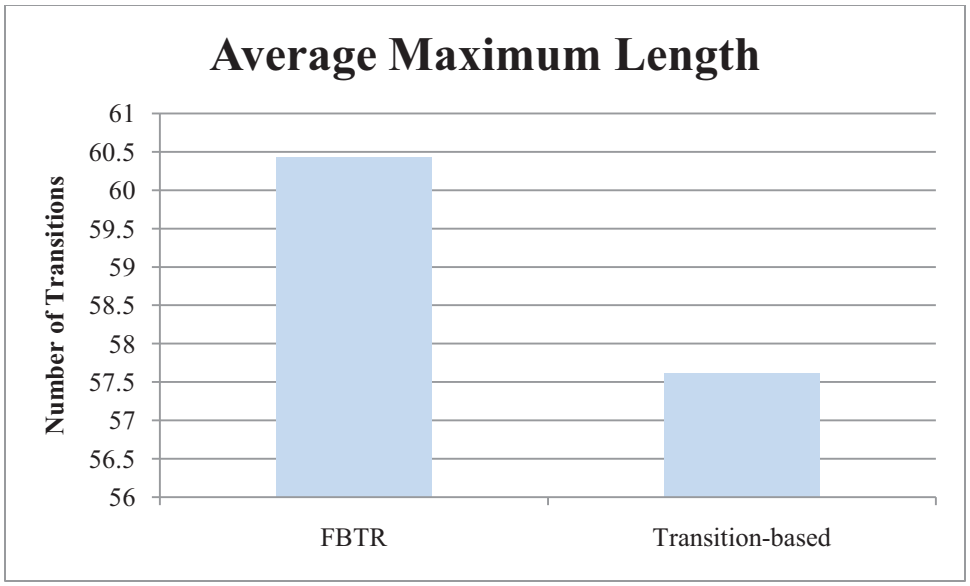

Figure 5.6 – The average maximum length snake found in competitions between populations using FBTR versus those populations using a transition-based representation.

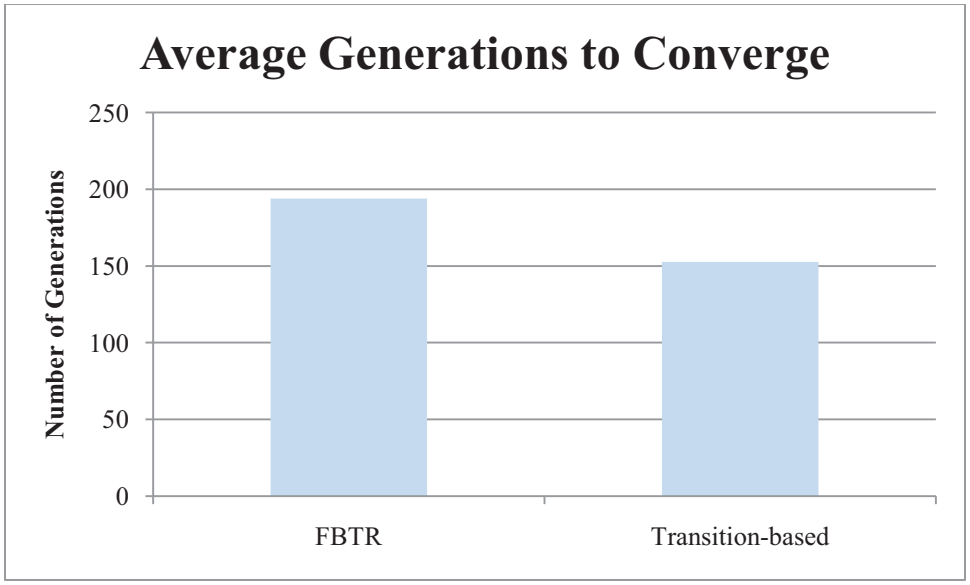

Figure 5.7 – The average generations to converge for populations using FBTR versus those populations using a transition-based representation.

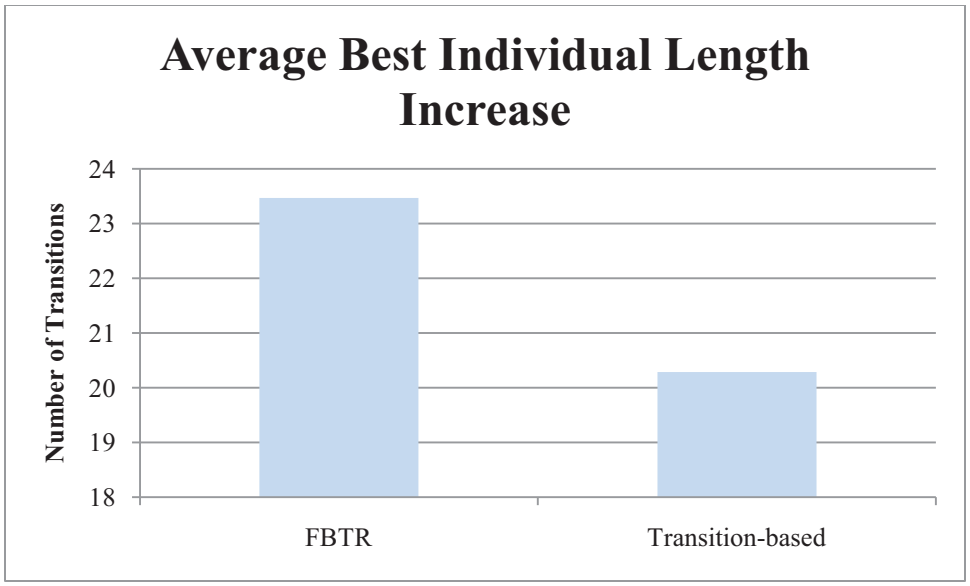

Figure 5.8 – The average increase of the maximum length snake from the first generation until the last in populations using FBTR versus those populations using a transition-based representation.

Subsequently, testing concentrated on comparing FBTR to canonical representation. Ten populations were set up to use a canonical representation and ten populations were set up to use FBTR. After 1800 competitions, FBTR won 890 of the encounters and canonical representation won 910. The average performance of both representations was relatively comparable. FBTR generated slightly longer snakes on average and had a slightly larger total increase in the maximum snake length than canonical representation. The maximum length snake found by FBTR for these runs was 78 and the maximum length snake found by the populations using a canonical representation was 74.

Table 5.5 – The parameters used to compare a canonical representation to FBTR.

Experiment Parameters	
Component	Options
Representations	Canonical, FBTR
Initialization	RRI
Snake blockers	Removed (SBRA)
Fitness Functions	$F_{L\_SSD\_D\_Equal}$, $F_{L\_SSD\_D}$, F_L, F_{L_SSD}, F_{SSD}, F_{SSD_D}, $F_{D\_SSD\_L}$, F_{L_D}, $F_{SSD\_L\_D}$, F_D
Crossover	Single Point
Mutation	Random, Xor
Mutation Frequency	0.5%, 1%, 5%
Selection	Tournament

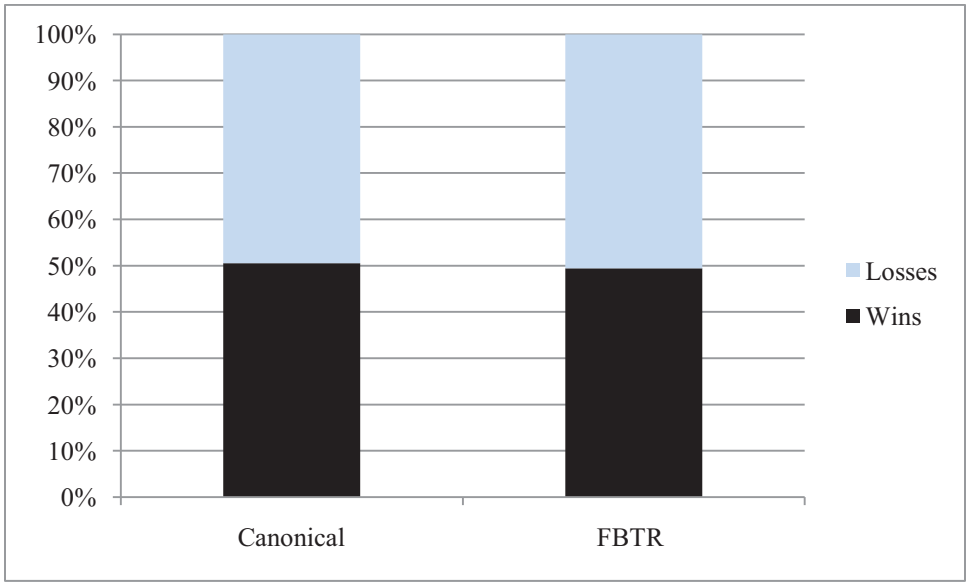

Figure 5.9 – The percentage of competitions won by populations using FBTR versus those populations using a canonical representation.

