Decoding Algorithms using Side-Effect Machines

Joseph Alexander Brown, BSc. (Hons.)

Computer Science

Submitted in partial fulfillment
of the requirements for the degree of

Masters of Science

Faculty of Computer Science, Brock University
St. Catharines, Ontario

To Elizabeth Reading.

Abstract

Bioinformatics applies computers to problems in molecular biology. Previous research has not addressed edit metric decoders. Decoders for quaternary edit metric codes are finding use in bioinformatics problems with applications to DNA. By using side effect machines we hope to be able to provide efficient decoding algorithms for this open problem. Two ideas for decoding algorithms are presented and examined. Both decoders use Side Effect Machines(SEMs) which are generalizations of finite state automata. Single Classifier Machines(SCMs) use a single side effect machine to classify all words within a code. Locking Side Effect Machines(LSEMs) use multiple side effect machines to create a tree structure of subclassification. The goal is to examine these techniques and provide new decoders for existing codes. Presented are ideas for best practices for the creation of these two types of new edit metric decoders.

Acknowledgements

First, I would like to thank Dr. Sheridan Houghten for her support as my supervisor. Her encouragement and sometimes putting me under deadline ensured the completion of this work. We have a great working relationship and she has given me opportunities that I doubt I would have otherwise. Her door and email were always open

I would also like to mention the efforts of the advisory committee. To Dr. Daniel Ashlock, I thank you for your enthusiasm towards learning and being open to new opinions and able to give insights into any topic I was unclear upon. To Dr. Brian Ross, I thank you for your additions and forcing me to provide arguments for the fundamental underpinnings of this work.

In the computer science department, my friends and colleagues made various comments on my work and encouraged me to continue this line of inquiry. The most important of which is my good friend Earl Foxwell who I constantly bounce ideas off until some of them stick. Further, I would like to thank in particular for their various inputs: Alexander Bailey, Steve Bergen, Baoling Bork, Cale Fairchild, Robert Flack, Tanaby Mofrad, Allen Poapst, Ke Qiu, Jon Ross, and Graham Sharp.

In the biology department I would like to thank Melissa Page and Ellen Robb for giving me some extra research materials into the errors that affect DNA. They also took the time to check my biological argument and made corrections. I may have a lab coat in my closet, but I am far from a biologist.

To all of the above, your support was invaluable to the accuracy and logic of my argument. Any error or omission of fact, which I doubt exist with some of the hawks above, is in the end solely mine. As are any typographical errors solely of my own making.

Finally, I would like to thank my family; my parents, Gary and Ruthann Brown; my sister, Holly. No amount of printed text could be sufficient to express my gratitude for your support and understanding. As well to the memory of Jack and Betty Brown who I know would be proud of me.

J.A.B.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Overview

In bioinformatics, the ideas of information theory and biology are combined. Biological principles are described as mathematical models and via this a large tool-set is available. This tool-set is now manipulating the very codes that life is created from, allowing previously unthinkable changes to be made. Computers must now be used in the manipulation of this data due to the shear sizes involved. This tool-set currently has holes. Previous ideas in information theory have not accounted for the needs of this toolkit. Therefore, this thesis will aim to provide for some of these lacking areas. The codes have not been studied, not due to lack of need, but due to the relative age of the discipline for which they are created.

The goal of this research will be to provide new generalized decoders for this tool-set using Side Effect Machines (SEMs). SEMs are powerful, small, and most importantly simple to implement. They are classifiers which have been used for bioinformatics. This thesis aims to extend their use into new ground: decoding.

1.2 Problem Statement

There is a goal in bioinformatics at the moment to allow the creation of Deoxyribonucleic acid (DNA) sequences that can be inserted into an organism to uniquely identify it. These markers require the ability to correct errors so that mutations caused in the DNA will not affect the ability to recover the marker. DNA is analogous to a communication channel. Shannon asserts that the "fundamental problem of communication is that of reproducing at one point, either exactly or approximately, a message selected at another point"[35]. We must ensure this via the use of error correction using codes which take into account the problems with using DNA as a communication channel.

DNA when used as a medium for a message has its own unique problems for error correction codes. An extremely restrictive distance metric, known as the edit metric, is used to allow for the types of common erros. Further, restrictions may be made by the allowances of biology affect the choice of a code used. The edit metric does not have code families with defined and simple decoders, unlike the Hamming metric codes.

The problem with finding efficient decoders for a random non-linear code is still an open problem with no general solution. When viewing the graph representation of codespace, the Hamming metric has been found to have a relatively simple graph structure. The edit metric graph is a superset of the Hamming graph and contains more edges and less geometric regularity, thus finding a high distance between two points is harder [9]. As edit metric graphs are more complex, they do not yield simple decoding methods based on the graph structure. The Hamming metric has been well studied due to its applications to computer science for transmission and storage of data. Finding decoders for edit metric codes also becomes a more difficult task and currently the generalized decoder for bioinformatics problems is a linear search. Other decoders have been created for edit metric codes; none are generalized enough to allow for a code that can handle all possible sets of biological restrictions.

1.3 Organization of the Thesis

The body of the thesis is organized as follows:

Chapter 2 gives an introduction and review of the noisy channel and error correction codes. Shown are some of the various metrics used in decoding, including the difficulty of the edit metric when compared to the Hamming metric. It also looks into the role of the bioinformatics problems which can be solved and how DNA can be transformed into codes. Finally, it gives a list of applications for which error correction codes for DNA are required.

Chapter 3 shows previous work on the creation and decoding of the edit metric. Critically viewed is how these methods are applied for bioinformatics problems. Finally, it shows an approach which could be taken using ideas from literature for a deterministic decoder. Sadly, the time required for its creation is prohibitive.

Chapter 4 discusses the use of evolutionary algorithms in attacking hard optimization problems. Two processes of evolution are described in detail. The first is Genetic Algorithms which sees solutions as breeding organisms subject to the rules of Darwinian evolution. Genetic Algorithms are electrical analogues to biological chromosomes. The second is Evolutionary Programming which make mutations to finite state machines.

Chapter 5 introduces the Side Effect Machine, a generalization of finite state machines. This creates a classification method which is used in both of the approaches for finding decoders for edit metric codes.

Chapter 6 shows the first look at a decoder using side effect machines: the Single Classifier Machine Decoder. It directly makes a probabilistic decoding. Special attention is made in regards to the explanation of runtime when compared to traditional methods. Differing settings are viewed for the evolutionary algorithms which produce them and these provide best practices in regards to the creation and use.

Chapter 7 presents the Locking Side Effect Machine which uses the ideas of a tree structure in order to have subclassifications of codes. The design of the partitions is viewed critically and three methods of creating the inital partitions are compared — random, lexicographic and K-means clustering.

Chapter 8 gives a summary of the methods presented and future areas of work using side effect machines which will be implemented. These future works include creating the code along with its decoder and the use of side effect machines in data mining.

Throughout the thesis, footnotes within the text will add additional information about results which were deemed interesting but were not necessary to the reading of the primary text and would break the flow of the discussion.

Chapter 2

Review of Error Correcting Codes

2.1 Noisy Channel

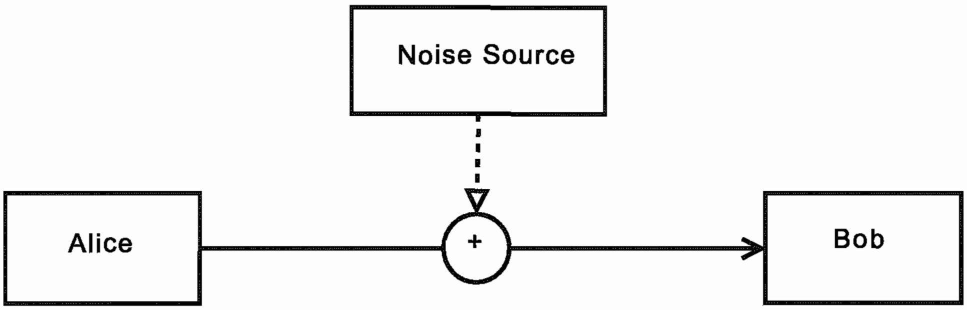


Figure 2.1: The Noisy Channel

Communication is imperfect. The need for error correction stems from the creation of errors through noise. Noise is an all encompassing idea for anything which will degrade the ability to send information along a channel. Examples of channels include records, radio signals, or even DNA. Thus a scratch in a record, a thunder strike causing a hiss in a radio signal, or the incorrect sequencing of DNA create noise and cause errors. The noise can be small or large; compare a slight hiss on a radio signal to not receiving the signal when diving through a tunnel. This degradation will mean at least a loss in the ability to fully understand a message, and in the worst case cause a misunderstanding of a message; perhaps nothing can be recovered from a sent message.

Discrete noise mathematically can be seen as an additive vector to the signal vector. *Error Detection* is the ability to test if this noise vector is non-zero. It does not however discover what the noise vector contains. *Error Correction* is the ability to discover the noise vector, allowing the subtraction of it from the signal vector and the recovery of the original meaning of a message.

Meaning, however, does not refer to the semantic meaning of the message, but to the syntactic meaning of the message. In sending two messages such as "I enjoy a good game of Risk on a Friday night" and "I ate strawberries with cream without the strawberries and without the cream" we show the following. The first sentence has meaning in the normal sense and is a legal English sentence. The second is at most a horrid poem it terms of the meaning. Yet, it is also a legal English sentence. Ergo, both are correct. They have no spelling mistakes and break no grammatical rules. We have rules which govern the use of language, and redundancy introduced which allows us to make correction if a sentence has a mistake. The idea of a spelling mistake should be evidence of this ability to correct and gain meaning from imperfect communication. The English language does not use every combination and permutation of the twenty-six letters. This redundancy allows for correction of mistakes. Further, Cryptanalysis can be viewed as error correction, albeit the errors were caused with the reasoning that errors hide the message. Frequency analysis is used commonly on classical ciphers and uses the idea of the redundancy in the English language in order to 'correct' the error and show the meaning of a message[26]. Moving away from caused noise to accidental noise, we can add helpful redundancies through the use of error correction codes.

2.2 Error Correction and Codes

Before the work of Claude Shannon the only way to remove noise in the signal was to change the channel — making a more powerful signal or making circuits less disrupted by electrical interference [31]. Shannon's Theorem [35] proved that on a noisy channel you can always send a message with an infinitesimally small level of error while still maintaining a decent information rate. The *rate* refers to the amount of information that can be sent in a given

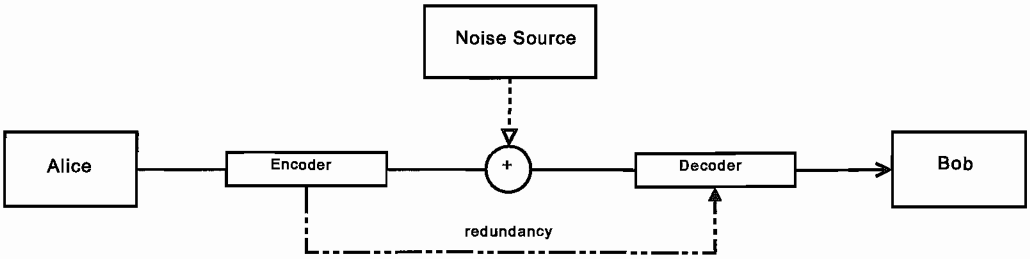


Figure 2.2: The Noisy Channel with a Correction Code

number of sent symbols. The proof is non-constructive. While we know that we can achieve a noiseless channel from a noisy channel, Shannon's Theorem gives no hints as to how this can be done.

Shannon views sending a message as a selection from a list of all possible messages [35]. This selection is the process of transforming a message into a *code* which adds redundancy. He stated that if a message was not part of a list of special messages, the *codewords*, then the sender could not have selected it — it must be an error caused by noise. If the received message is not a codeword than it is called an *error pattern*.

As we are looking at a subset of all possible selections, some of the code is redundant — the code does not carry data up to the maximum possible rate. In general the amount of redundancy of a code is indicative of its correction ability. However, it is in opposition to the rate of information being sent. There is, therefore, a trade off in the number of errors that can be corrected and the rate at which information can be transmitted via a code.

2.2.1 Distance Metric

A distance metric[36] on a set $\mathbb{X}$ is defined by a function $d : \mathbb{X} \times \mathbb{X} \to \mathbb{R}$ where $\forall x, y \in \mathbb{X}$:

1. $d(x, y) \geq 0$
2. $d(x, y) = 0 \Leftrightarrow x = y$
3. $d(x, y) = d(y, x)$
4. $d(x, z) + d(z, y) \geq d(x, y)$

2.2.2 Hamming Metric

Given two equal length strings, $x = \{x_1, x_2, \ldots, x_n\}$ and $y = \{y_1, y_2, \ldots, y_n\}$, the *Hamming* distance between them is defined as the number of locations in which their symbols differ[19]. That is, the Hamming distance is the minimum number of substitutions required to transform x into y or y into x. The algorithm for computing this relation is given in Algorithm 1.

Input: Two Strings, $x = \{x_1, x_2, \ldots, x_n\}$ and $y = \{y_1, y_2, \ldots, y_n\}$
Output: Integer Value of the Hamming Metric Distance
distance $\leftarrow 0$;
for $i \leftarrow 1$ **to** n **do**
| **if** $x_i \neq y_i$ **then** *distance* $\leftarrow$ *distance* $+ 1$;
end
return *distance*

Algorithm 1: Algorithm for Calculating Hamming Distance

2.2.3 Edit Metric

Given two stings, $x = \{x_1, x_2, \ldots, x_n\}$ and $y = \{y_1, y_2, \ldots, y_m\}$, the edit distance is defined as the minimum number of additions, substitutions or deletions required to transform x into y or y into x. It is also known as *Levenshtein* distance[28]. The fastest known algorithm for computing the edit distance is given in [37] and is reproduced as Algorithm 2.

2.2.4 Edit Metric Codes

A $(n, M, d)_q$ code is a set of words for which:

1. n is the number of symbols in a word, also known as the *length* of the code
2. M is the number of words used by the code, also known as the *size* of the code
3. d is the *minimum distance* between codewords using the edit metric
4. q is the number of symbols in the *alphabet*

```
Input: Two Strings, x = {x_1, x_2, ..., x_n} and y = {y_1, y_2, ..., y_m}
Output: Integer Value of the Edit Metric Distance
int d[0, ..., n][0, ..., m];
for i ← 0 to n do
  | d[i][0] ← i;
end
for j ← 0 to m do
  | d[0][j] ← j;
end
for i ← 1 to n do
  for j ← 1 to m do
    if x_i = y_j then
      | cost ← 0;
    else
      cost ← 1;
      d[i][j] = MIN( d[i − 1][j] + 1, d[i][j − 1] + 1,
      d[i − 1][j − 1] + cost );
    end
  end
end
return d[n][m]
```

Algorithm 2: Dynamic Programming Algorithm for Calculating Levenshtein Distance [37]

If the properties of a channel are not known in advance then we rely on a *maximum-likelihood decoding rule* in which an error pattern is corrected to the closest codeword in terms of the chosen distance metric[36][1]. Small numbers of errors are more frequent in general and therefore this assumption makes sense in the general case. A code of this type can correct up to $t = \lfloor (d-1)/2 \rfloor$ errors [24, 36], (Figure 2.3). Some error patterns are equidistant to more than one codeword; in this case the correction is ambiguous. There can also be error patterns that are greater than t distance from every codeword and similarly they cannot be corrected. If a code contains no such error patterns that have these properties which prevent correction, it is known as a *perfect* code[24].

[1]This rule has the minimum error when the channel input probabilities are equal[36].

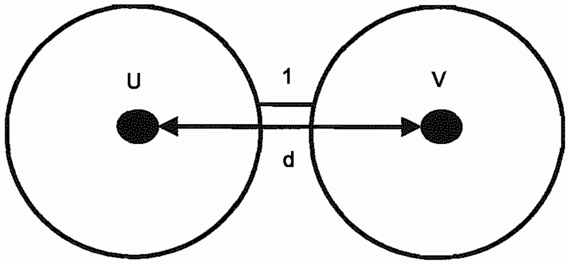


Figure 2.3: View of the Sphere Correction Bounds of Codewords u and v.

The Difficult Metric

Comparing the distance metrics, we see that the edit metric is more strongly connected than the Hamming metric. For example the Hamming distance between 01230123 and 12301230 is 8 since there is a substitution in each symbol. However, the edit distance is only 2 as the deletion of the first symbol and the addition of symbol to the end would have the same effect. Figure 2.4 shows the differences in the graph for binary strings of length three.