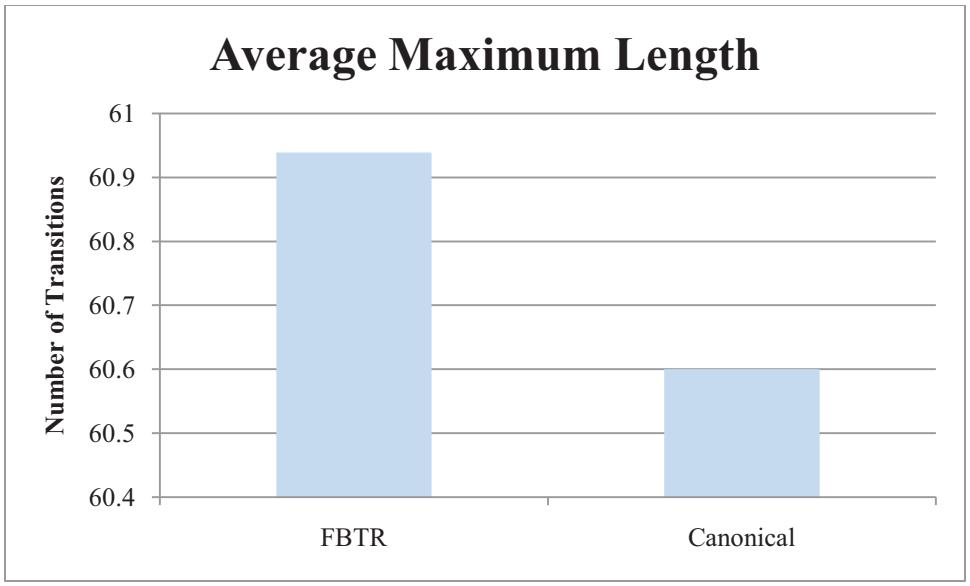

Figure 5.10 – The average maximum length snake found in competitions between populations using FBTR versus those populations using a canonical representation.

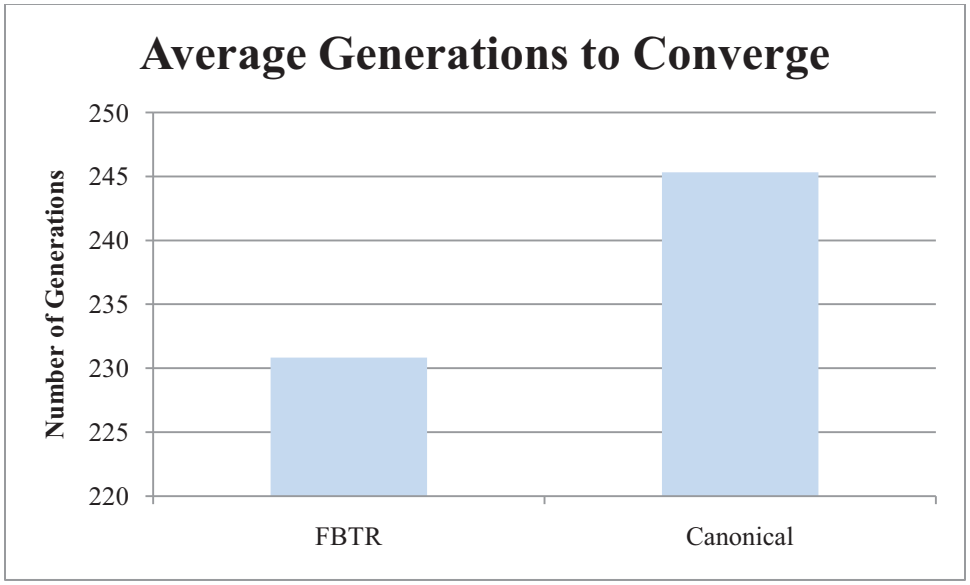

Figure 5.11 – The average generations to converge for populations using FBTR versus those populations using a canonical representation.

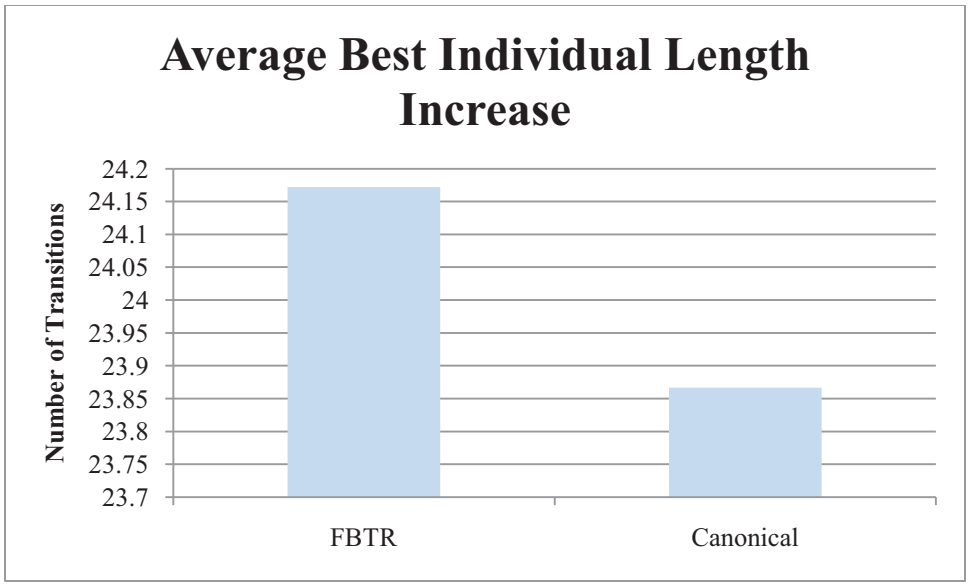

Figure 5.12 – The average increase of the maximum length snake from the first generation until the last in populations using FBTR versus those populations using a canonical representation.

5.5 INITIALIZATION

Experimentation with the two initialization methods led to some interesting results. Initialization is the only GA operator that was not compared head to head. All of the results from the other GA experiments were run using RRI because RHI tends to generate snakes that are too large for the GA to effectively make any progress. After 630 populations were run after being initialized with RHI, the average maximum length snake found was 71.1, but the average increase over the course of the runs was only 0.34 added vertices. In other words, on average, the GA did not increase the length of the longest snake by even one transition from start to finish. Apparently, populations consisting of many individuals that are at or near local maxima are less likely to produce offspring that are better than the parents. It is possible that the partial solutions contained in the individuals were not as compatible for building better snakes as the partial solutions that are present in a GA that has been run from a much lower starting point. The maximum length snake found during these runs was 79.

Table 5.6 – The parameters used to test RHI as an initialization technique.

Experiment Parameters	
Component	Options
Representations	FBTR
Initialization	RHI
Snake blockers	Removed (SBRA)
Fitness Functions	$F_{L\_SSD\_D\_Equal}$, $F_{L\_SSD\_D}$, F_{L}, F_{L_SSD}, F_{SSD}, F_{SSD_D}, $F_{D\_SSD\_L}$, F_{L_D}, $F_{SSD\_L\_D}$, F_{D}
Crossover	Single Point, Double Point, Triple Point, Uniform
Mutation	Xor
Mutation Frequency	0.5%, 1%, 5%
Selection	Rank-Based, Roulette Wheel, Tournament

5.6 FITNESS

Because fitness functions have so much impact on the performance of the GA, particular effort was devoted toward exploring fitness functions. Five components to the fitness function were used: the length of the longest snake in an individual, tightness, skin density, selective skin density, and how closely the individual matches a target distribution. In order to get an accurate representation of how the selected components of the fitness function impacted the performance of the GA as a whole, all of the fitness functions listed in Table 5.2 were used, each placing different emphasis on the components.

To evaluate these fitness functions, each was run head-to-head against the others in 144 competitions. The top three were F_{SSD}, $F_{L\_SSD\_D\_Equal}$, and then $F_{L\_T\_D}$. In general, length and selective skin density were the best indicators of individual fitness. Skin density and tightness were not effective as independent measurements. The maximum length snake found by this round of experimentation was 76 and belonged to a population using F_{SSD_D}.

Table 5.7 – The parameters used to compare a transition-based representation to FBTR.