The edit metric is seen as a much more difficult metric to create useful codes. This is due in part to its small of automorphism group. An automorphism is a structure preserving transformation. It is an isomorphism from an object to itself. In codes this would mean two codes with differing codewords with the same error correction properties, number of codewords, and distances, that can be changed into each other via a one-to-one and onto mapping of the symbols in each codeword.

Campbell in her PhD thesis[9] proves some of the geometrical reasons as to why creating codes and decoders for the edit metric is a hard problem. In order to create an equivalent code with the same properties but different code words we can use an automorphism of the code. The Hamming metric allows automorphisms by any permutation of the columns of a code and/or any permutation of the symbols of the code. The edit metric's only automorphisms are to reverse all the words simultaneously or to permute the symbols. Thus, edit metric codes have considerably smaller automorphism groups when compared to the Hamming metric, as shown in Table 2.1 for a quaternary code. This relation makes searching for a code harder, as we can make fewer assumptions about its structure.

In general, if the automorphism group is large then there are many equivalent codes. Automorphisms are used to restrict the search space of codes;

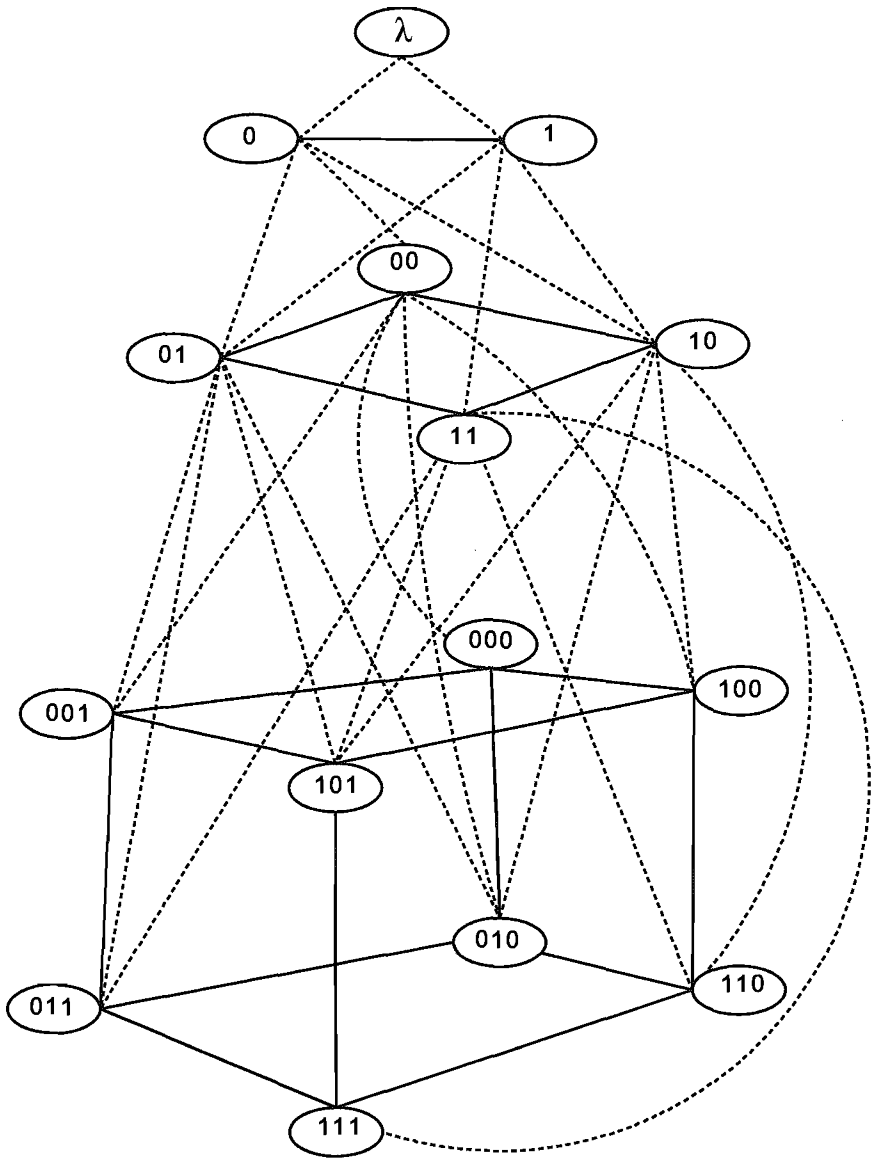


Figure 2.4: Edit metric graph up to distance three. Substitutions are solid lines; additions and deletions are dotted lines. The empty codeword is represented by λ.

given any code, the equivalent codes are easily generated. A deliberate choice of which code to search for reduces the computational time to find a code.

n	Hamming	Edit
1	4	4
2	8	8
3	36	8
4	96	8
5	480	8
$\vdots$	$\vdots$	$\vdots$
n	$4(n!)$	$4(2)$

Table 2.1: Number of automorphisms in a length n quaternary code

2.2.5 Euclidean Metric

The Euclidean metric is the 'ordinary' straight-line distance between two points in an n-dimensional space, $d : \mathbb{R}^n \times \mathbb{R}^n \rightarrow \mathbb{R}$. Given two vectors of a n-dimensional space, $x = \{x_1, x_2, \ldots, x_n\}$ and $y = \{y_1, y_2, \ldots, y_n\}$, the straight line between them is $d(x, y) = \sqrt{(x_1 - y_1)^2 + ... + (x_n - y_n)^2}$ that is $d(x, y) = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2}$.

Input: Two Real Vectors, $x = \{x_1, x_2, \ldots, x_n\}$ and $y = \{y_1, y_2, \ldots, y_n\}$
Output: Real Value of the Euclidean Distance
int distance;
for $i \leftarrow 1$ **to** n **do**
| $distance \leftarrow distance + (x_i - y_i)^2$
end
return $\sqrt{distance}$

Algorithm 3: Algorithm for Euclidean Distance

Note that when checking for relative distance — is point A closer to point B or point C — the square root can be ignored. By ignoring the square root we avoid a computational cost associated with it. Further, if $x, y \in \mathbb{Z}$ then only integer math calculations are required which are faster than floating point calculations. This gives us the additional bonus of being able to compare distances exactly without worrying about rounding errors[2].

2.3 Bioinformatics

2.3.1 Deoxyribonucleic acid (DNA)

The Deoxyribonucleic acid(DNA) is the digital code of life itself. See Figure 2.5 for an image of the structure. DNA's long polymer backbone of nucleotides consists of a phosphate group stripped of one oxygen atom, a sugar known as ribose and one base. It is sufficient to name each nucleotide by the base it contains as it is the only area which differs in a nucleotide. DNA has four base amino acids:

$$purine \begin{cases} adenine & (A) \\ guanine & (G) \end{cases}$$

$$pyrimidine \begin{cases} cytosine & (C) \\ thymine & (T) \end{cases}$$

The strand of DNA connects all the nucleotides in a chain with covalent bonds which are very strong as the atoms share electrons. In many organisms a strand of DNA bonds with a inverse strand — A bonds with T and G bonds to C. These are known as interstrand bonds, created via a hydrogen bond, and are much weaker than the covalent bonds on the backbone. This setup of bonds allows for the strands to separate in order to replicate. Errors can occur in this replication.

2.3.2 Biological Errors

Two common types of errors in DNA[17] are:

1. *Base Pair Substitutions* — occur when one or more base pairs in a gene are changed (**substitution**).
2. *Frameshift Mutations* — occur when one or more base pairs are inserted (**insertion**) or removed (**deletion**) in a gene. It also changes the reading frame.

[2] This 'fast distance' is commonly used in computational geometry.

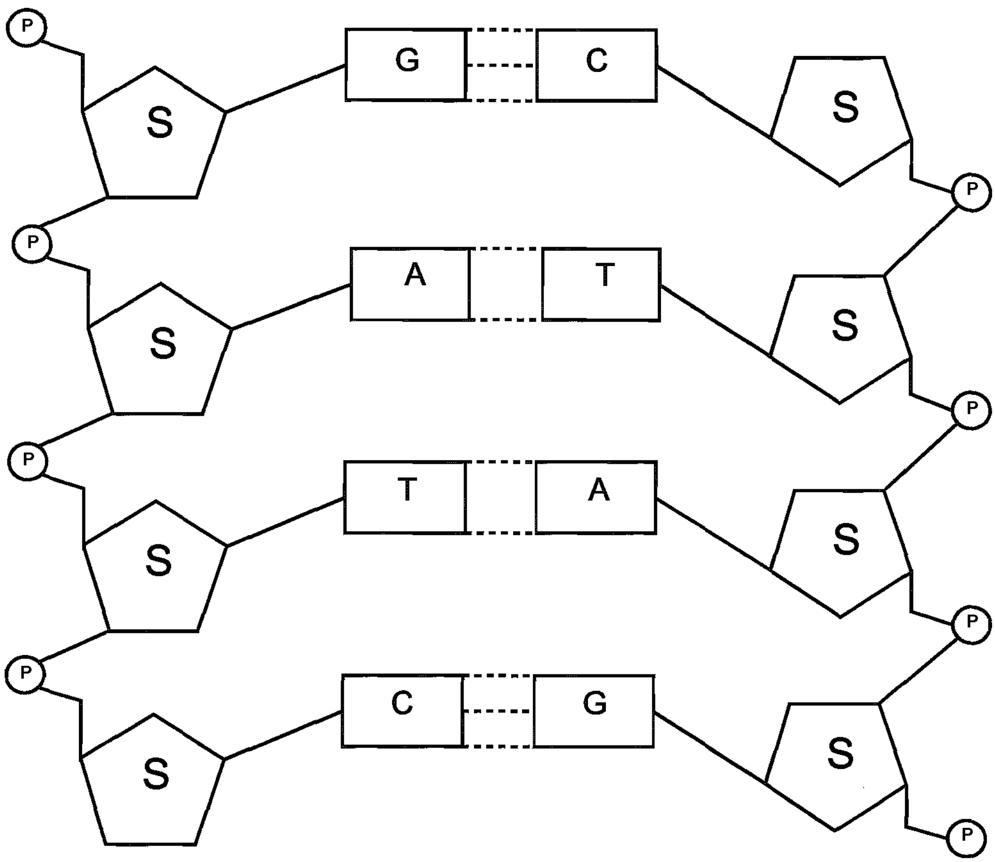


Figure 2.5: Structure of Deoxyribonucleic acid (DNA)

Base pair substitutions have two subtypes. The first is transition errors where one purine/pyrimidine is exchanged with another purine/pyrimidine. The second is transversions where a purine is swapped for a pyrimidine or a pyrimidine is swapped for a purine. See Table 2.2 for the enumeration of these errors.

Frameshift Mutations are insertions and deletions that shift the reading frame. The reading frame is the start of three base points where the encoding proteins begins. Each three base pairs encode one of the twenty amino acids used in proteins or as a 'punctuation' which allows for the mRNA to know where to start and stop transcription. Insertions and deletions in sets of three base points do not shift the reading frame, but still cause changes in the function of the genome as the three base pair code is added or removed.

Error Type	Mutational Error
Transition	$A \rightarrow G, G \rightarrow A$
	$C \rightarrow T, T \rightarrow C$
Transversion	$A \rightarrow C, C \rightarrow A$
	$A \rightarrow T, T \rightarrow A$
	$G \rightarrow C, C \rightarrow G$
	$G \rightarrow T, T \rightarrow G$

Table 2.2: Base Pair Substitutions in DNA[17]

Different mRNA stands can be encoded leading to a malfunctioning protein or no protein being created[3].

Any number insertions and deletions will cause unwanted results when we use DNA as a data transfer media. Therefore, an error correcting code which is based in DNA must correct any number of insertions or deletions regardless of the change in the reading frame.

2.3.3 Sequencing Errors

Sequencing techniques are prone to errors. As a strand of DNA is sequenced, the process can create errors in the forms of substitutions, insertions and deletions.

2.4 From DNA to Codes

As DNA consists of four bases, we can make a bijective mapping to quaternary codes, that is, the values of $0, \ldots, 3$. As the errors may be formed by additions, substitutions, or deletions, the distance metric chosen for the code must take this into account. Therefore, the edit metric is used. The code selected may also have to obey further restrictions laid out by the biological question it must solve.

[3]The tiny virus known as $\phi X174$ uses the idea of reading frames to 'compress' its own genetic code[20]. Some of its nine genes overlap — that is it has two distinct genes encoded in the same stretch of DNA. One gene is even contained completely inside of another. This compression is allowed as the 'second' gene is shifted exactly one base pair relative to the first. An error in this virus would most likely cause huge problems in the coding of multiple functions of the virus. One may argue that the virus has a very high information rate, but it lacks error recovery.

2.5 Biological Applications of Codes

One such application of biological error correction codes is in the 'barcoding' of Expressed Sequence Tags (EST)[3, 9]. Certain genes, significant areas of DNA which encode into proteins, are only expressed with suitable conditions: drought, frost, damage and disease, and period of life cycle to name a few. DNA is translated first into RNA which is then translated into the final protein. The understanding of the location and purpose of a gene is therefore of great microbiological importance. For cost savings the sequencing of strands of DNA can be done in parallel, however this leaves the problem of finding the parameters to which a strand was exposed. Marking each strand with a unique identifier solves this issue effectively. As the strand must be sequenced this inserted marker is prone to error, adding a correction code removes the error and allows a better probability of correct identification.

Second, there is the applications to intellectual property rights of seed producers[9] and protection of consumers to genetically modified organisms. Though the insertion of a sufficent marker into a genetically modified organism, copyright could be enforced. Foodstuffs would be tested to check for the presence of these markers. This in turn would allow for consumer protection for those who enjoy organic foods. The certification agents could monitor such foodstuffs for the presence of markers which signify genetic modification. Those which test positive for markers would not be certified organic.

Third, the study of epidemics could also benefit from marking of bacteria. A case study in the rate of spread of disease could be tested by marking a benign pathogen, infecting a sample, and then testing for the marker in the populations near its introduction. In the case of livestock diseases, if the organism is gentically modified, the marker would allow for better tracing of a particular animal back to the point of origin. Labs also grow cultures of rare and harmful pathogens. Addition of marker code would allow tracing this pathogen back to the lab of origin if discovered in the wild.

Chapter 3

Literature Review

3.1 Edit Metric Code Creation

3.1.1 Conway's Lexicode Algorithm

Conway's lexicode algorithm is a greedy algorithm used to construct edit metric codes. To begin, let C be a (n, d) code with an empty set of codewords. Look at each possible codeword in turn, lexicographically, and add the word if it is at least Hamming Distance d from every word in C. It was originally defined for the Hamming metric, but the edit metric can be subsituted.

Input: An Alphabet Σ, a minimum distance d and an ordered subset $S \subset \Sigma^n$.
Output: $CONWAY(S)$, a subset of S that has pairwise minimum distance d.

set R;
forall $s \in S$ *in order* **do**
  if s *is at least distance* d *from every member in* R **then**
    $R \leftarrow R \cup s$
  end
end
return $CONWAY(S) \leftarrow R$

Algorithm 4: Conway's Lexicode Algorithm

The algorithm defined by Conway[10] considers $S = \Sigma^n$ in lexicographical order.

3.1.2 Evolutionary Additions

Ashlock et al.(2002)[2] used Conway's lexicode algorithm as the basis for a genetic algorithm's fitness function. Called the greedy closure evolutionary algorithm, it seeds Conway's lexicode algorithm with a small code which satisfies the chosen minimum distance. The binary genetic operator compares the seeds and keeps any common words in the children. The remainder of the words in the seeds are then distributed randomly into the two children. Any child seeds violating the distance rules are given a fitness of zero. A seed not in violation is scored by the size of the code produced by Conway's lexicode algorithm using the seed as a starting point.

Houghten et al.[25] provided a faster method for finding codes using Conway's lexicode algorithm. In this variation the codes themselves are stored as the chromosomes. In the binary operator a single new code is produced by shuffling the two parent codes together and adding one new random codeword to the end. This resultant code then undergoes Conway's lexicode algorithm to remove words which do not satisfy the distance bounds. The codes found were smaller then those found in [2]. However, the process is much faster. This allows codes of larger n values to be discovered with less difficulty. The resultant code can also be added to by using the result from the GA as a seed for Conway's lexicode algorithm.

Baker et al.[7] provided a heuristic for extending fixed length edit metric codes into variable length codes. This operates by taking the best fixed-length edit code with the same parameters and then adding as many shorter length codewords that fit within the distance restriction.