Experiment Parameters	
Component	Options
Representations	Transition-based, FBTR
Initialization	RRI
Snake blockers	Removed (SBRA)
Fitness Functions	All fitness functions listed in Table 5.2.
Crossover	Single Point
Mutation	Random, Xor
Mutation Frequency	0.5%, 1%, 5%
Selection	Tournament

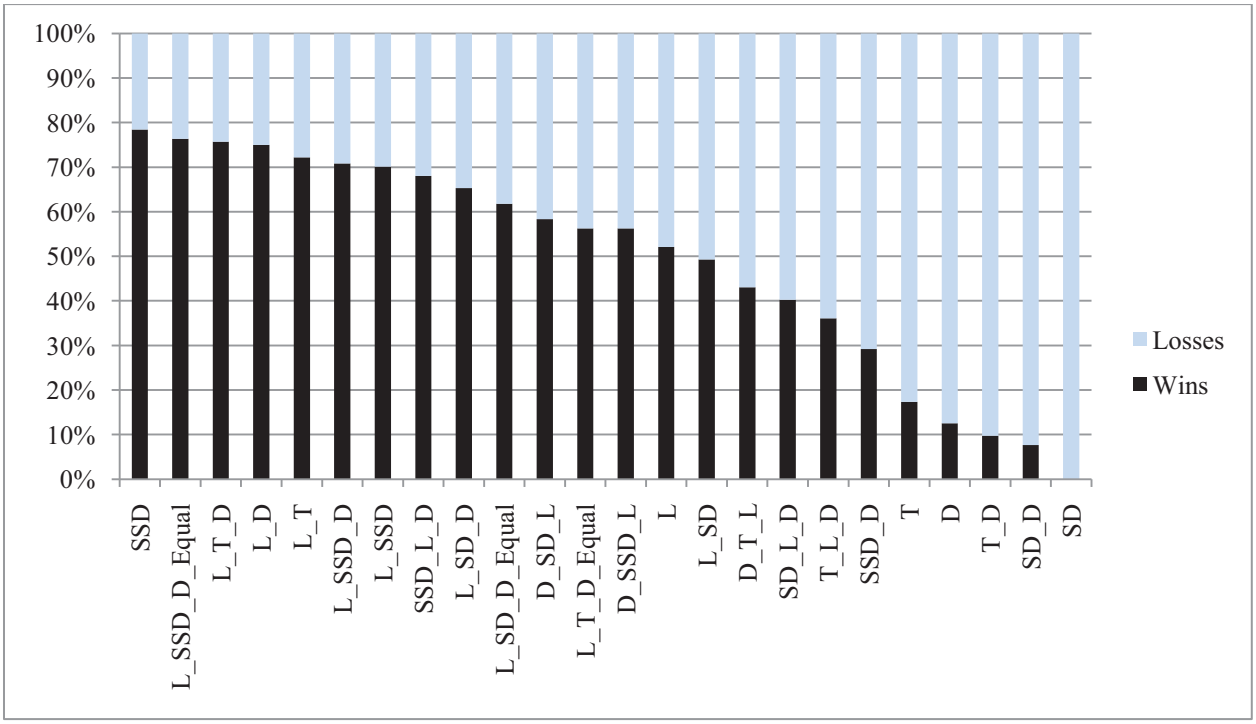

Figure 5.13 – The percentage of competitions won by populations using each fitness function available.

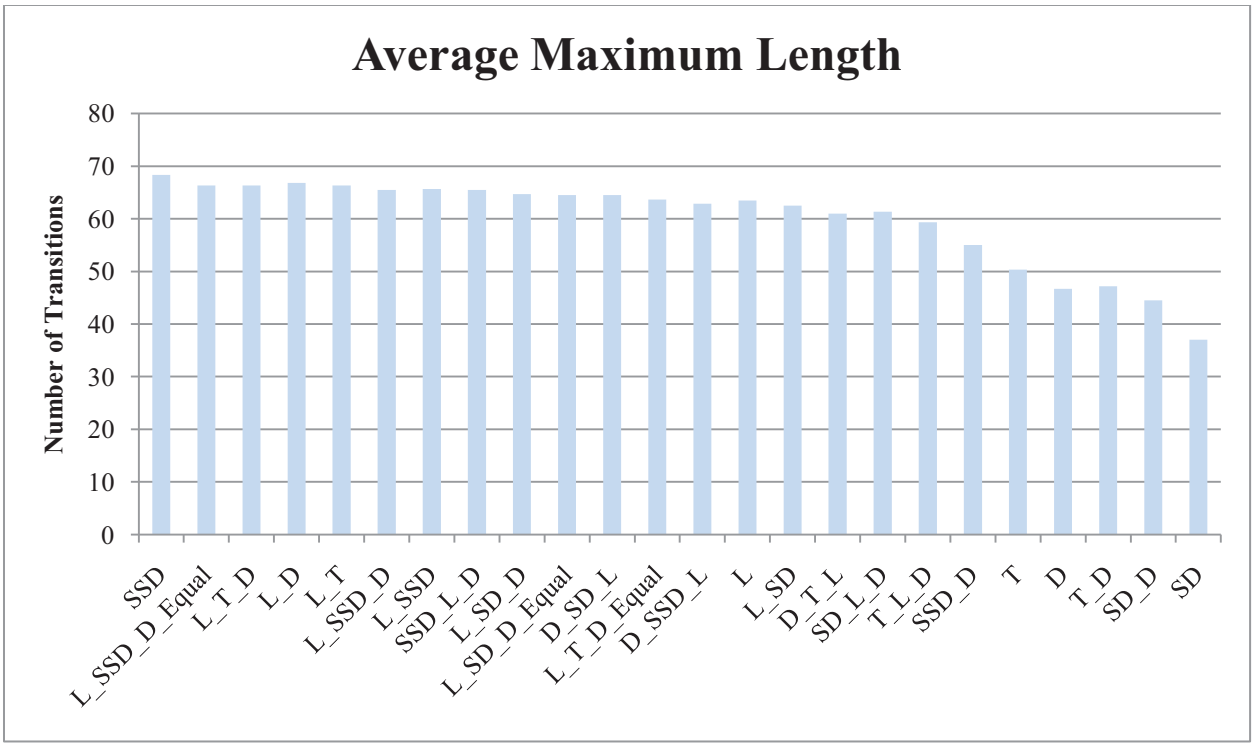

Figure 5.14 – The average maximum length snake found in competitions between populations using each fitness function available.

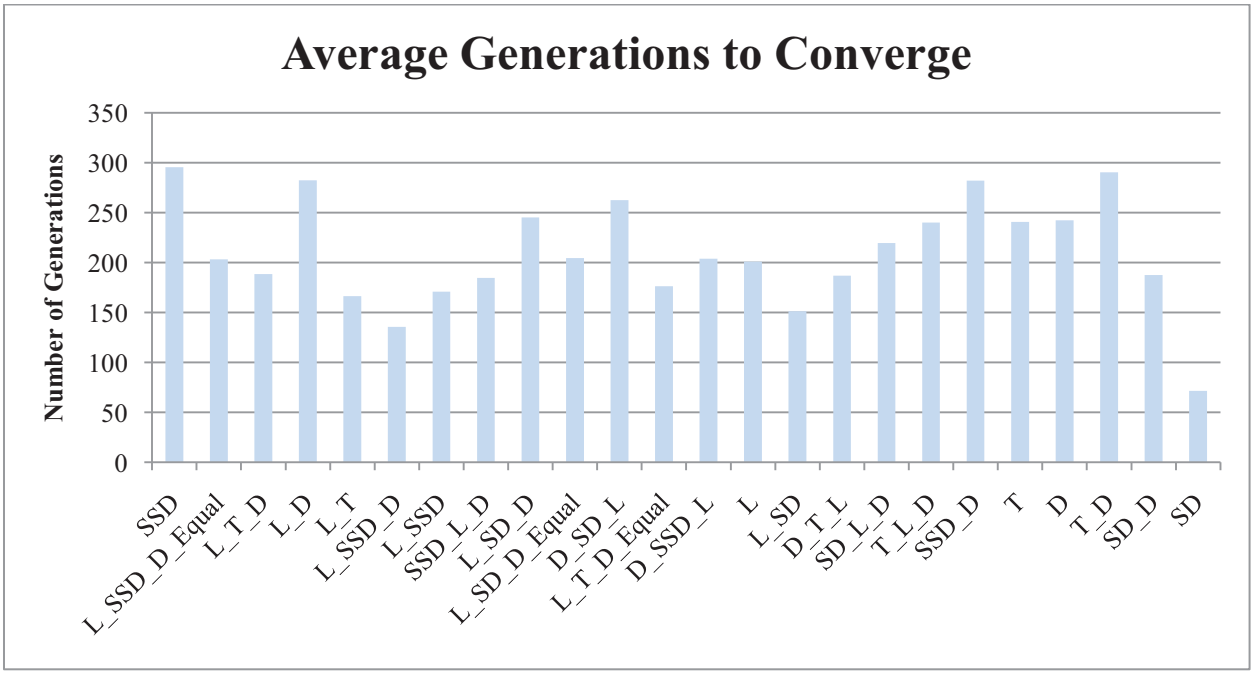

Figure 5.15 – The average generations to converge for populations using each fitness function available.