Ashlock et al.(2009)[4] improved on the work in [2] and [25] by finding that crossover was actually harmful to the process of finding codes. The evolutionary algorithms using crossover would converge extremely quickly and then begin to find good solutions. Once diversity is removed, the crossover becomes ineffective as we have null crossovers — we are more likely to have a crossover which creates children identical to their parents. Therefore, by removing crossover the algorithm does not need this extra step of convergence before mutation becomes the only effective way to make changes to the population. Mutation is also computationally faster and as such there is a speed increase.

3.2 State of the Art Decoders for Edit Metric

3.2.1 Comma-free Codes

Comma free codes were introduced by Crick et al.[12]in 1957. This paper presented the mathematical reasoning for how the amino acids, some twenty, could be coded by four nucleotides. They proposed that the most likely coding was 'non-overlapping', implying the existence of an unambiguous start and end to a codeword.

A code is called comma-free if and only if given two codewords $x = x_1x_2\ldots x_n$ and $y = y_1y_2\ldots y_n$, then the overlaps $x_i\ldots x_ny_1\ldots y_{i-1}, (0 < i \leq n)$ are not codewords.

This allows the decoders to regain synchronization of the decoder, stopping an error from propagating for the remainder of the code. However, comma-free codes do not make correction to insertion or deletion errors in the blocks in which the error originated.

3.2.2 Marker Codes

Marker codes were proposed by Sellers[34] in order to allow for correction in the edit metric. The marker code acts as a *concatenation code* — an inner code identifies the insertion and deletion errors and an outer code corrects the errors. A marker code adds a unique marker sequence to the end of each codeword. This marker sequence acts as a signal to the code to regain synchronization, allowing the outer burst-error-coding code to detect addition/deletion errors between the markers and correct for them. The longer the added marker sequence to the burst-error-code, the more errors that can be corrected. The addition of these marker sequences does add extra redundancy which limits the rate at which the information can be sent.

3.2.3 Watermark Codes

Watermark codes were first described by Davey et al.[14] and further are compared with Marker codes by Ratzer and MacKay[33]. The code is also a concatenated code which relies on an optimal inner code to which it sends the errors. They combine, via a binary add, a random watermark string into an outer optimal error correcting code which is designed to correct substitution errors. This watermark is analogous to a sheet of paper where the watermark is 'under' the data written 'on top' of it. When the sheet of paper is 'bent', similar to deletion of a character, or 'stretched', similar to the addition of a character, the known watermark gives clues as to where these happened. By inferring the location of the additions or deletions, the inner code first removes the additions which leaves the code a symbol short. This is an equivalent error to an incorrect final symbol due to a subsitution to a null symbol. The error pattern after these removals is then passed to the outer error correction code as substitution errors. The outer code then corrects the errors as if there were substitution errors allowing the use of the Hamming metric.

3.2.4 Suitability to General Bioinformatics Purposes

These forms of codes do not solve the problem as presented, mostly due to how a set of codewords itself is edited to allow for correction. Marker codes would add long marker sequences of the same symbols which would directly affect the temperature of bonding for DNA, one of the more common constraints. Further, a longer sequence must be used to have the same amount of corrective ability. Watermark codes add a random vector. It is unknown therefore if a code created with watermarks will allow for biological constraints as the vector chosen will edit symbols. The problem then becomes selecting an inner code and a 'random' watermark which meet the constraints which may not be computable in a reasonable fashion — a marker cannot be uniformly random if we must have restrictions on it, and uniformly random markers are found to have the best properties[14].

3.2.5 Aho-Corasick Decoder

An extension of a DFA, see Section 5.1.1, can be used in order to decode a message. Previously, finite state machines have been used to calculate the edit distance for a regular language. Konstantindis[27] proves that the problem is solvable in polynomial time and bounds are set on the size of the automation which will accept the language. Finite edit metric codes are regular languages as we can create an enumeration of each element.

By viewing a decoding algorithm as a mapping of error patterns into partitions with their codeword, we can divide the set by using a finite state machine to accept or reject strings. This sorts error patterns into two groups at each stage. We repeatedly to split the set in half until we are left with a

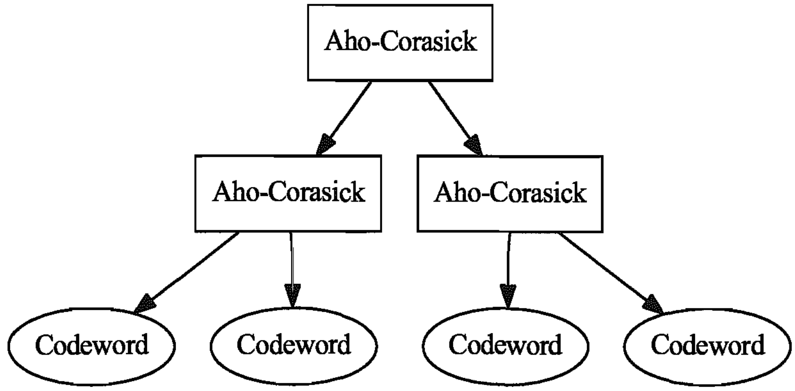


Figure 3.1: Aho-Corasick Used as Part of a Decoder

single partition and its codeword. The members of each partition would be enumerated by an application of the edit metric distance function. The Aho-Corasick algorithm[1] could then take the partition as the keywords to create a finite state machine to decide upon which set to pass the error towards.

Unfortunately, there is the need to enumerate the set and create the partitions of the codewords with their error patterns. This requires computing the edit distance of each error pattern to all codewords which is computationally expensive. For a quaternary code this would be $O(4^N MN^2)$. The proposed Aho-Corasick method would before creation already have every error mapped to the correction which is in effect a look-up table decoder. Using this mapping the problem can be solved in $O(1)$. The Aho-Corasick algorithm being run on this set then only gives a space complexity reduction as we only need to save the final machine that is created and not the entire mapping. The runtime complexity would increase to $O(Nlog_2 M)$ for the savings of only storing the machine.

The creation time for a single code becomes even more unbearable when we consider the number of biological restriction codes that may need to be created. Each one would require its own decoder. The ideas of using a finite state machine, or another similar machine, for the classification does show promise. However, the generation time of the decoder must be taken into account.

Chapter 4

Review of Evolutionary Algorithms

4.1 Genetic Algorithms

Genetic Algorithms (GAs) are a form of evolutionary algorithm and meta-heuristic, see [18, 21]. They use the principles of Darwinian Evolution, especially natural selection. The principles are also known as the survival of the fittest. They provide approximate solutions for optimization problems.

4.1.1 Biological Backing

The idea presented by a GA stems from the biological idea of a single species in a given isolated environment. In GAs this population has the goal of searching a problem instance and finding a good approximate solution; the solution is an organism in the population. The organisms are placed under pressure due to the environment (shelter, food, water) and therefore have a nominal ability to survive, and to breed. There is a fitness score on a problem instance which is used in selection for breeding. The organisms may breed or continue to live, using *inheritance* (Section 4.1.7), *crossover* (Section 4.1.7), and *elitism*(Section 4.1.6). During the breeding slight changes may appear in the organism that are from neither parent and are caused by a mistake in the genetic material; this is *mutation* (Section 4.1.7). As only the fit survive, the organism will hopefully become a specialist at survival in that environment and give a good approximation of an optimal solution.

4.1.2 Solution Representation as a Chromosome

The representation of the solution, known as a chromosome, is a data type that encodes all information necessary to represent one solution to the problem. The chromosome is not necessarily a direct mapping to the solution in that there may be a transcription step. This is similar to the biological theory of there being a genotype and phenotype.

Darwin remarks "Isolation [...] is an important element in the process of natural selection. In a confined or isolated area, if not very large, the organic and inorganic conditions of life will generally be in a great degree uniform; so that natural selection will tend to modify all of the individuals of a varying species throughout the area in the same manner in relation to the same conditions" [13]. Therefore by using this paradigm, upon a single problem the GA should in general see even different chromosomal representations, or species, converge to a solution which is fit.

4.1.3 Initialization

The population is normally initialized randomly. This is to ensure the entire search space is examined. The GA may also be initialized by a selection of good known solutions. This process is know as seeding.

4.1.4 Fitness Function and Selection

The fitness function is a mapping from a chromosome to a value that represents how well the candidate solution solves the problem. These rankings are then used to determine the breeding partners. This process is called selection. The fitness function is problem specific and this function can be the deciding factor on the direction of the genetic algorithm's search paths.

4.1.5 Generations

The algorithm is allowed to run for a number of generations. In each generation the population undergoes an update. The fitness function is calculated for each member of the population and selection of breeding candidates is made. Genetic Operators are applied to the population and the result becomes the population in the next generation. Usually, the GA runs for a number of generations to ensure that it converges, where convergence is the point at which the GA cannot make large improvements in the approximation. The point of convergence is normally decided by empirical testing. The stopping parameter may be defined by other factors coming from the results of the GA itself, such as the difference in the average solution fitness or reaching a defined value of fitness, with a upper bound defined to stop the GA in the worst case. In order to compare various sizes of population the number of breeding events is normally used as a stopping condition. This ensures that each population size has the same opportunity to make changes via applications of Genetic Operators.

4.1.6 Elitism

In order to ensure that we hold onto the best solution thus far, a small portion of the population, normally one chromosome, is selected to be elite. The fittest chromosome in a generation is copied to the next without modification. This chromosome is allowed to then also be selected to be a breeding parent in the normal course of selection.

4.1.7 Genetic Operators

The following types of operators are applied to the population probabilistically.

Crossover

Crossover creates new candidate solutions by combining the genetic material of two chromosomes together. Each child inherits some material from both parents, which hopefully causes the formation of a better solution that shares properties from both.

Mutation

Mutation promotes diversity in the population and prevents evolutionary stagnation. By making a small change to a single chromosome, the area searched by the GA expands. It is also used to prevent premature loss of helpful genetic data.

Inheritance

The chromosome is copied into the new population unchanged.

4.2 Evolutionary Programming

Evolutionary Programming(EP) is a form of Evolutionary Algorithm created by Lawrence J. Fogel to model prediction problems [16]. Problems which predict the next symbol likely to occur given a sequence of symbols observed thus far are modeled through his technique. The EP model relies on the manipulation of a finite state machine which outputs the next predicted symbol on each transition. The finite state machine population is changed through mutation alone, where the parent is replaced by a child only if the number of errors produced by the child would be less than itself.

The machine is used online — that is the problem instances are ongoing during the evolutionary process. Therefore, the concept of a generation is the number of mutations that can be applied and tested before the next prediction instance is given on a new better machine. The mutation operators may include: changing the connections between the states, changing a transition output, changing the initial state, adding a state, or removing a state.

A concept of a version of crossover is examined. The idea is to create a new state machine by looking at the majority logic of the machines. The states are combined and the output symbol is decided by the output of the machines. Fogel notes that at least three machines are needed to show a clear majority.

Chapter 5

Side Effect Machines

5.1 Deterministic Finite Automation

A Deterministic Finite Automation(DFA) is a type of automation which has no temporary storage and makes a binary classification of an input string. See [22, 29] for an introduction to their uses. The only memory it contains is the current state in which it resides. The string is read in one symbol at a time. Each symbol causes a state transition in the machine based upon the symbol read. When the string is empty, i.e. no symbols are left to read, the final state may be either an accepting or a denying state. This creates the binary classification of the string: belongs to the set or does not belong to the set.

5.1.1 Formal Definition

A Deterministic Finite Automation, $M = (Q, \Sigma, \delta, q_0, F)$, is comprised of:

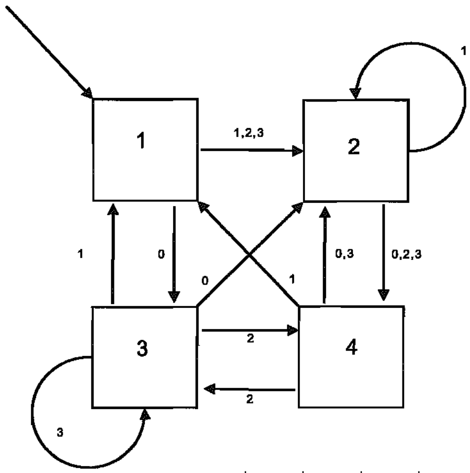
1. A finite set of internal states, Q.

2. A finite set of input symbols, Σ.

3. A transition function defined by $\delta : Q \times \Sigma \rightarrow Q$.

4. An initial state, $q_0 \in Q$.

5. A set of final states which accept the string $F \subseteq Q$.



received pattern	c_1	c_2	c_3	c_4
011231023133	2	4	3	3
132123012313	1	6	1	4
313000213221	2	5	1	4

Figure 5.1: Example four state SEM with examples of output vectors

5.2 Side Effect Machine

The idea of a side effect machine is a generalized extension of the DFAs decribed in Section 5.1. The side effect machine is less interested however in the accepting or denying states but instead in the value of a side effect counter. A counter is attached to each state in the machine. When the state is entered by the machine the counter value updates, normally by incrementing by one. This then provides an injective mapping from the string space of the input into a S-dimensional vector space, where S is the number of states in the machine. The classification therefore comes not from the final state of the machine but from the S-dimensional vector.

5.2.1 Example

Figure 5.1 shows a four state side effect machine. As a convention the SEM always begins on a set state, usually state 1. The classification vector is (c_1, c_2, c_3, c_4) where c_i, $1 \leq i \leq 4$, holds the number of times state i was entered. For example, an input of 011231023133 gives a path through the states of 312422433124. This path yields a classifying vector $c = (2, 4, 3, 3)$, since state 1 is visited 2 times, state 2 is visited 4 times, and states 3 and 4 are each visited 3 times.

5.3 Background

Side effect machine are based on other finite state machines. Recently such a machine was used to classify PCR primers which used an incremental reward fitness function[3]. The machine is allowed three responses when given a primer to classify: good, bad or no idea — *(+,-,?)*. First the final state acted as the only classifier, then the idea was to score the primer based on *+* giving 1 point, - giving -1 point and *?* giving no change in the decision.

Side Effect Machines(SEM) were introduced fully in [6] where they were previously used in bioinformatics applications upon DNA. In [6] side effect machines were used to classify sequences of synthetic DNA to allow for the use of PCR primers. The approach was continued in [5] to look at biological data from *zea mays*(corn). The machines found good classification upon synthetic data but were weaker on the biological data. The reason for this was speculated to be the entropy of the biological DNA compared to the Synthetic approach; the greater the amount of entropy in the strings, the better the machines worked.

The sequences were passed through a genetic algorithm which created candidate side effect machines. The machines then ran the primers and the results of the states were K-means clustered, see Section 7.2. Following that the classifications were evaluated via their Rand index(see Section 7.1).

5.4 GA using SEM

The search space of SEMs is large. In a quaternary machine each state has $4S$ interconnections. These interconnections can be to any of the S states. Therefore, there are S^{4S} ways to arrange the interconnections. This large search space for an optimization problem leads in the direction of using evolutionary computation. GAs are a natural choice as they have previously been used in the creation of SEMs[5, 6]. In order to use a GA the representation and genetic operators must be defined, as in [5, 6].

state	0	1	2	3
1	3	2	2	2
2	4	2	4	4
3	2	1	4	3
4	2	1	3	2

Table 5.1: Example four state SEM transition matrix

5.4.1 Representation

The SEM is represented by a transition matrix. The matrix is of size $S \times |\Sigma|$ where S is the number of states in the machine and $|\Sigma|$ is the number of symbols in the language Σ. For example, the transition matrix in Table 5.1 constructs the machine shown in Figure 5.1.

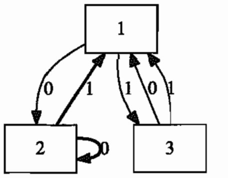
5.4.2 Genetic Operators for SEM

Crossover

The crossover used is two point crossover. In this crossover two points are selected randomly in the first parent. All the edges between those points overwrite the edges in a second parent to create a single child chromosome. This creates a new SEM with states from both parents. An example is shown in Figure 5.2.

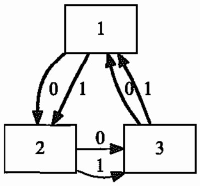
Mutation

Mutation takes one link in the SEM and changes the state it points to randomly. An example of mutation is shown in Figure 5.3. The number of mutations, that is the number of links changed, in a single application of the operator can be varied.