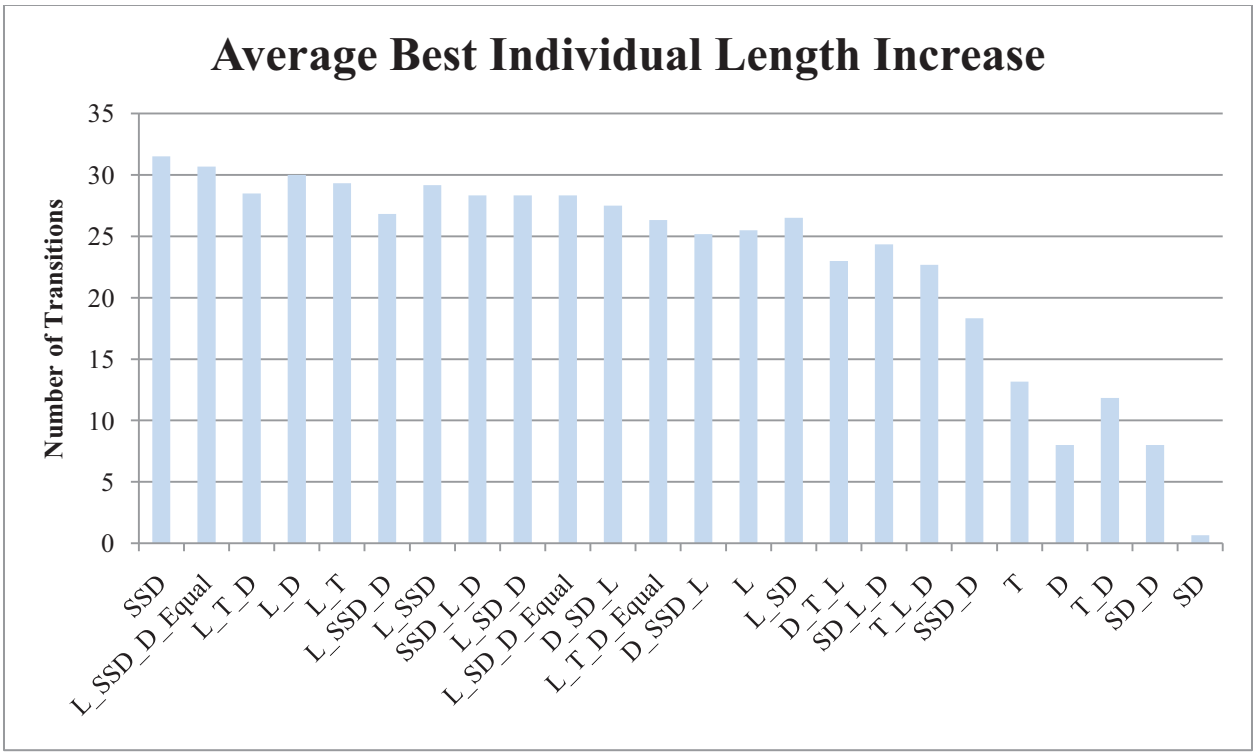

Figure 5.16 – The average increase of the maximum length snake from the first generation until the last in populations using each fitness function available.

5.7 SELECTION OPERATORS

Experiments for this thesis used four selection operators: random selection, rank-based selection, roulette wheel selection, and tournament selection. The performance of the selection operators was tested in 540 head-to-head competitions for each operator. Tournament selection had the best performance. Roulette wheel selection led rank-based selection, but both had mediocre performance overall. Random selection performed rather poorly. The best snake found in this experiment was a length 70 snake, which was found using tournament selection.

Table 5.8 – The parameters used to compare the selection operators.

Experiment Parameters	
Component	Options
Representations	FBTR
Initialization	RRI
Snake blockers	Removed (SBRA)
Fitness Functions	$F_{L\_SSD\_D\_Equal}$, $F_{L\_SSD\_D}$, F_{L}, F_{L_SSD}, F_{SSD}, F_{SSD_D}, $F_{D\_SSD\_L}$, F_{L_D}, $F_{SSD\_L\_D}$, F_{D}
Crossover	Single Point, Double Point, Triple Point
Mutation	Random, Xor
Mutation Frequency	0.5%, 1%, 5%
Selection	Random, Rank-Based, Roulette Wheel, Tournament

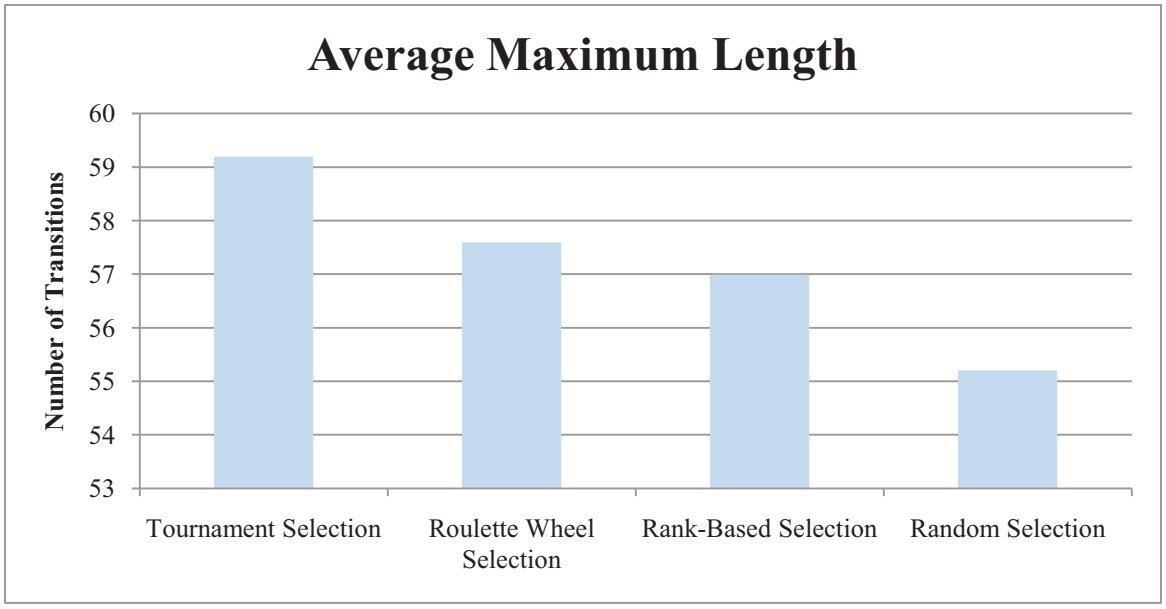

Figure 5.17 – The average maximum length snake found in competitions between the four selection operators.

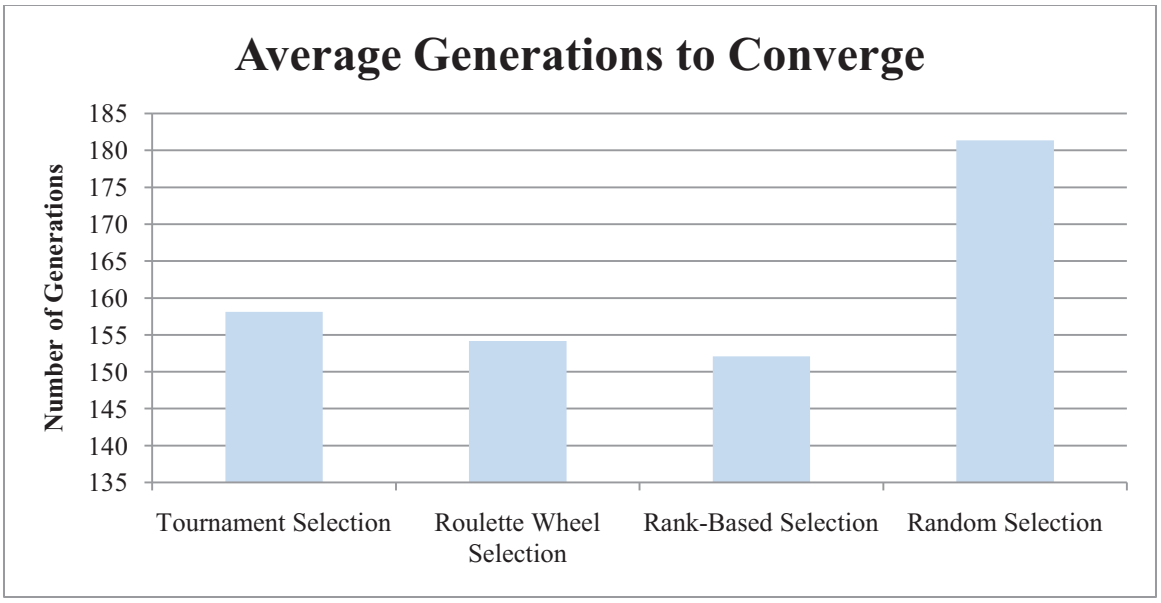

Figure 5.18 – The average generations to converge for populations using the four selection operators.

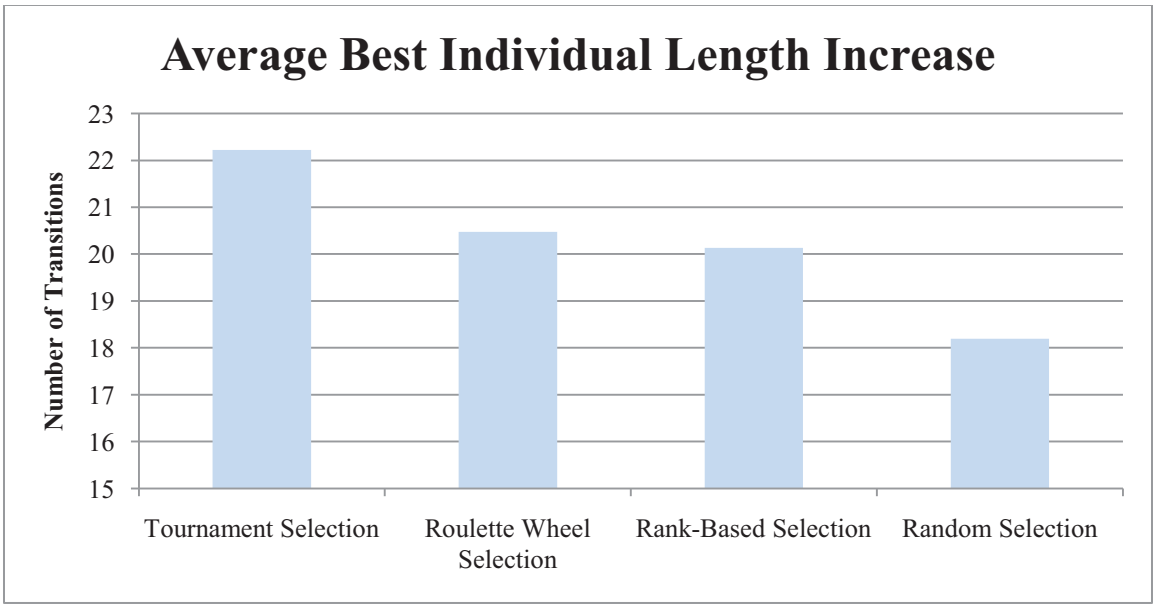

Figure 5.19 – The average increase of the maximum length snake from the first generation until the last in populations using the four selection operators.

5.8 REPRODUCTION OPERATORS

Several reproduction operators were tried with varying levels of success. Uniform crossover was not conducive to the nature of the snake problem. After much experimentation, uniform crossover was abandoned because it failed to produce good results. Despite trying a range a crossover probabilities from 5% to 30%, uniform crossover rarely succeeded in creating good offspring. This was an anticipated result because the relationships between the transitions play an important part in the strength of the offspring. Uniform crossover frequently created offspring with highly fragmented genetic material from the parents. Below, some measurements from early trials using uniform crossover are shown. The results are indicative of the overall trend of performance for uniform crossover when compared to single or double point crossover.

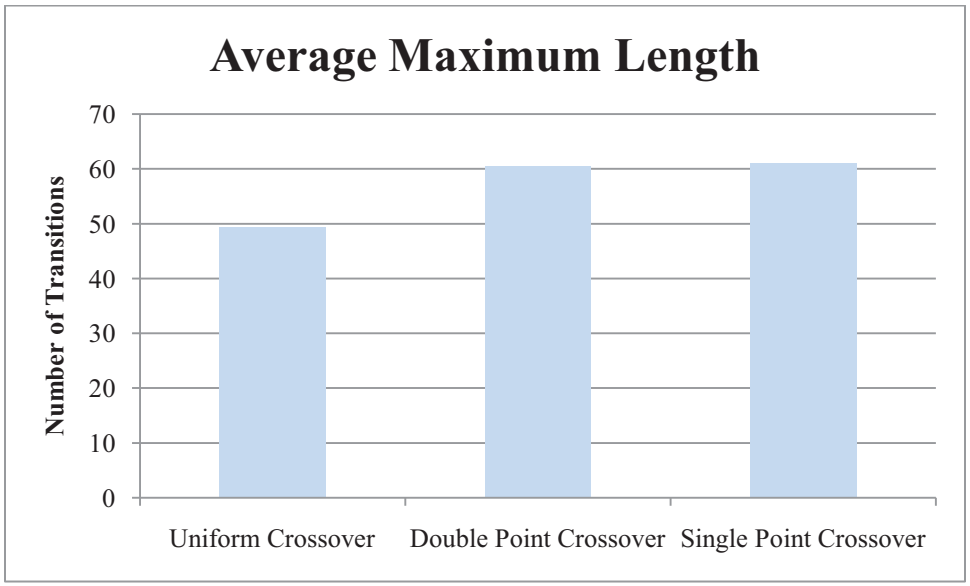

Figure 5.20 – The average maximum length snake found in competitions between populations using uniform, double point, and single point crossover.

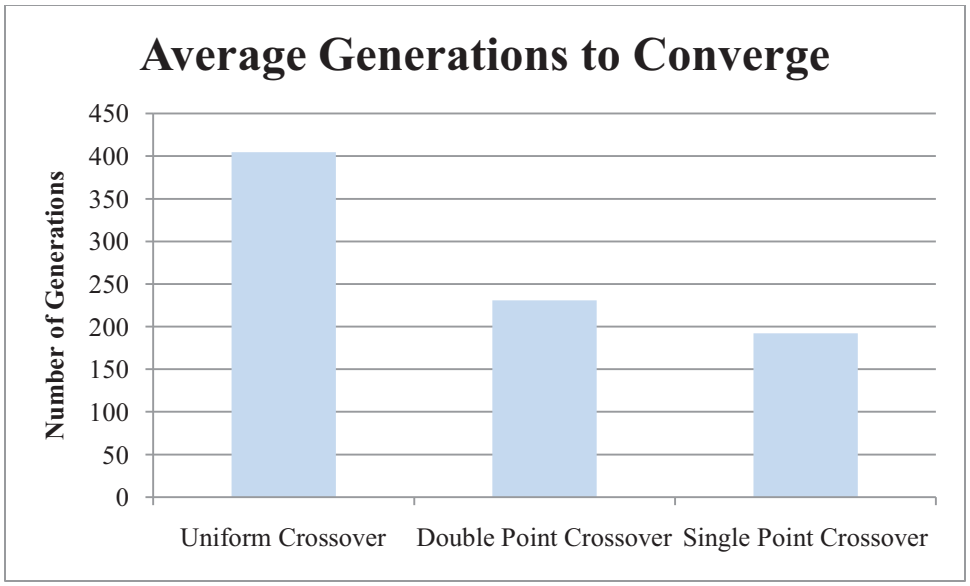

Figure 5.21 – The average generations to converge for populations using uniform, double point, and single point crossover.

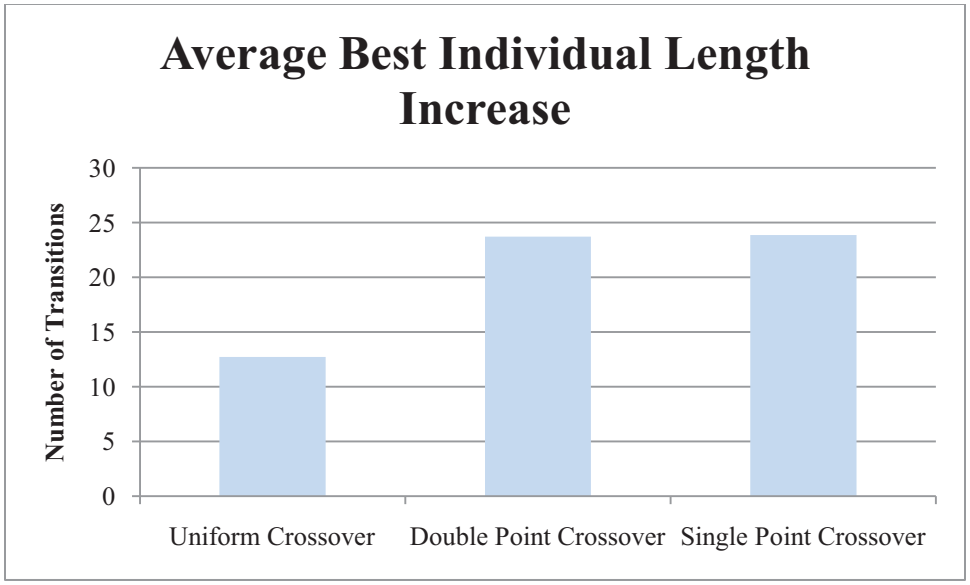

Figure 5.22 – The average increase of the maximum length snake from the first generation until the last in populations using uniform, double point, and single point crossover.

Afterward, focus was placed on single, double, and triple point crossover. Each of these operators has a much higher likelihood of maintaining relative transition relationships intact across generations. Each operator participated in 240 competitions. Single point crossover was

the leader with an average maximum snake length of 58.68. Double point crossover was a close second with an average maximum snake length of 57.55. In last place was triple point crossover with an average maximum snake length of 55.51. The longest snake found in these experiments was length 70 and was found using single point crossover.

Table 5.9 – The parameters used to compare the selection operators.