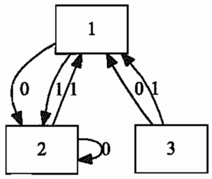
state	0	1
1	2	3
2	**2**	**1**
3	1	1

(a) parent one



state	0	1
1	**2**	**2**
2	3	3
3	**1**	**1**

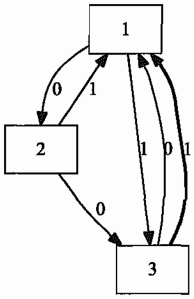
(b) parent two



state	0	1
1	2	2
2	2	1
3	1	1

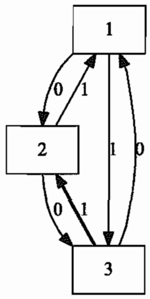
(c) child

Figure 5.2: Crossover of a 3 state binary SEM — crossover point is the second state, the selected edges are bold.



state	0	1
1	2	3
2	3	1
3	1	**1**

(a) before mutation



state	0	1
1	2	3
2	3	1
3	1	**2**

(b) after mutation

Figure 5.3: Mutation in a 3 state binary SEM — mutation occurs in bold edge

Chapter 6

Single Classifier Machine Decoder

The Single Classifier Machine (SCM) Decoder is a SEM which uses the abilities of a SEM to transform the error pattern into a classifying vector. The idea of this decoder is to remove the main loss in efficiency caused by calculating the edit metric for each word. The SCM is a SEM which has been created to minimize the Euclidean distance of the vector results of an error pattern and its respective codeword. Preliminary results from this chapter were published in [8].

The creation of a SCM first involves finding a SEM which will classify the set, normally using a GA (see Section 4.1). After the creation of the SEM the codewords are run though the SEM and the vector results are saved. This forms a mapping from classifying vectors back to the codewords. The SEM, along with the mapping of vectors to codewords, forms the SCM decoder.

To decode a given error pattern, the SCM produces its classifying vector, which is then compared via Euclidean distance to the classifying vector of each codeword. The codeword with the closest vector is the decoded error pattern. Note that the SEM will classify incorrectly at times; we search for a SEM which is correct in the majority of cases. Verification that we have the correct result could be made by calculating the *Levenshtein* or edit distance of the error pattern to the chosen codeword. If this is within the correction capacity of the code, the maximum number of errors a code can correct, then we have made the correct decoding at the additional cost of runtime. A negative verification would have an error response such as 'unable to decode'.

6.1 Fuzzy Classification

The SCM can also be extended to make a fuzzy classification. The SCM already has stored each codeword's classifying vector, and we compare the error pattern's classifying vector when we make a direct classification. In a direct classification we choose the codeword with the closest vector. In a fuzzy classification, the classification vectors of all codewords are inserted into a list sorted in increasing order of Euclidian distance to the classification vector of the codeword. This list of codewords is then compared to the received pattern using *Levenshtein* distance until we find a distance which is less than or equal to the correction capacity of the code—a correct decoding.

A tolerance value can be selected for the range of distance if we want to restrict runtime and return 'unable to decode'. This value is the maximum radius of a hypersphere about the error pattern's classifying vector within which we look for valid codewords. The fuzzy classification will find *Levenshtein* distance for every codeword where the Euclidean distance from the error pattern's classifying vector to the codeword's classifying vector is less than a chosen radius—the tolerance value. If this tolerance is made infinite then we look at every codeword. The fuzzy-SCM creates a list of codewords as input to a linear search. The goal is to ensure a correct decoding earlier in the list.

6.2 Runtime Complexity

Recall that n is the length of a codeword, M is the number of codewords in a code, and S is the number of states in a SEM. Without using a SEM, the general decoding technique used for biological purposes is a linear search taking $O(Mn^2)$, as we must calculate the *Levenshtein* distance of the error pattern to every codeword and select the smallest. The SEM produces the classifying vector for the error pattern in $O(n)$ as the SEM must make a transition for each symbol and add to the classifying vector. The SEM requires $O(S)$ time to find the Euclidean distance from the error pattern's classifying vector to a given codeword's classifying vector, and this must be done for each of the M codewords. The SCM therefore requires a total of $O(n+SM)$ time to decode the error pattern. This becomes $O(n^2+n+SM)$ if we verify the correctness.

The fuzzy machine would require $O(Mn^2 + n + SM)$ time to decode if we were to allow the tolerance value to be infinite, as it may need to make a verification of the correctness for every codeword. However, the upper bound does not show the true runtime accurately. The fuzzy-SCM decoder will probabilistically, based on the properties of the SEM, make the correct classification in $\epsilon n^2 + n + SM$ for some small integer ϵ. By setting the tolerance and accepting some errors we can reduce the worst case runtime. Finding a SEM which classifies effectively is the determining factor on the runtime.

6.3 Experimental Settings

The generational population was varied with settings of 11, 25, 51, and 101 chromosomes. One chromosome of the population is considered elite. The tests were allowed to run for 100000 mating events to ensure convergence. The crossover rate is set to 90% and mutation rate to 10%. Selection was a K-2 Tournament and a chromosome may be subject to both crossover and mutation in the same round of breeding. The number of mutations is allowed to vary with the settings of 1, 2, 7, and 12. Each training set was run with 30 different pseudo random number seed values for statistical significance. The fuzzy SCM was examined with a tolerance of an Euclidean distance of 3. This value was selected by looking at the average Euclidean distance of the errors to the correct codeword for the non-fuzzy SCM.

6.3.1 Distance Two

For each code, two sets of error patterns were generated randomly. Errors of distances 1 and 2 were examined. Error patterns at distance 1 from a codeword were selected to view the effect of a single substitution error, and those at distance 2 were selected to examine a combination of a single insertion and a single deletion, or two substitutions. Two sets of n error patterns using these distances were created for each codeword. The first set was used for the training of the GA on that codeword, and the second was used to verify that the GA was learning the patterns and not just memorizing. Five $(12, M, 7)_4$ codes were tested, available in Appendix A.

Crossover and Mutation Settings	90%/10%
# Mutations	1, 2, 7 and 12
SEM States	6, 12, and 18
Population	11, 25, 51, and 101
Elite	1

6.3.2 Distance Three

The error correcting ability of the SCM was then tested upon $t = 3$ errors which is the full number of errors that a distance $d = 7$ code can correct; remember from Section 2.2.4 that $t = \lfloor (d-1)/2 \rfloor$ and in this case $t = \lfloor (7-1)/2 \rfloor = 3$. Distance three errors are the upper bound on the correction ability for this set of codes. The errors were generated as for distance two tests, adding the distance three errors which are either three substitutions, or a substitution and deletion followed by an insertion.

Crossover and Mutation Settings	90%/10%
# Mutations	1, 2, 7 and 12
SEM States	6, 12, and 18
Population	11, 25, 51, and 101
Elite	1

6.3.3 New Fitness Function

The origional fitness function was the number of corrections made equaled the fitness. The fitness function was then modified to take into account the distance of the corrected error pattern during training. Greater emphasis was placed on the correction of error patterns at higher distances from corresponding codewords. The fitness function equated the score of a single example from the training set to the distance from its corresponding codeword, e.g. a correction of a three error example would add three points to the fitness for the SCM. The training and verification data was the same as the distance three tests.

Crossover and Mutation Settings	90%/10%
# Mutations	1, 2, 7 and 12
SEM States	6, 12, and 18
Population	11, 25, 51, and 101
Elite	1

6.4 Results

6.4.1 Distance Two

The full tables for these results are in Appendix B.1. For each distance there were nM error patterns tested for each distance. A perfect classification would need to correct all these errors.

The greatest difference in the results happens due to the number of states in the side effect machines. The implication being that the representation of the space is only fully explored when a larger SEM space is allowed. The most drastic change happens between 6 and 12 states and is statistically significant. Smaller numbers of states require a higher number of mutations per application of the mutation operator to occur in order to fully be explored. The number of mutations hinders performance for larger populations as the power of selection pressure is removed. The small population effects experienced in the tests for PCR primers[5] are not present in this application of SEMs.

There is a close relation in the numbers of corrected codewords in the training and verification sets. This shows that the SEM is learning the attributes which make up the mapping from error pattern to codeword and not simply memorizing the training sets.

For the first code, decoders with a population of 51, using 2 mutations, gave the best average fitness for both the 12 and 18 state SEM. The 18 state SEM is slightly better. Further, these sets also have a reduced standard deviation compared to other sets. The best SCM of this type was an 18 state SEM created with a population of 25 with 1 mutation. It corrected 81% of all errors from both the training and verification data. The SCM is a highly effective classifier for this information as a random selection of one codeword from the possible fifty-five would lead to only 1.81% of error patterns being corrected.

Fuzzy

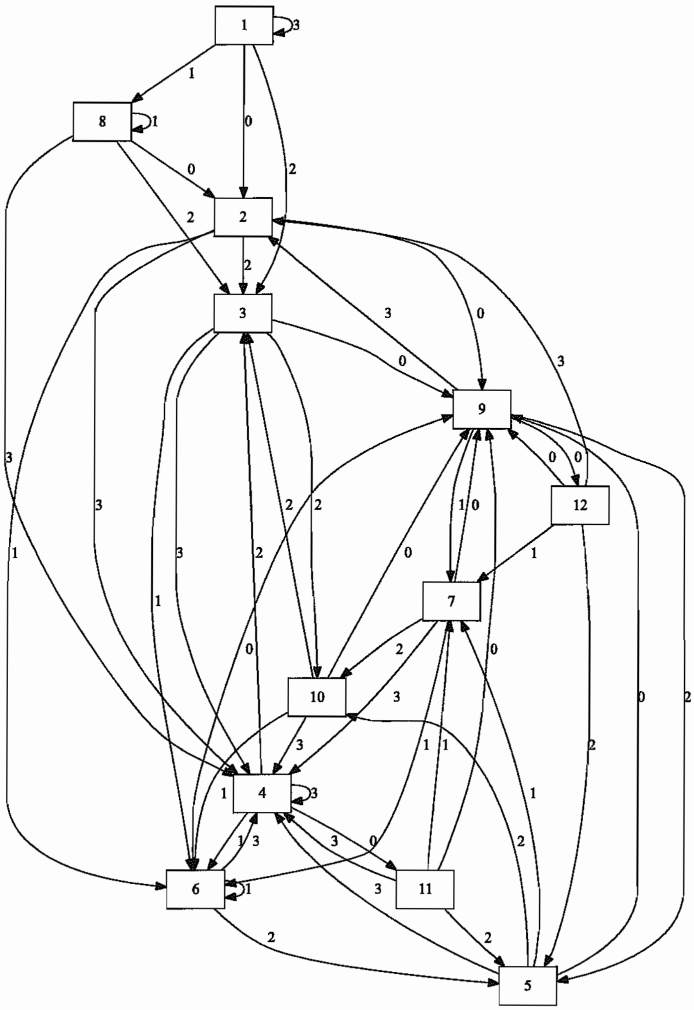
The fuzzy-SCM provided a ten percent increase in the ability of classification at the tolerance level of 3. The standard deviation of the results was greater than the normal SCM, which is to be expected. The smaller populations of machines fared better as the fuzzy classification is able to generalize the errors and place like codewords together. However, the GA lacks the ability to leave local optima at smaller populations due to the selection pressure incurred.

The same settings on the Fuzzy machines gave the best results as the direct SCM. However, the 12 state machine scored slightly better, especially when correcting the harder distance 2 errors. Interestingly, a fuzzy-SCM created with 12 or 18 states fared as well as the best direct SCM decoder, even if their direct decoder was lacking. We must however look at the expense of runtime caused by this gain. The larger the gain in correction ability, the more times we must rely on additional tests of *Levenshtein* distance. Clearly, it is better in terms of runtime to start with a good direct decoder, rather than a weaker one, before adding the fuzzy classification.

For the first code tested, the best SCM of this type was a 12 state SEM created with a population of 25 with 1 mutation. It corrected 93.86% of the all errors from both the training and verification data, see Figure 6.1. The structure of this machine is astounding in how it mirrors the code. Each state has at least one entry into it. However, the initial state 1 is only entered by itself in a loop and state 8 is only entered by itself in a loop and by state 1. These two states act as collectors for runs of the values of 3 and 1 respectively at the start of a word. The SEM is more likely to enter some states based on a single value, e.g. a value of 0 is most likely to send the machine to state 9. After seeing the machine, we noticed there is a large number of runs of a single value in the $(12, 55, 7)_4$ code. Therefore, the GA evolved the SEM to use runs of the same value in order to act as a classification method; we expect the GA to find a method of classification and this classifies the code adequately.

6.4.2 Distance Three

The results in terms of the number of states, number of mutations, and fuzzy machines were close to what was found for the distance two tests as shown in Section 6.4.1. Distance three errors were the hardest to correct. Adding correction to distance three causes a reduction of correction to distance one errors. Table 6.2 shows the results of training a 12 state machine using a population of 51 and allowing up to 2 mutations.



state	0	1	2	3	state	0	1	2	3
1	2	8	3	1	7	9	6	10	4
2	9	6	3	4	8	2	8	3	4
3	9	6	10	4	9	12	7	5	2
4	11	6	3	4	10	9	6	3	4
5	9	7	10	4	11	9	7	5	4
6	9	6	5	4	12	9	7	5	2

Figure 6.1: Best fuzzy machine for the first code — 12 states — corrects 93.86% of errors in training and verification

6.4.3 New Fitness Function

A subset of the tests is shown in Table 6.1 as a good indication of the rest of the results. The parameter settings for this table were: 12 states, population of 51, 90% crossover, and 10% mutation. The change to the fitness function hindered the ability of the SCM to classify the code. There is a statistically significant reduction in the ability to decode distance one errors for every code, excepting the verification of the exact machine in Code #3 and #4. Distance two error correction is hindered to a statistically significant level in the codes. The distance three error range which we aimed to correct was not improved. Code #5 had the worst results for this technique. Therefore, this change to the fitness function is not recommended. By reaching for the errors at a greater distance we lose the ability for the decoder to generalize all errors.

6.5 Number of States

Figures 6.2–6.6 show the average and a 95% confidence interval created during a study of how the number of states affects the memorization of the training set. The runs were conducted on each of the five codes with a population of 51 machines, with 90% crossover rate and 10% mutation rate. They allowed two mutations. The number of states differs between 2 and 30; noting of course that a one state machine has a fitness of zero as all codewords would map to the same vector.

Crossover and Mutation Settings	90%/10%%
# Mutations	2
SEM States	2–30
Population	51
Elite	1

As expected, as the number of states increases at the beginning there is a gain in the effect of memorization and generalization. However, this gain is subject to diminishing returns as good smaller machines are found and the additional states are never used. Notice how the beginning of this long flat region in all five of the testing codes begins when the number of states is 12, which is equal to the length of the codes tested. Some of the extra states are never used and may actually hinder the evolutionary process later on. This

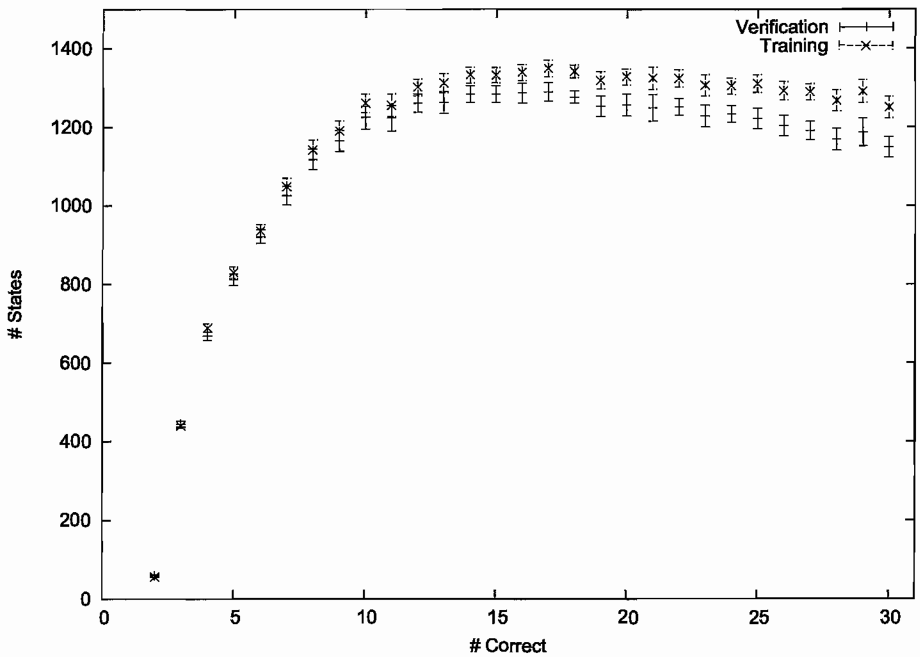


Figure 6.2: Code #1, Correcting up to three errors: Average fitness of best machine over 30 runs with 95% confidence interval with varying numbers of states. Perfect score is 1980.