Experiment Parameters	
Component	Options
Representations	FBTR
Initialization	RRI
Snake blockers	Removed (SBRA)
Fitness Functions	$F_{L\_SSD\_D\_Equal}$, $F_{L\_SSD\_D}$, F_L, F_{L_SSD}, F_{SSD}, F_{SSD_D}, $F_{D\_SSD\_L}$, F_{L_D}, $F_{SSD\_L\_D}$, F_D
Crossover	Single Point, Double Point, Triple Point
Mutation	Random, Xor
Mutation Frequency	0.5%, 1%, 5%
Selection	Random, Rank-Based, Roulette Wheel, Tournament

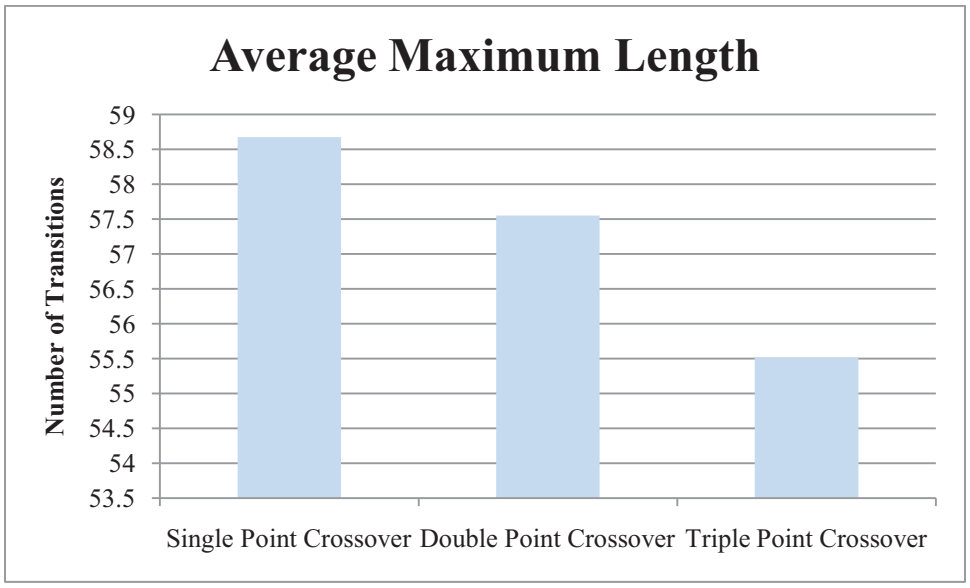

Figure 5.23 – The average maximum length snake found in competitions between the crossover operators.

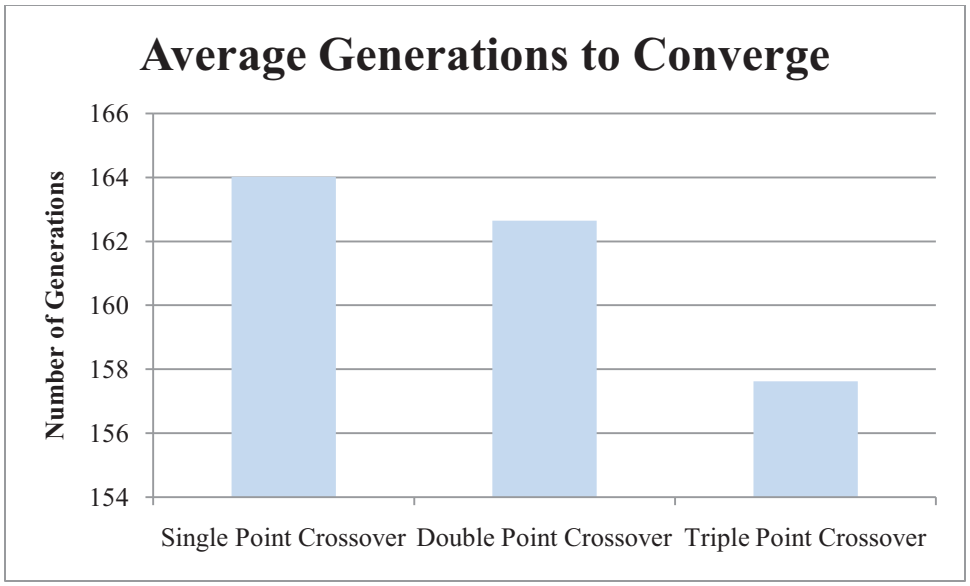

Figure 5.24 – The average generations to converge for populations using the crossover operators.

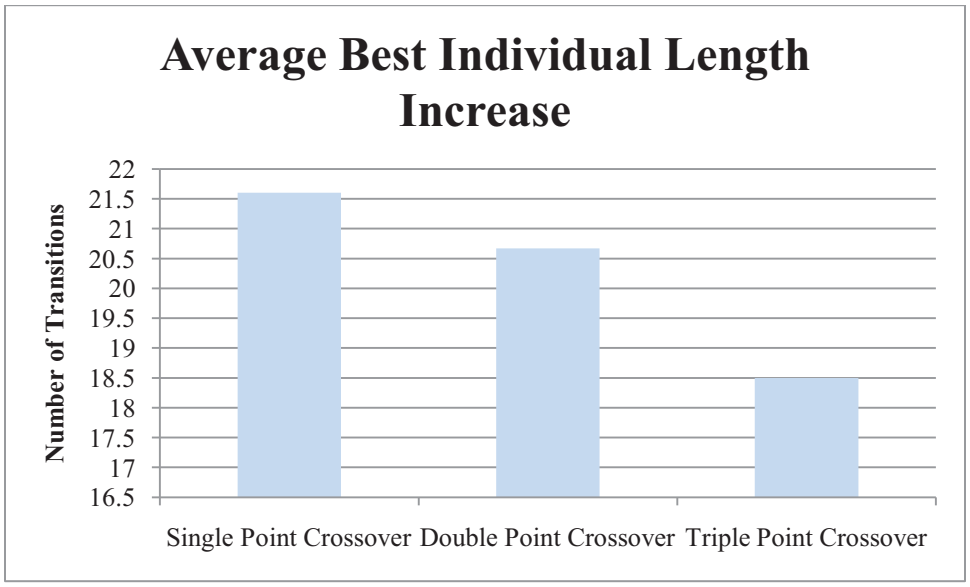

Figure 5.25 – The average increase of the maximum length snake from the first generation until the last in populations using the crossover operators.

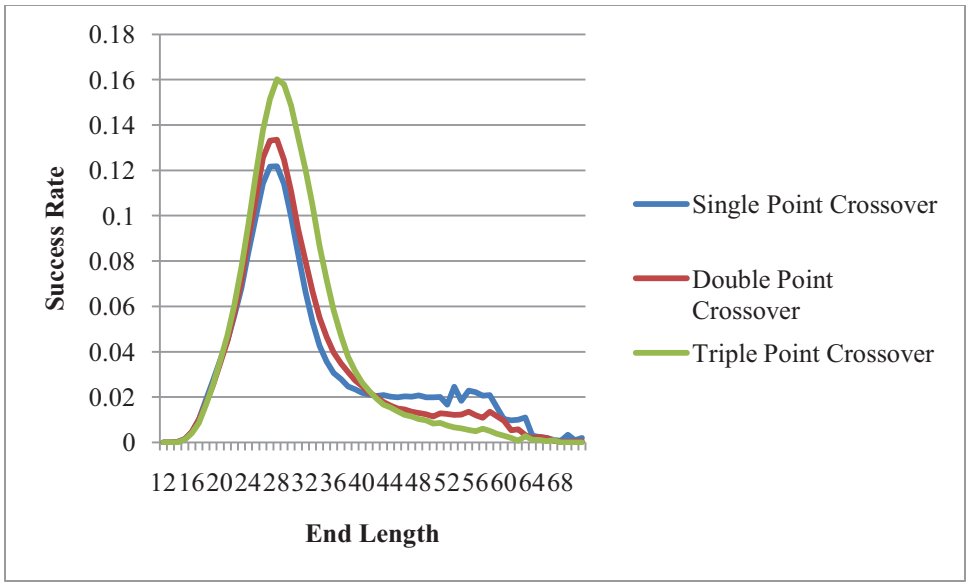

Figure 5.26 – The relative performance of single, double, and triple point crossover on individuals by length. While triple point crossover is more effect for shorter snakes, single point crossover becomes more effective on longer snakes, giving single point crossover an advantage over the course of the GA run.

5.9 MUTATION OPERATORS

Random mutation and Xor mutation were the only two operators used in this project. Each operator was used in 360 competitions and was tested at frequencies of 0.5%, 1%, and 5% for 120 competitions each. Xor mutation had an average maximum length snake of 59.11 while random mutation had an average maximum length snake of 57.38. The best length snakes for these experiments were all of length 70 and one was discovered by using Xor mutation at a frequency of 1% and two were found using random mutation at frequencies of 0.5% and 1%.

Table 5.10 – The parameters used to compare the selection operators.