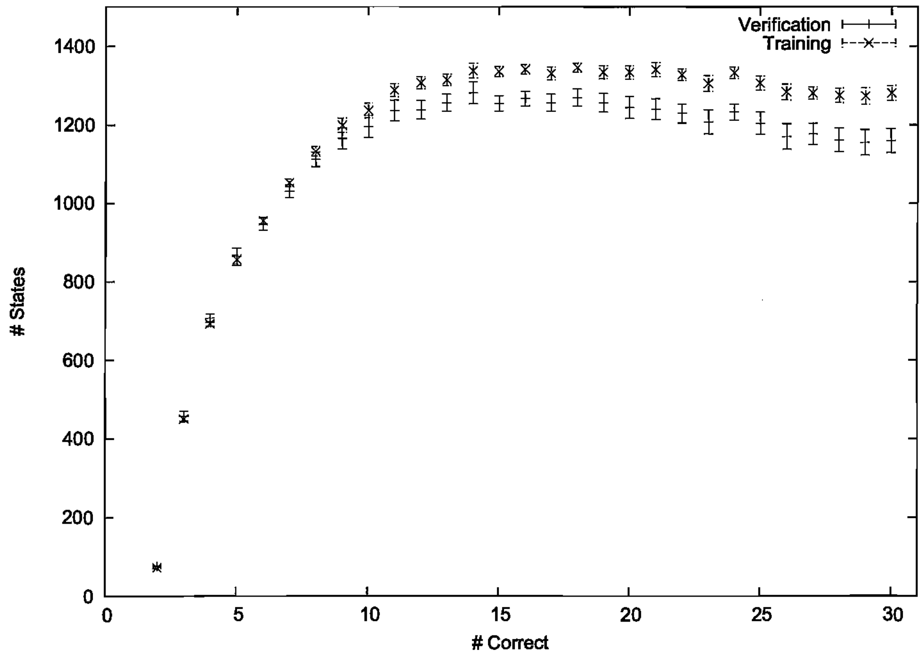


Figure 6.3: Code #2, Correcting up to three errors: Average fitness of best machine over 30 runs with 95% confidence interval with varying numbers of states. Perfect score is 2016.

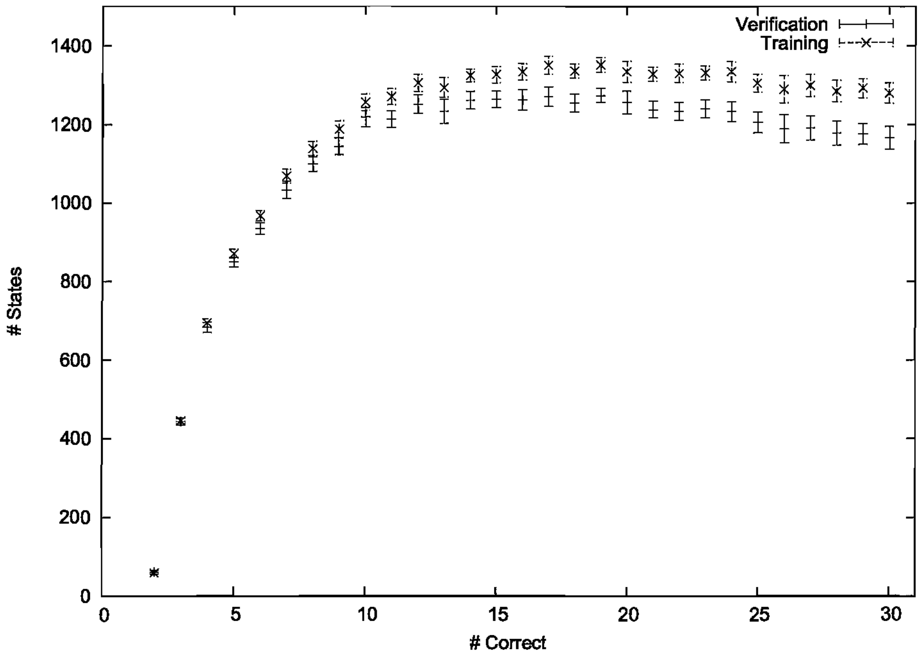


Figure 6.4: Code #3, Correcting up to three errors: Average fitness of best machine over 30 runs with 95% confidence interval with varying numbers of states. Perfect score is 2016.

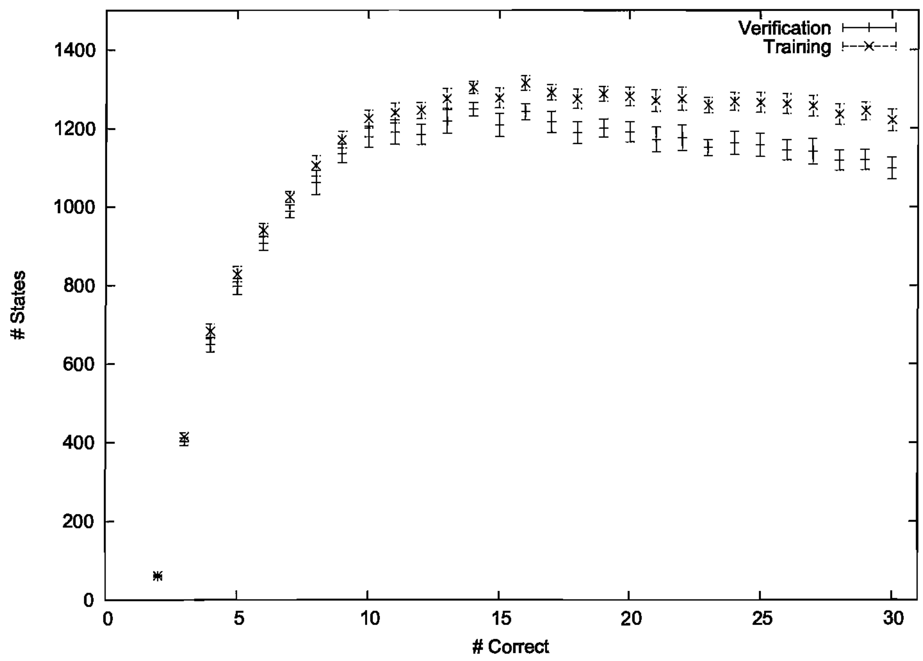


Figure 6.5: Code #4, Correcting up to three errors: Average fitness of best machine over 30 runs with 95% confidence interval with varying numbers of states. Perfect score is 1944.

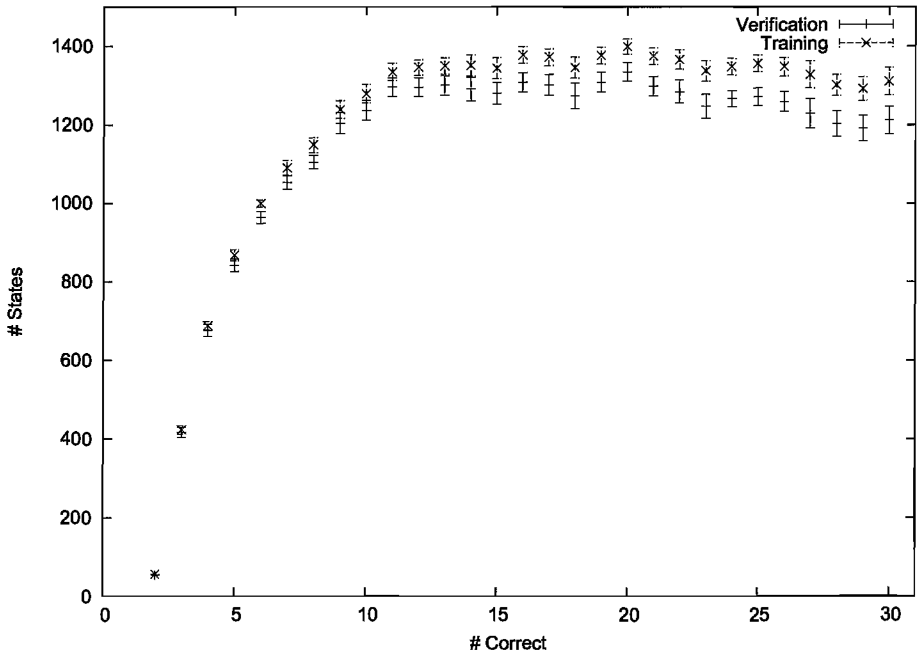


Figure 6.6: Code #5, Correcting up to three errors: Average fitness of best machine over 30 runs with 95% confidence interval with varying numbers of states. Perfect score is 2124.

is evidenced in the verification data which diverges from the training data as the number of sates increases. Seeing as how the number of breeding events is unchanged as the size of the search space is increased with the number of states this also makes the effect of exploitation via GA more difficult.

The efficiency of the decoder rests upon the number of states in the side effect machine, S. Note that the method of evolution used combines machines of a set size, but this size is not necessarily the number of states a machine uses. Through the process of evolution, states can be cut away by having no incoming edges. Further, there can be states which, while having incoming edges, are not reachable by an n length string in n or less moves through the SEM. This means that a setting of S states has as a subset all $1, \ldots, S-1$ state solutions.

6.6 Crossover v. Mutation

There exists a misconception that without crossover a genetic algorithm is just a random search. This is not the case as it does not take into account the idea of selection. There is a fear that removing the crossover rate and increasing the mutation rate makes a work less important and that it could be replaced via a random search. The rates of crossover and mutation should be judged upon the problem instance empirically and then allow for a discovery as to the reasoning behind why a crossover or mutation is successful or unsuccessful. Tests were therefore carried out to look at the rates of crossover and mutation in order to view the relative effect of each genetic operator to finding the solution.

6.6.1 Experimental Settings

Code #1 was selected to undergo further tests at a distance of two in order to establish the effects of changing the crossover and mutation rates. The tests were carried out with the number of breeding events set to 50000 to allow for a faster runtime. Tests were made with the crossover set to the values of 0, 50, 75, 80, 90 and 100 percent. The mutation was set to the values of 10, 20 and 50 percent.

Secondly, distance three tests were carried out with just 50% mutation on all of the five testing codes. The tests were carried out with the number of breeding events set to 50000 to allow for a faster runtime.

Crossover Rate	0, 50, 75, 80, 90 and 100%
Mutation Rate	10, 20 and 50%
# Mutations	1, 2, 7, and 12
SEM States	12
Population	51
Elite	1

6.6.2 Results

A section of these tests with the number of states set to 12 and with a population of 51 is in Appendix B.2. High mutation with low crossover fared as well as high crossover with low mutation. Most noticeable is when the mutation is set to 50 percent; it provides a benefit regardless of the crossover rate.

This benefit is also significant in that average of some of the 50% mutation only runs are close to the best runs found for the previous distance two tests. Previous tests had double the number of breeding events; twice the amount of runtime to find a good solution. Further, the mutation operation is computationally cheaper than crossover.

Tests on distance three codes gave similar results. The results are shown in Table 6.2. While it may be noted that the benefit is only statistically significant in a few occasions, tests are never signficantly worse. Thus, using mutation only does as well or better than having crossover rates at the conventional setting.

6.6.3 Unsuitability of the Problem for Crossover

There are many reasons why this problem shows an unsuitability for crossover and why a mutation only operator is prudent. The first revolves about the idea of breeding partners being of a similar genetic stock. The crossover of states between two machines which are not similar enough will lead to the creation of unused states and disruption of structures. It is an infertile crossover. When the machines are too similar the crossover becomes ineffective. Therefore, after a level of connectivity has been established, crossover has a large chance of breaking the connectivity and creating children which are of extremely low fitness. Secondly, there is a large number of isomorphic side effect machines. Two good machines might have a similar fitness yet still provide infertile crossovers. The application of mutation alone would allow for an exploration of these two isomorphic groups, the best one killing off the other via selection. Third, these flaws could be fixed by changing the representation of the machine. However, storing a SEM as just its transition matrix and making edits via this mechanism is understandable, simple, and elegant.

The idea of Evolutionary Programming as shown in Section 4.2 shows how mutation only on finite state machines can be effective without a change to the representation. As side effect machines share this common representation, one can have a hypothesis that the ideas are transferable.

Parameters				Exact		Fuzzy	
Code	Fitness	Type	Distance	Average	Std Dev	Average	Std Dev
Code #1	Old	Training	1	**557.8**	**17.8681**	**584.1**	**20.6404**
			2	**470.033**	**17.6058**	514.433	23.313
			3	316.633	19.0543	395.367	36.646
		Verification	1	**541.4**	**20.1351**	**573.933**	**22.3374**
			2	452.9	17.529	511.4	26.363
			3	284.033	17.0567	377.5	35.08
	New	Training	1	521.367	25.512	559.567	28.2265
			2	445.867	23.8454	504.5	33.5803
			3	307.6	17.2838	408.833	33.0601
		Verification	1	514.333	28.9486	552.8	30.9576
			2	440.833	28.7559	504.3	37.4379
			3	283.767	18.7445	391	36.9967
Code #2	Old	Training	1	**544.733**	**19.7063**	**580.9**	**20.3272**
			2	460.167	20.3624	**528.867**	**25.2624**
			3	302.567	24.0569	**418.767**	**37.4725**
		Verification	1	**530.6**	**22.9175**	**573.6**	**25.62**
			2	425.6	23.9937	**505.167**	**31.9483**
			3	282.333	23.9559	416.767	35.1624
	New	Training	1	519.033	29.706	558.767	28.1862
			2	450.167	27.5594	513.8	31.3417
			3	303.8	27.3324	397.6	41.7121
		Verification	1	514.433	32.3608	556.367	32.3723
			2	415.733	30.4755	486.167	35.4256
			3	277.967	28.3458	399.4	38.627
Code #3	Old	Training	1	**543.367**	**22.4553**	**585**	**24.9579**
			2	457.567	18.7868	**533.167**	**27.4554**
			3	304.833	23.5139	436.467	42.1006
		Verification	1	531.4	25.4539	**575.533**	**27.797**
			2	439.833	25.5884	**523**	**35.9722**
			3	279.5	22.1682	417.233	43.3595
	New	Training	1	525.633	30.6757	565.367	31.6266
			2	448.067	25.8723	513.267	37.1474
			3	309.433	23.3632	416.5	48.1862
		Verification	1	519.167	31.8651	559.033	34.5228
			2	431.1	27.5297	501.2	39.3826
			3	276.167	24.8569	395.4	53.2668
Code #4	Old	Training	1	**521.367**	**26.4099**	**559.233**	**26.9004**
			2	441.7	18.9412	**514.667**	**29.7024**
			3	284.033	17.6567	407.767	41.3961
		Verification	1	506.233	28.9705	**553.8**	**26.313**
			2	421.767	24.8938	**504.5**	**33.0306**
			3	259.267	23.7195	395.2	46.972
	New	Training	1	503.633	29.808	541.3	28.9353
			2	433.2	23.0268	496.933	25.6111
			3	289.467	19.7759	390.533	33.7514
		Verification	1	494.767	28.2546	536.933	25.5477
			2	417.967	27.1921	488.167	29.3188
			3	258.033	28.629	373.467	40.0472
Code #5	Old	Training	1	**558.467**	**22.2614**	**599.767**	**24.5184**
			2	**474.6**	**17.3555**	**553.5**	**27.2381**
			3	312.533	22.6072	**446.933**	**41.3896**
		Verification	1	**555.6**	**27.0154**	**598.467**	**26.2425**
			2	459	23.5548	**544.067**	**33.8923**
			3	**281.7**	**21.2654**	**432**	**42.0025**
	New	Training	1	530.133	25.8907	573.733	23.5518
			2	456.767	20.2258	523.167	25.6315
			3	311.3	23.0594	410.367	38.2438
		Verification	1	530.867	27.6153	575.4	24.5716
			2	447.1	27.4708	521.033	31.194
			3	266	24.6241	388	37.3215

Table 6.1: Effect of the New Fitness Function — Statistically Significant Results in Bold

	Parameters				Exact		Fuzzy	
Code	Crossover	Mutation	Type	Distance	Average	Std Dev	Average	Std Dev
Code #1	90	10	Training	1	539.333	22.7995	578	22.5511
				2	458.1	22.628	527.867	27.4575
				3	301.2	22.5853	428.433	35.3306
			Verification	1	525.967	25.4551	570.267	25.8096
				2	449.6	28.6002	527.233	27.281
				3	281.9	21.2609	417.567	33.739
	0	50	Training	1	543.6	25.8064	585.8	25.3791
				2	460.167	23.2662	535.8	27.4231
				3	302.8	20.9998	438.033	38.111
			Verification	1	530.3	27.9792	576.2	29.3333
				2	451.5	25.3659	534.567	31.2528
				3	281.167	24.2531	430.933	33.8892
Code #2	90	10	Training	1	544.733	19.7063	580.9	20.3272
				2	460.167	20.3624	528.867	25.2624
				3	302.567	24.0569	418.767	37.4725
			Verification	1	530.6	22.9175	573.6	25.62
				2	425.6	23.9937	505.167	31.9483
				3	282.333	23.9559	416.767	35.1624
	0	50	Training	1	553.067	24.1746	586.8	24.0149
				2	**471.067**	**20.2177**	541.3	27.1752
				3	310.967	20.4863	**438.2**	**35.5551**
			Verification	1	**544.7**	**20.6801**	584.067	23.7558
				2	**442.1**	**24.8726**	**522.433**	**28.0734**
				3	291.567	20.5455	**438.767**	**31.0236**
Code #3	90	10	Training	1	543.367	22.4553	585	24.9579
				2	457.567	18.7868	533.167	27.4554
				3	304.833	23.5139	436.467	42.1006
			Verification	1	531.4	25.4539	575.533	27.797
				2	439.833	25.5884	523	35.9722
				3	279.5	22.1682	417.233	43.3595
	0	50	Training	1	548.367	23.535	585.4	21.9539
				2	462.4	23.261	531.867	27.5026
				3	**316.933**	**16.9806**	439.367	32.8995
			Verification	1	537.7	26.1153	575.533	24.5058
				2	445.9	19.0505	524.733	27.5793
				3	285.967	23.4013	424.667	38.2752
Code #4	90	10	Training	1	521.367	26.4099	559.233	26.9004
				2	441.7	18.9412	514.667	29.7024
				3	284.033	17.6567	407.767	41.3961
			Verification	1	506.233	28.9705	553.8	26.313
				2	421.767	24.8938	504.5	33.0306
				3	259.267	23.7195	395.2	46.972
	0	50	Training	1	528.567	27.4524	563.4	24.5351
				2	449.067	23.5225	518.933	26.8417
				3	291.233	24.0039	417.267	37.4948
			Verification	1	515.533	30.6512	557.3	25.0078
				2	432.733	26.6703	512	30.3497
				3	269.367	28.1198	407.033	41.8219
Code #5	90	10	Training	1	558.467	22.2614	599.767	24.5184
				2	474.6	17.3555	553.5	27.2381
				3	312.533	22.6072	446.933	41.3896
			Verification	1	555.6	27.0154	598.467	26.2425
				2	459	23.5548	544.067	33.8923
				3	281.7	21.2654	432	42.0025
	0	50	Training	1	559.4	27.5238	603.867	29.6773
				2	477.8	19.3505	563.3	31.1616
				3	308.033	23.5042	451.533	40.3867
			Verification	1	557.533	27.295	606.2	29.4611
				2	460.033	22.3568	554.5	32.2808
				3	278.967	22.8526	439.333	41.8868

Table 6.2: Crossover v. Mutation — Statistically Significant Results in Bold

Chapter 7

Locking Side Effect Machine Decoder

The idea of a locking side effect machine comes from a single dial combination lock. In a single dial combination each number must be put in one at a time to move to the next number. Each cam is rotated in a clockwise and then counter clockwise fashion in turn. This process therefore makes a subclassification at each cam to allow the process to continue or to reset the lock. The final number then unlocks the entire lock. Therefore, final classification is made by a set of interlinking machines which have subclassifications. In this analogy each side effect machine is a cam in the lock. The subclassification for each machine is how many times the lock would be turned in the necessary direction. If the correct number of turns is made then it is passed to the next side effect machine in the chain. Otherwise, it returns back to the first level.

Locking Side Effect Machines (LSEM) use the idea of multiple levels to split the codewords into partitions to better classify the code. The error pattern is inserted into the first layer of the LSEM decoder. Each layer first runs the error pattern through a SEM which produces the classifying vector. This classifying vector is then measured via Euclidean distance to the classifying vector of each of that SEM's codeword classifying vectors. K-nearest neighbours (KNN) is then run upon the output (Section 7.3). This gives the classification of the error pattern. There are K more SEM/final codewords under the current layer. It is a tree structure (Figure 7.1). If the classification points to a SEM, then a new layer is entered and the process continues. Otherwise, we have reached a final codeword and that final codeword is re-

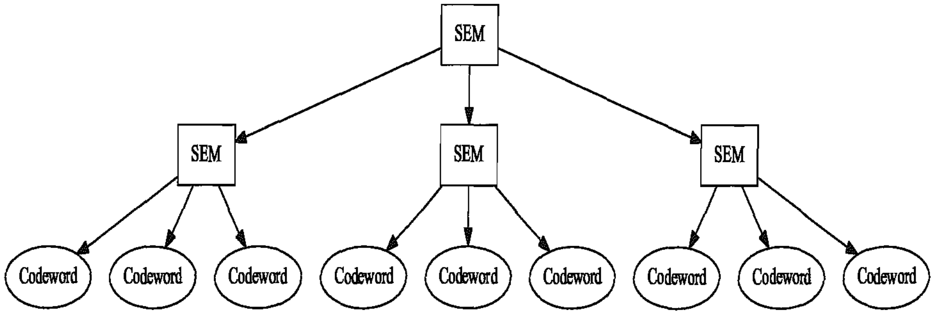


Figure 7.1: Locking Side Effect Machine Decoder Tree Structure

turned as a result. This process does require an exponential number of SEMs to be created. The process of creating further layers can be stopped at any layer, with either to a SCM or a linear search used to complete the decoding.

7.1 Rand Index

The *Rand index*[32] is used to determine how well a clustering of data partitions has been classified into sets. This measure works well as a fitness metric. It tests for the similarity of two partitions of a set even when the number of data points for each set is uneven. If we have a goal partition we compare it to a candidate partition. The index returns a real value in [0,1], where 1 is a perfect classification and 0 is an incorrect classification. Given a set S of n elements and two partitions, X and Y of S, we can define the following:

a, the number of pairs of elements in S that are in the same set in X and in the same set in Y.

b, the number of pairs of elements in S that are in different sets in X and in different sets in Y.

c, the number of pairs of elements in S that are in the same set in X and in different sets in Y.

d, the number of pairs of elements in S that are in different sets in X and in the same set in Y.

The Rand index, given in [32], is:

$$R = \frac{a+b}{a+b+c+d} = \frac{a+b}{\binom{n}{2}}.$$

As we can easily determine the partitioning we require, that is the sets of errors and their codewords, the Rand index will compare the side effect machine's results. The *adjusted Rand index*[23] is used in this study as it corrects for random chance and is calculated in a much sorter runtime.

7.2 K-Means Clustering

K-means Clustering is an unsupervised algorithm which splits a set of data points into K groups. See [30] for the algorithm. The initial set of K means, which are not necessarily data points, are placed into the dimensional space. The data points are then assigned to the closest, in terms of a distance metric, K mean. The center of mass or centroid is calculated for each of the K sets. The K mean is then changed to the location of the centroid and the process of assignment of data and centroid finding continues until no data points are reassigned.

7.3 K-Nearest Neighbours

The goal of K-Nearest Neighbours (KNN) is to assign a previously unclassified data point to the nearest set of previously classified points[11, 15]. The classification is unsupervised. KNN assigns an unknown data point by finding the classification of the K nearest known data points and taking a majority vote. Ties are broken using some reasonable deterministic method.

In this study the codewords' classification vectors are the previously classified data points, and these are the only ones stored by our algorithms. All received patterns' classification vectors become the data points yet to be classified. Therefore, a SEM's goal is to place a received pattern's classification vector close to that of the correct codeword's classification vector.

7.3.1 K-Nearest Neighbours with Homes

One of the problems with KNN for the classification of codewords is that a codeword is not guaranteed to be properly classified. When a codeword comes in as a received vector there could be close codewords from the opposing set. Therefore, we make the addition that if a received pattern lands upon a codeword it is classified to that codeword.

7.4 Runtime Complexity

Recall that n is the length of a codeword, M is the number of codewords in a code, and S is the number of states in a SEM. Let K be the number of partitions we are cutting the code into at each layer. A full decoder using this method is $O((n + SM)log_k M)$.

The layered structure of the classification means that for each layer ($L = 0, \ldots, log_k M - 1$) the Euclidean distance would only be measured to $\frac{M}{k^L}$ points. In the worst case this can be M for the number of codewords for a layer to be tested. Therefore, each layer takes $n+SM$ time to run the side effect machine to find the classification vector and then find the Euclidean distance to each of the classifying vectors of the codewords to run KNN. There is $log_k M$ layers giving a final runtime of $O((n + SM)log_k M)$.

Once again the order hides the true complexity. As there are fewer codewords to consider at each layer, the side effect machine necessary to make a classification in the later levels most likely can be smaller. This leads to a reduction in the S value in later levels.

All of these complexities are assuming that the tree structure created by the decoder is a height-balanced tree[1] in order to ensure that its depth is logarithmic. It is not necessary that the tree be complete as this implies that all nodes are on the left.

7.5 Initial Tests

7.5.1 Experimental Settings

The initial tests were carried out on Code #1 with a population of 51 machines. Two sets of experiments where undertaken using: 1) 90% crossover and 10% mutation and 2) a 50% mutation only setting. Up to two mutations were allowed. Code #1 was split into an equal partitioning which allowed a maximum difference of one word and codewords in each partition were randomly chosen. The K values of 3 and 5 were used in KNN. The SEM was allowed to have 3, 6, 9, 12 and 18 states. Two sets of error patterns were generated randomly. Two sets of n error patterns using upto distance three were created for each codeword. The first set was used for the training of the

[1] AVL trees are examples of height-balanced trees, used to make the look up of a node logarithmic.

GA on that codeword, and the second was used to verify that the GA was learning the patterns and not just memorizing. Five $(12, M, 7)_4$ codes were tested, available in Appendix A.

Crossover and Mutation Settings	90%/10% and 0%/50%
# Mutations	2
SEM States	3, 6, 12, and 18
Population	51
Elite	1
KNN	3 and 5

7.5.2 Results

Figures 7.2–7.5 show the results of the training for the first three layers. They are presented in the tree structure of the final decoder. The tree showing the mean and standard deviation of the training set for each node.

The results were not good for the training set. The top level of classification never broke much higher than 0.6 on the Rand index even when using a 18 state SEM. The belief is that the random partitioning of the initial codewords greatly hinders the ability of the SEM to find patterns. This is a disappointing result considering that the machine only needs to split the code in half, so a random assignment would have a 50% rate of success. Therefore, more advanced methods for the initial partitioning need to be considered. These tests show evidence that the deeper levels have easier classifications which will require fewer states. The right side in the second level of classification for example has one fewer codeword than the right and this effect can be seen in the greater Rand index values.

The mutation only strategy provided better results when the machines were smaller. As the machines grew the difference was less noticeable. The same can be said of the K value for KNN. In smaller machines, looking at five neighbours was more helpful, while larger machines removed this gain since larger machines would allow for larger distances in the Euclidean classification vectors and smaller machines would group their findings closer together. The number of neighbours that are required to be considered would depend on how close the groupings of classification vectors are to each other. A larger representation provided by more states would allow for greater separation between the codeword points and fewer would have to be tested by KNN.