Experiment Parameters	
Component	Options
Representations	FBTR
Initialization	RRI
Snake blockers	Removed (SBRA)
Fitness Functions	$F_{L\_SSD\_D\_Equal}$, $F_{L\_SSD\_D}$, F_{L}, F_{L_SSD}, F_{SSD}, F_{SSD_D}, $F_{D\_SSD\_L}$, F_{L_D}, $F_{SSD\_L\_D}$, F_{D}
Crossover	Single Point, Double Point, Triple Point
Mutation	Random, Xor
Mutation Frequency	0.5%, 1%, 5%
Selection	Random, Rank-Based, Roulette Wheel, Tournament

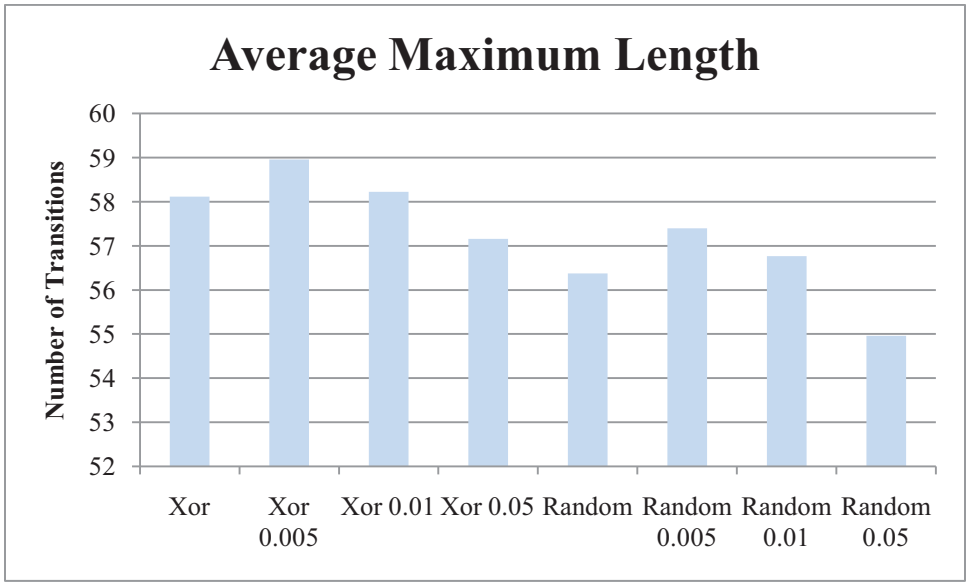

Figure 5.27 – The average maximum length snake found in competitions between the mutation operators, broken down by frequency and averaged across all frequencies.

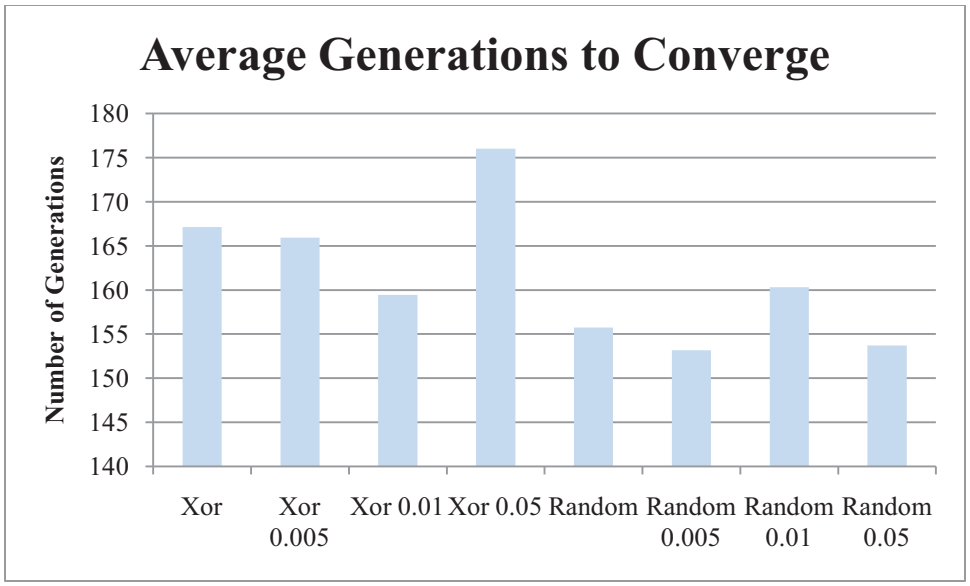

Figure 5.28 – The average generations to converge for populations using the mutation operators, broken down by frequency and averaged across all frequencies.

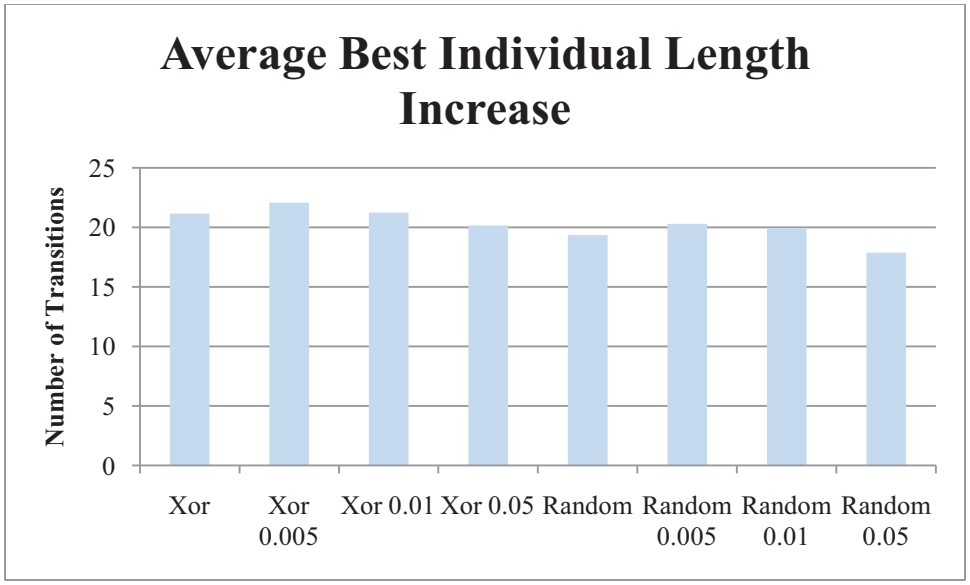

Figure 5.29 – The average increase of the maximum length snake from the first generation until the last in populations using the mutation operators, broken down by frequency and averaged across all frequencies.

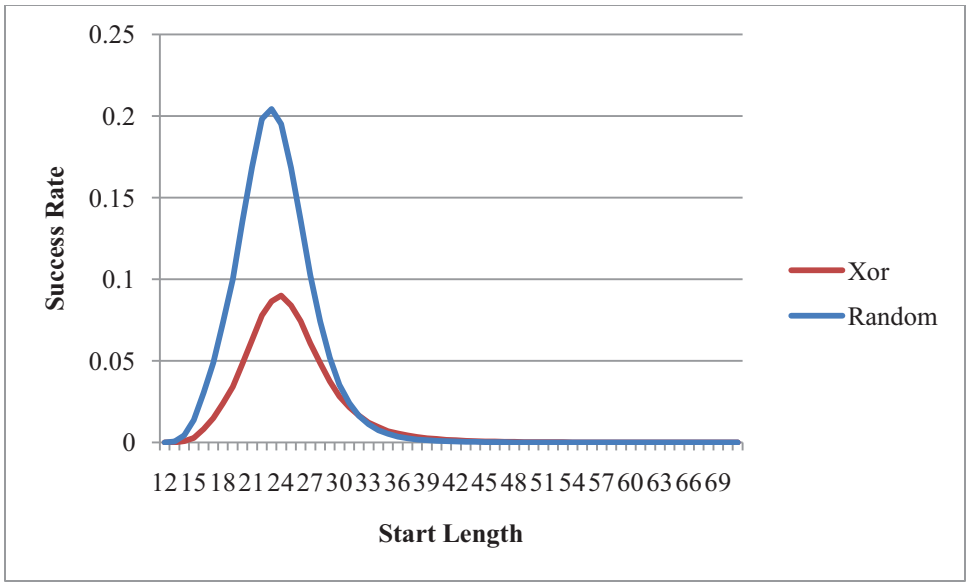

Figure 5.30 – Mutation success rate by start length.