.5823 .0113

.6039 .0238 .6581 .0642

.6470 .0650 .6667 .0684 .6619 .0602 .6663 .0580

(a) 3 State Machines

.6432 .0180

.6651 .0279 .7158 .0651

.7117 .0720 .7255 .0689 .7188 .0611 .7201 .0579

(b) 6 State Machines

.6511 .0183

.6749 .0314 .7286 .0655

.7199 .0719 .7368 .0703 .7351 .0629 .7369 .0596

(c) 9 State Machines

.6427 .0149

.6689 .0333 .7349 .0755

.7197 .0788 .7433 .0811 .7417 .0716 .7432 .0675

(d) 12 State Machines

.6456 .0136

.6810 .0414 .7479 .0821

.7351 .0851 .7603 .0870 .7562 .0783 .7604 .0742

(e) 18 State Machines

Figure 7.2: Crossover 90%, Mutation 10%, and for KNN, $K = 3$

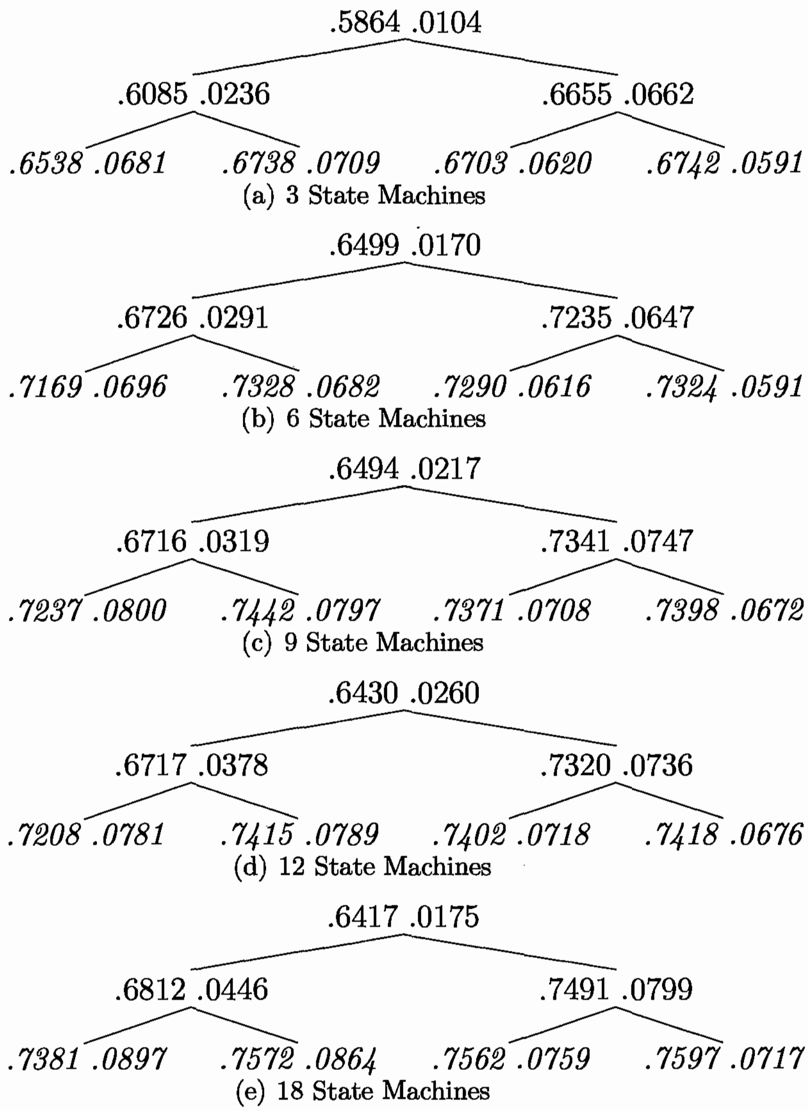


Figure 7.3: No Crossover, Mutation 50%, and for KNN, $K = 3$

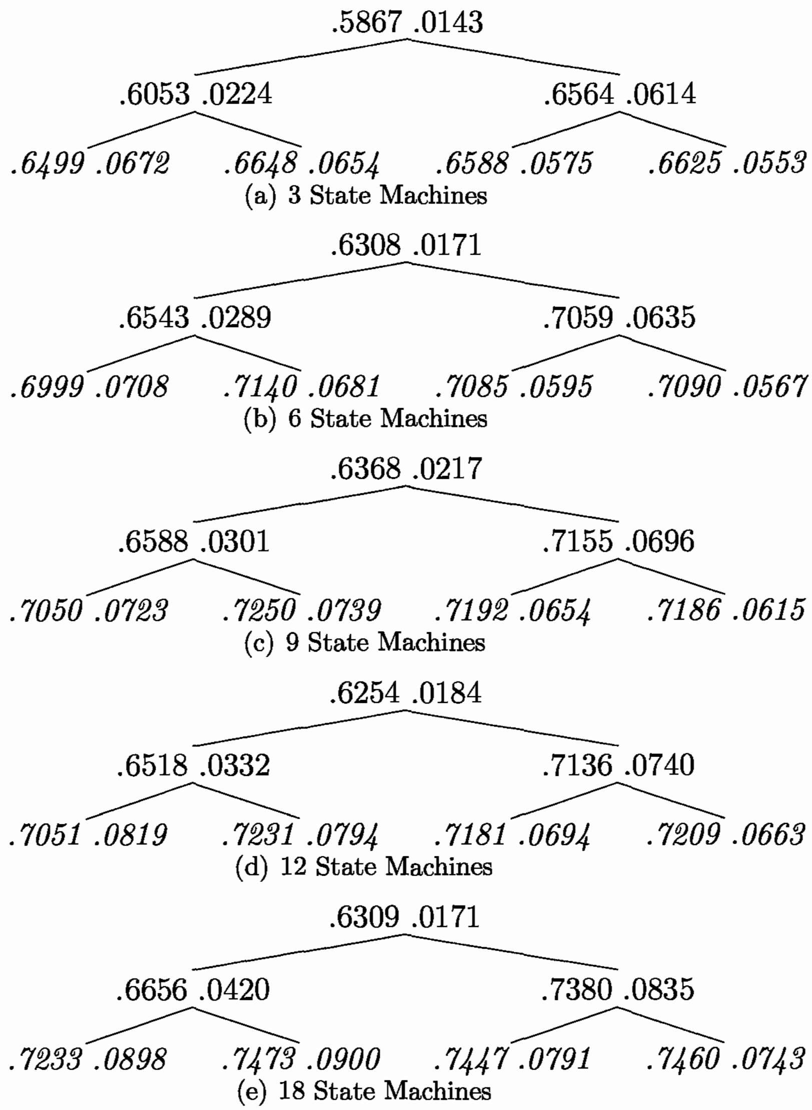


Figure 7.4: Crossover 90%, Mutation 10%, and for KNN, $K = 5$

.5936 .0167

.6123 .0229 .6653 .0653

.6587 .0691 .6759 .0686 .6679 .0611 .6705 .0577

(a) 3 State Machines

.6431 .0138

.6627 .0236 .7185 .0667

.7112 .0733 .7280 .0712 .7224 .0629 .7216 .0594

(b) 6 State Machines

.6438 .0265

.6611 .0314 .7215 .0728

.7108 .0784 .7302 .0782 .7246 .0682 .7250 .0649

(c) 9 State Machines

.6282 .0283

.6571 .0384 .7221 .0755

.7094 .0831 .7294 .0820 .7276 .0729 .7297 .0687

(d) 12 State Machines

.6226 .0181

.6622 .0438 .7347 .0823

.7200 .0903 .7432 .0891 .7425 .0787 .7445 .0740

(e) 18 State Machines

Figure 7.5: No Crossover, Mutation 50%, and for KNN, $K = 5$

7.6 Methods for Finding Partitions

7.6.1 Experimental Settings

The methods for finding partitions were trained on the top level classification, seen as the hardest to make. All of the five testing codes were used. Two settings for crossover and mutation were used: 90%/10% as it is the most common setting and 0%/50% to test the usefulness of the mutation-only strategy. The SEM was allowed to have up to 3, 6, 9, 12 and 18 states. The population was 51 side effect machines. The training and verification data was the same as used for the first set of tests.

Crossover and Mutation Settings	90%/10% and 0%/50%
# Mutations	2
SEM States	3, 6, 12, and 18
Population	51
Elite	1
KNN	3, 5, 7, and 9

The three different methods used are examined below.

7.6.2 Random

The random case acts as the control group for all further studies. The codewords are randomly divided into two groups. The two groups are selected to have a difference in size of not more than one codeword between them.

7.6.3 Lexicographic

SEMs in Chapter 6 have shown that they have these collecting states on the same symbol. Therefore, a partitioning which makes use of this property may lead to better classification. Keeping this in mind, the next method is to divide the code partitions lexicographically. It groups the code in this case into two partitions based on the first symbol, i.e. codewords beginning with 0 and 1 as the first group; 2 and 3 as the second.

There is the concern that the classifications provided by a SEM using this method will have a disproportionate amount of errors on the symbol used for partitioning. The sole reason behind classification could by caused by the symbol used in the partitioning. As an example, this could lead to an error in the first symbol in the first layer to cause a misclassification more frequently.

7.6.4 K-means Clustering

The other idea is to use K-means clustering to generate the initial partitioning of the code. This initial partitioning then acts as a basis upon which the evolution will progress. That is, if the codewords are close in terms of Euclidean distance then we want the classification vectors to be close in terms of the Euclidean distance as well.

The success of the partitioning provided by K means was tested by looking at the intracluster distance over the intercluster distance. That is, the data points within the same cluster should be tightly packed, while the space between clusters should be large.

The intracluster distance was measured by taking the mean distance of each data point to every other within the same cluster, then taking the mean across all clusters. No single large cluster thus dominated the value. The intercluster distance was calculated by taking the mean distance from each K-means centroid to every other.

The lower the value produced by dividing the intracluster distance by the intercluster distance, the tighter the clustering and the farther apart the clusters are from each other. Thirty runs of the K-means algorithm were made and the best result was taken. K-means will sometimes create a partitioning with only one data point within a cluster. The code was written to respond with an error if this is the case. The partitioning should have sets that are equal in size as much as possible in order to allow for the reduction in runtime provided by the division.

The values for the codes are shown below. Thirty runs of K-means were created to allow statistical significance. The lowest valued partition was used in the tests.

Code	Best (Lowest)	Average	Standard Deviation
1	38.44	59.09	11.23
2	40.75	49.78	8.83
3	44.18	59.60	16.10
4	37.02	52.57	7.39
5	43.24	55.14	8.54

7.7 Results

7.7.1 Partitioning Methods

For the partitioning methods the lexicographic sample provides the best classification in the majority of instances. This is attributed to the idea that the classification is heavily weighted on the first symbol. K-means is significantly better than random in all codes excepting Code #2 and is ranged between 0.7 and 0.8 on the rand index. In mutation only tests the K-means method is significantly better than the lexicographical method in Codes #1 and 4.

On Lexicographic Partitioning

The use of lexicographic partitioning, while the best method of classification, has the concern of misclassification based on the first symbol. This could be avoided via a careful selection of training examples which show errors in the first symbol more than average. This would strengthen the resistance of the machine to those types of errors. How this affects the training will be left for future study.

Further, lexicographic partitioning could be used for the first few levels of the LSEM. As the classification level of the random partitioning method increases on smaller classifications for later levels we could use this method. The concerns about the level symbol bias remain in those levels created by Lexicographic LSEM. These concerns could be removed in lower levels by using SCM, random LSEM, or linear search techniques.

7.7.2 Number of Neighbours

The number of neighbours tested has no effect on the correction ability. Figures 7.6–7.25 show the 95% confidence interval with a constant number of 12 states. The change given by the number of neighbours is not statistically significant and is almost constant.

7.7.3 Number of States

The number of states follows a similar pattern to the result for SCM. Figures 7.26–7.45 show the 95% confidence interval with a constant neighbourhood of 3. The inflection point occurs at around 6 states and the gains level off at about this point. More states slightly hinders the K-means partitioning beyond this point.

7.8 Crossover v. Mutation

The use of crossover for this method is much more useful than for a SCM. The problems with crossover upon this representation have been discussed in Section 6.6.3.

In mutation only tests the K-means and random methods have improvement until the 6 or 9 state machines. The lexicographical method degrades as the number of states increases levelling off at 12 states, which is the length of the code. The larger search space hinders the ability of mutation to find a compact SEM. For random and K-means methods the increased search space allows for more differentiation beyond the initial symbol that can guide the lexicographical method. Tests using crossover maintain a regular progression which reaches its peak at 6 states.

In terms of the number of neighbours the rates of crossover and mutation does not make a significant change and using crossover shows an advantage.