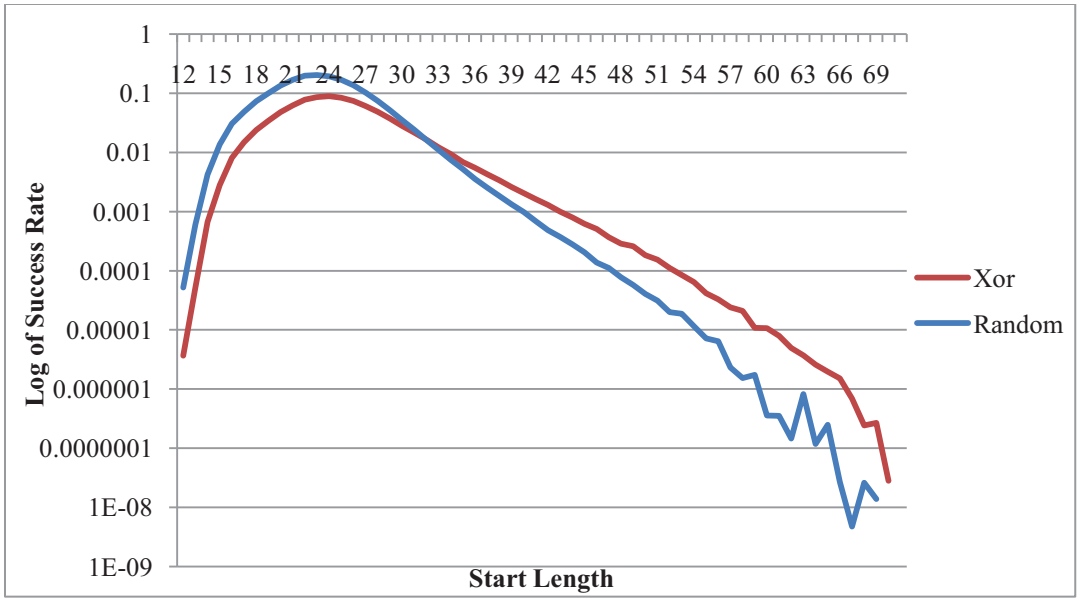

Figure 5.31 – Mutation success rate by start length, this is the same graph as Figure 5.30, but is graphed using log base 10 to better show the differences in longer snakes.

CHAPTER 6

CONCLUSIONS AND FUTURE DIRECTIONS

6.1 CONCLUSIONS

The removal of snake blockers has been shown to greatly improve the performance of the GA. Leaving snake blockers in place did not prevent convergence but did have negative impact on the lengths of the snakes and caused the GA to stop making forward progress sooner. Because removing snake blockers has such a positive impact on the GA results, it seems to be a good tactic to remove snake blockers and benefit from the reduction is search space.

In addition, Frequency-Based Transition Reassignment has been proven to be a more effective representation than a transition-based representation and appears to be effective at reducing the overall representation space associated with the problem. When compared to a transitional representation, Frequency-Based Transition Reassignment populations produced longer snakes. However, comparing Frequency-Based Transition Reassignment to a canonical representation is less clear cut. While using a canonical representation barely outperformed Frequency-Based Transition Reassignment, it did not improve the results enough to be truly significant. Given that using Frequency-Based Transition Reassignment seems to be just as effective as using a canonical representation, further experimentation is warranted.

Restricted Random Initialization has been shown to be a more conservative method of individual initialization. Restricted Heuristic Initialization produces snakes that are too long to the GA to be constructive. As a result, it appears that the use of Restricted Heuristic

Initialization might be better served as a method for initializing only part of the initial population. It is possible that the long snakes Restricted Heuristic Initialization produces are comprised of partial solutions that do not mesh well together, whereas the gradual improvement in a GA initialized with Restricted Random Initialization leads to partial solutions that are more compatible.

Furthermore, selective skin density is an effective component for fitness functions and has proven itself as both a contributing factor and as a standalone measurement of an individual's fitness. While target distribution did not prove to be as effective as a standalone measurement of individual fitness, it did contribute to the overall effectiveness of the fitness function when used in conjunction with length and selective skin density.

Additionally, the results from the experiments have led to some general conclusions. Operators that are more effective on longer snakes are better operators than those that are more effective on shorter snakes. This was illustrated by both the crossover operators and the mutation operators. Despite double and triple point crossover outperforming single point crossover on when the average snake size in the population was small, single point crossover was more effective on longer snakes. Likewise, random mutation was more successful when snake sizes were low, but Xor was more effective on longer snakes.

Furthermore, since most of the operators with longer times of convergence resulted in better individuals, these operators are preferable to those that converge more rapidly because they tend to get stuck in local maxima. Moreover, quality of the individuals in the initial population, as determined by the fitness function, is not always a good indicator of how effective the GA will be.

6.2 FUTURE DIRECTIONS

Despite the nearly equal performance of Frequency-Based Transition Reassignment to a canonical representation in dimension 8, Frequency-Based Transition Reassignment may have an advantage in higher dimensions and when the length of the representation is increase beyond 110 transitions. It is possible that Frequency-Based Transition Reassignment would have a higher success rate than a canonical representation when the average length of the snake were less than half of the length of the representation. By increasing the length of the representation, there may be an opportunity for snakes to develop independently in the front and in the back and then merge as the GA converges. This seems more likely using Frequency-Based Transition Reassignment then a canonical representation because the canonical representation should be less effective for the second half of the chromosome. If the representation is only 110 transitions and the average length of the snakes exceeds 55, then there will necessarily be interaction between the snakes across the chromosome. In this situation, it is possible that the representations are on level ground. However, further experimentation in higher dimensions and with longer representations would be needed to confirm or dismiss this suspicion.

While the concept of target distribution was not an extremely effective fitness factor, it would be interesting to continue experimentation. One option would be to adapt the target distribution based on the contents of the best individuals in the current population. The target distribution could mimic the distribution in the best individuals in an effort to move future populations toward those individuals. Alternatively, the target distribution could become a target of another learning mechanism in order to learn optimal distributions. On the other hand, given that GAs have a tendency to diverge too quickly, perhaps the target distribution could be used to

explicitly combat the distributions present in the best individuals in order to avoid being stuck in a local maxima.

Additional experimentation with selective skin density would also be warranted. Selective skin density still has untapped potential. The advantage of selective skin density and skin density in general, is that, unlike other fitness measurements, skin density says something about specific transitions in the individual. What seems to be missing from the snake problem is a set of operators that are still effective when the length of the snake is large. An operator that could perform crossover while preserving the transitions with the highest skin density may have an increased likelihood of being effective on longer snakes. For other types of heuristic search, skin density can be treated as a measurement of how constrained a given transition is.

Finally, despite the fact that Restricted Heuristic Initialization was largely ineffective in the experiments done here because it caused the GA to converge too quickly, it is possible that more experimentation would lead to methods of making this an effective initialization method. If the reason that the GA fails to make progress is, in fact, related to the fact that the partial solutions in the population are incompatible, this could be worked around by creating a method of classifying individuals based on compatible genetic material. If this could be accomplished, it could lead to better results and drastically reduce search time.

REFERENCES

[Bitterman04] Bitterman, Derrick Scott (2004) *New Lower Bounds for the Snake-in-the-Box Problem.* Master of Science Thesis, The University of Georgia. http://getd.galib.uga.edu/public/bitterman_derrick_s_200412_ms/bitterman_derrick_s_200412_ms.pdf

[Casella05] Casella, D. A. and Potter, W. D. (2005) "Using Evolutionary Techniques to Hunt for Snakes and Coils." *2005 IEEE Congress on Evolutionary Computation (CEC2005)* 3:2499–2504.

[Engelbrecht02] Engelbrecht, Andries P. (2002) *Computational Intelligence: An Introduction*. Hoboken: John Wiley & Sons, Ltd.

[Hardas05] Hardas, Shilpa P. (2005) *An Ant Colony Approach to the Snake-in-the-Box Problem.* Master of Science Thesis, The University of Georgia. http://getd.galib.uga.edu/public/hardas_shilpa_p_200508_ms/hardas_shilpa_p_200508_ms.pdf

[Holland75] Holland, J.H. (1975) *Adaptation in Natural and Artificial Systems*. Ann Arbor: The University of Michigan Press.

[Kautz58] Kautz, W. H. (1958) "Unit-Distance Error-Checking Codes." *IRE Trans. Electronic Computers* 7:179-180.

[Klee67] Klee, V. (1967) "A Method for Constructing Circuit Codes." *Journal of the Association for Computing Machinery* 14:520-538.

[Klee70] Klee, V. (1970) “What is the Maximum Length of a d-Dimensional Snake?” *American Mathematics Monthly* 77:63-65.

[Kochut96] Kochut, K.J. (1996) “Snake-in-the-Box Codes for Dimension 7.” *Journal of Combinatorial Mathematics and Combinatorial Computing* 20:175-185.

[Potter94] Potter, W.D.; Robinson, R.W.; Miller, J.A., Kochut, K.J.; and Redys, D.Z. (1994) “Using The Genetic Algorithm to Find Snake-in-the-Box Codes.” *Industrial and Engineering Applications of Artificial Intelligence and Expert Systems - Proceedings of the Seventh International Conference*. Austin, Texas. 421-426.

[Rajan99] Rajan, Dayanand S. and Shende, Anil M. (1999) “Maximal and Reversible Snakes in Hypercubes.” *24th Annual Australasian Conference on Combinatorial Mathematics and Combinatorial Computing*.

[Taylor07] Taylor, Christopher Allen (2007) *A Comprehensive Framework for the Snake-in-the-Box Problem.* Master of Science Thesis, The University of Georgia. http://getd.galib.uga.edu/public/taylor_christopher_a_200712_ms/taylor_christopher_a_200712_ms.pdf

[Tuohy07] Tuohy, Daniel R. (2007) “Searching for Snake-in-the-Box Codes With Evolved Pruning Models.” *Proceedings of the 2007 International Conference on Genetic and Evolutionary Methods, GEM 2007*. Las Vegas, Nevada. 3-9.