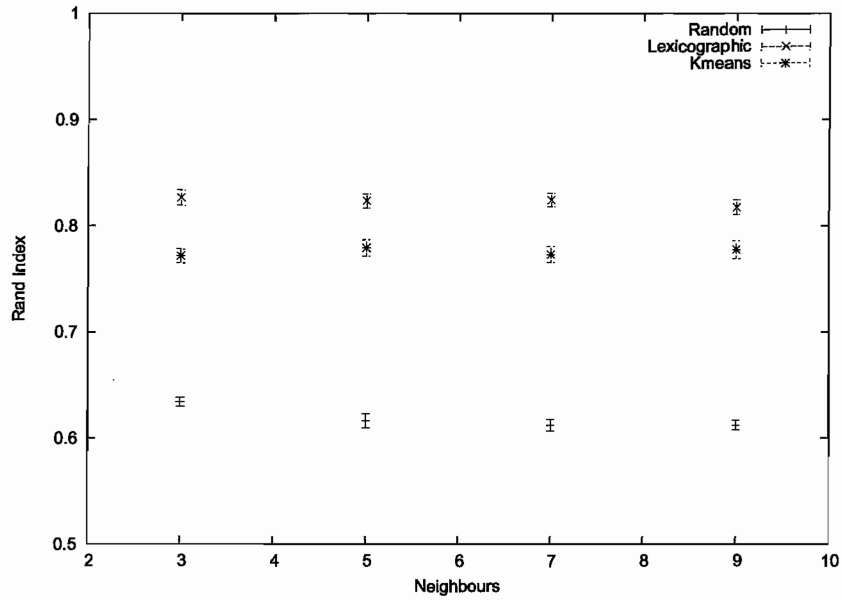


Figure 7.6: Comparison of Neighbours for Code #1, 90%/10% — Training

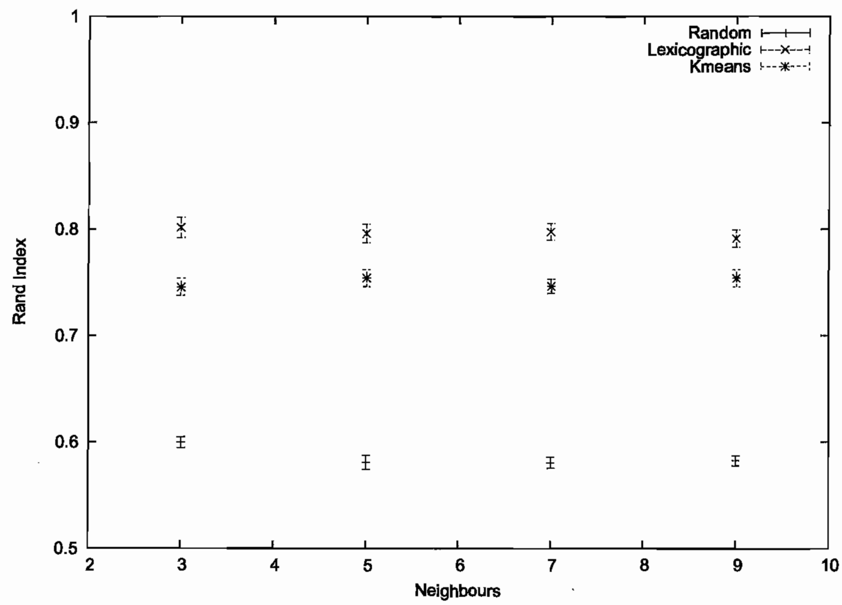


Figure 7.7: Comparison of Neighbours for Code #1, 90%/10% — Verification

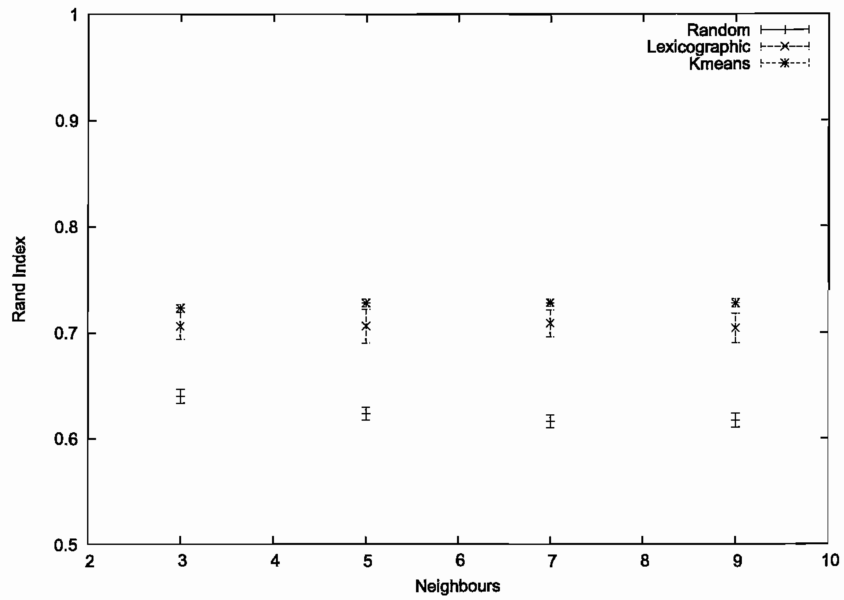


Figure 7.8: Comparison of Neighbours for Code #1, 0%/50% — Training

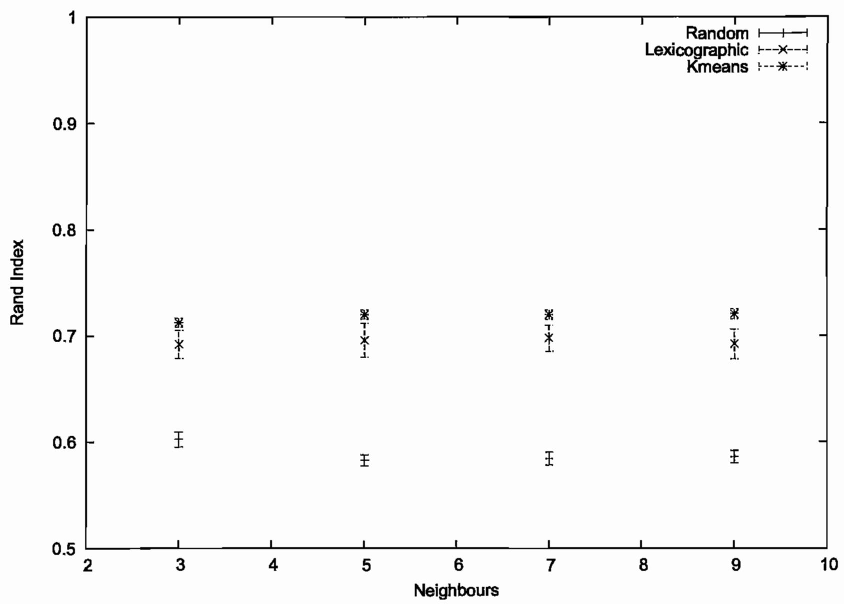


Figure 7.9: Comparison of Neighbours for Code #1, 0%/50% — Verification

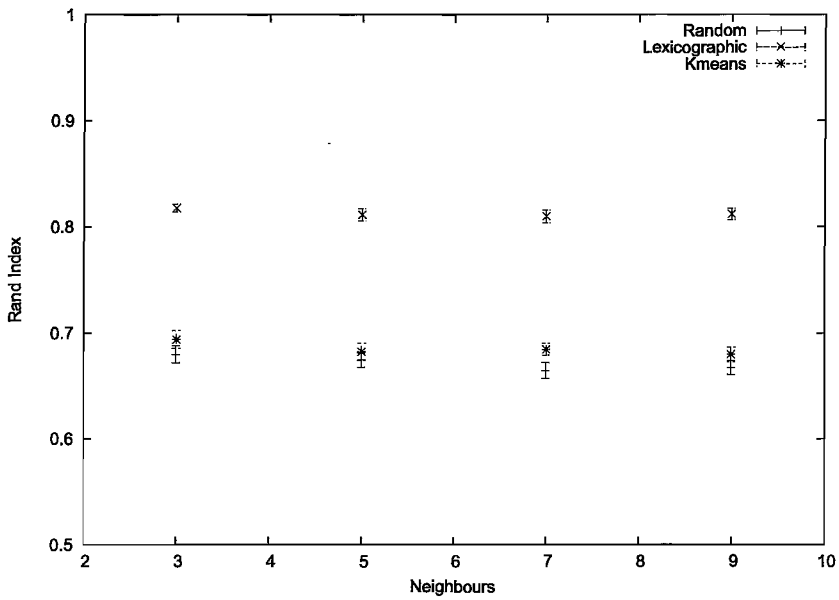


Figure 7.10: Comparison of Neighbours for Code #2, 90%/10% — Training

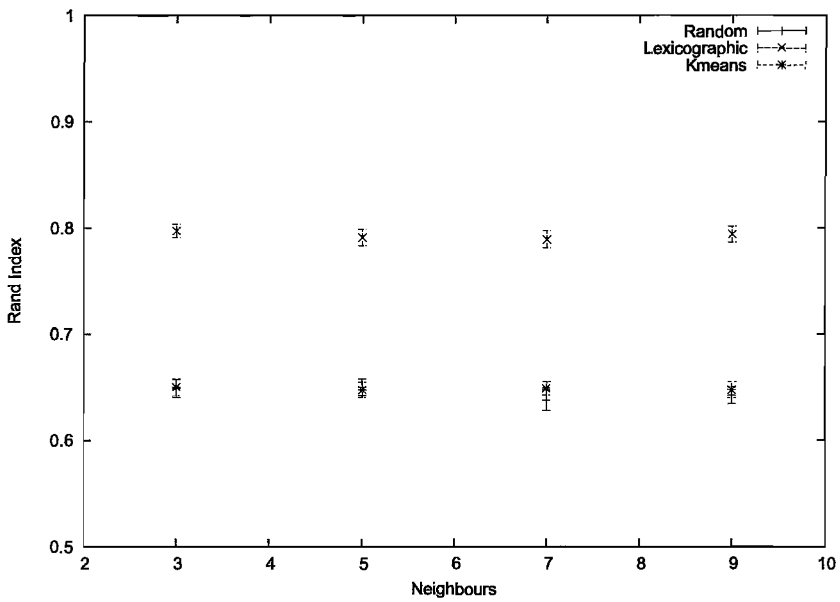


Figure 7.11: Comparison of Neighbours for Code #2, 90%/10% — Verification

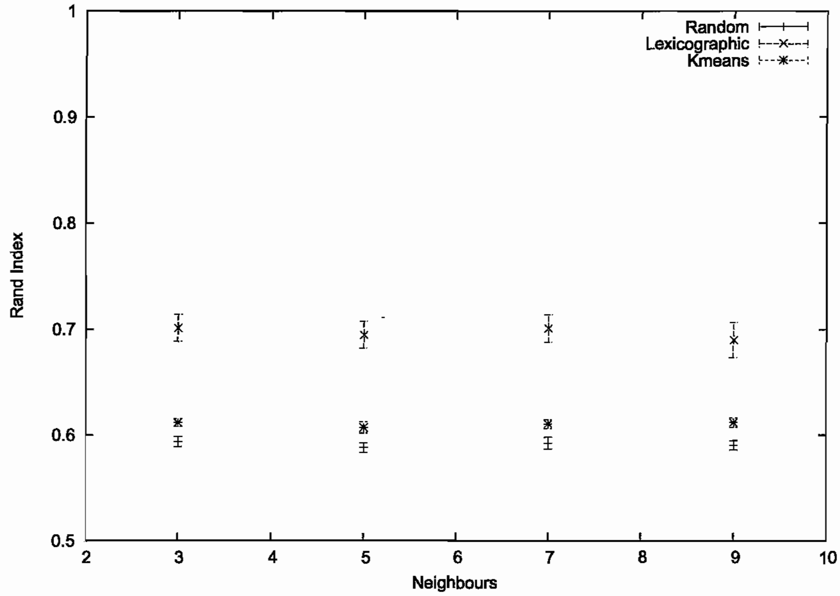


Figure 7.12: Comparison of Neighbours for Code #2, 0%/50% — Training

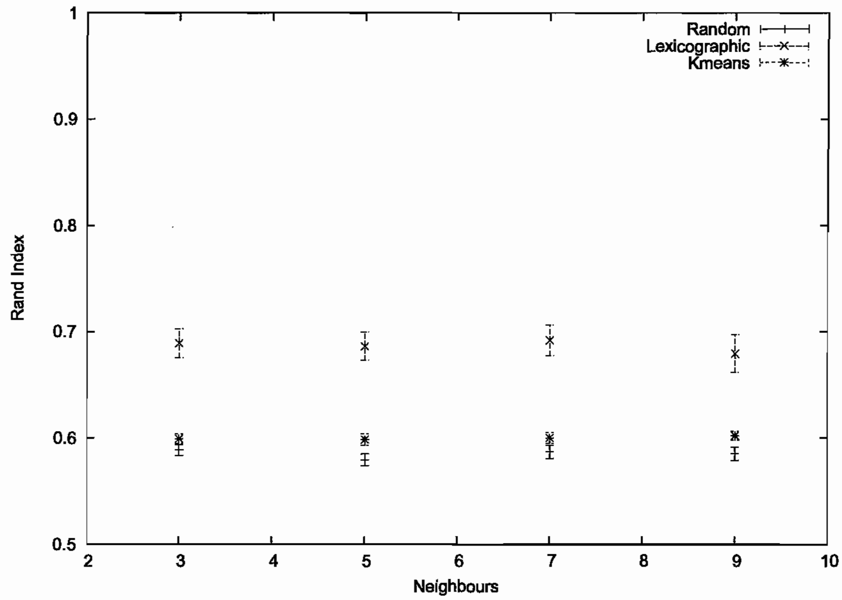


Figure 7.13: Comparison of Neighbours for Code #2, 0%/50% — Verification

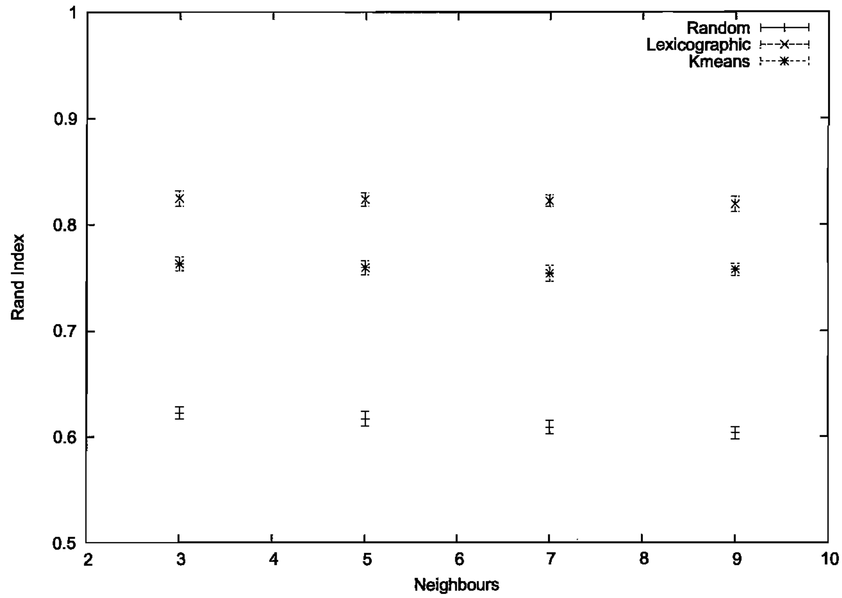


Figure 7.14: Comparison of Neighbours for Code #3, 90%/10% — Training

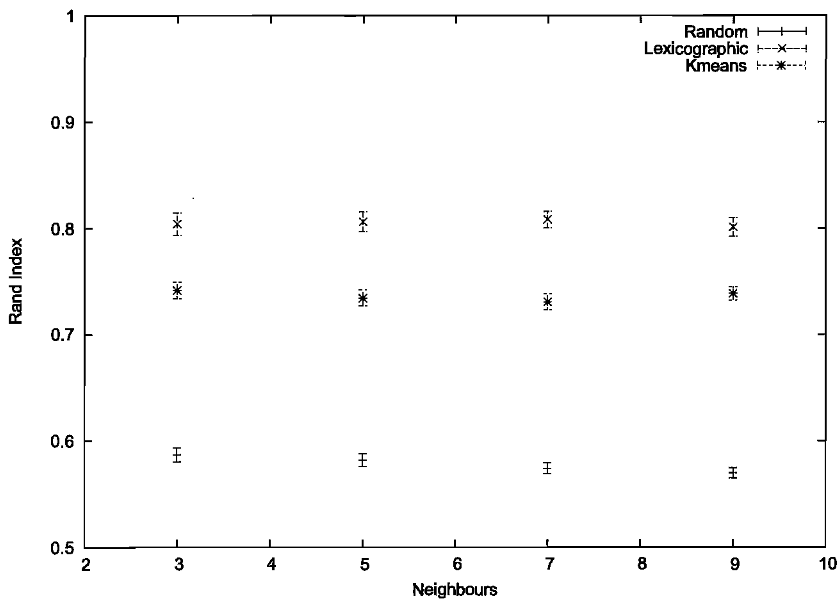


Figure 7.15: Comparison of Neighbours for Code #3, 90%/10% — Verification

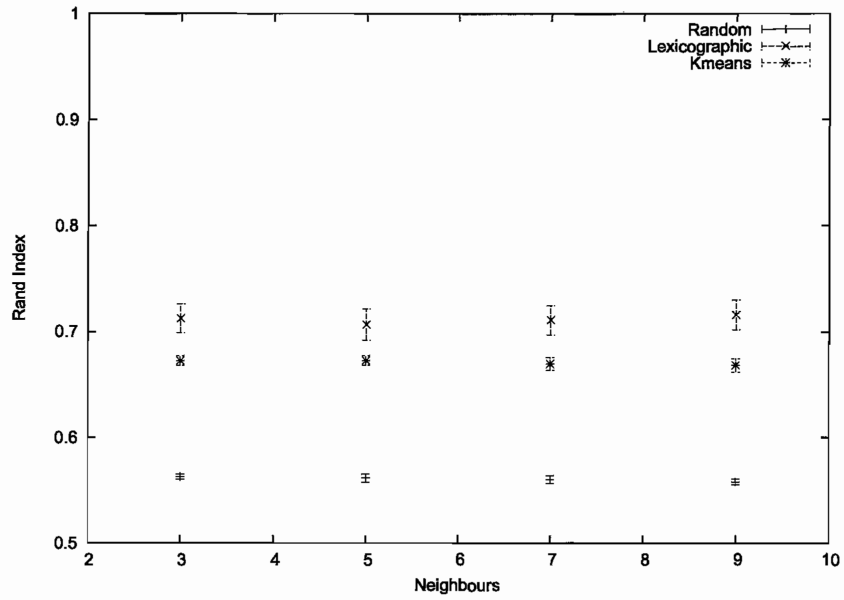


Figure 7.16: Comparison of Neighbours for Code #3, 0%/50% — Training

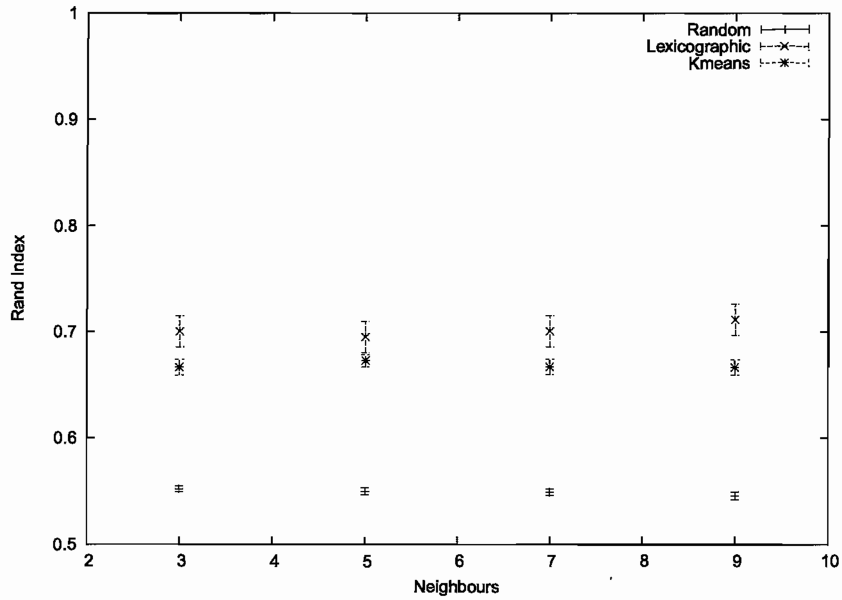


Figure 7.17: Comparison of Neighbours for Code #3, 0%/50% — Verification

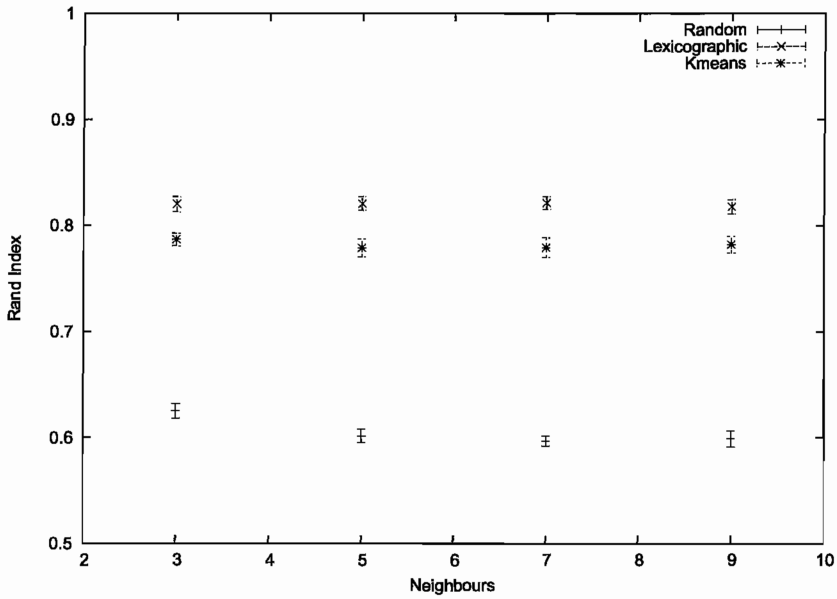


Figure 7.18: Comparison of Neighbours for Code #4, 90%/10% — Training

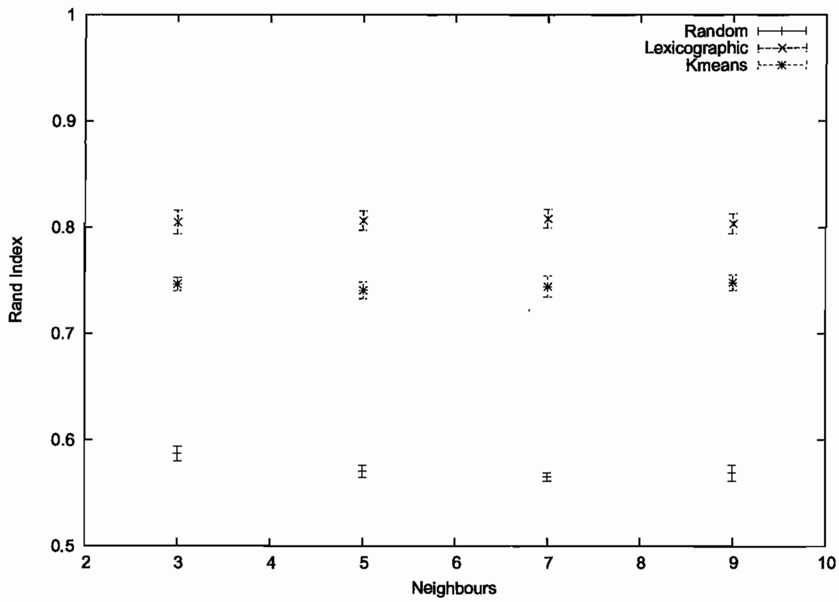


Figure 7.19: Comparison of Neighbours for Code #4, 90%/10% — Verification

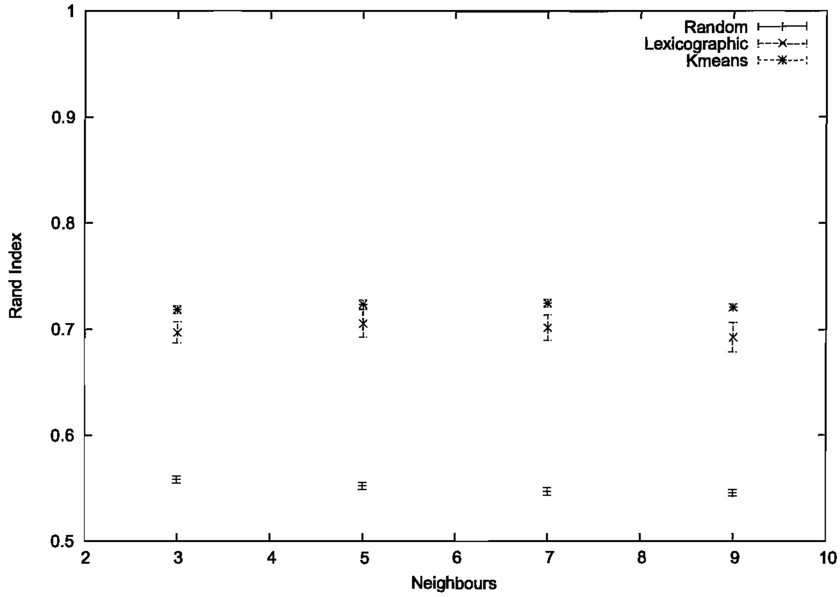


Figure 7.20: Comparison of Neighbours for Code #4, 0%/50% — Training

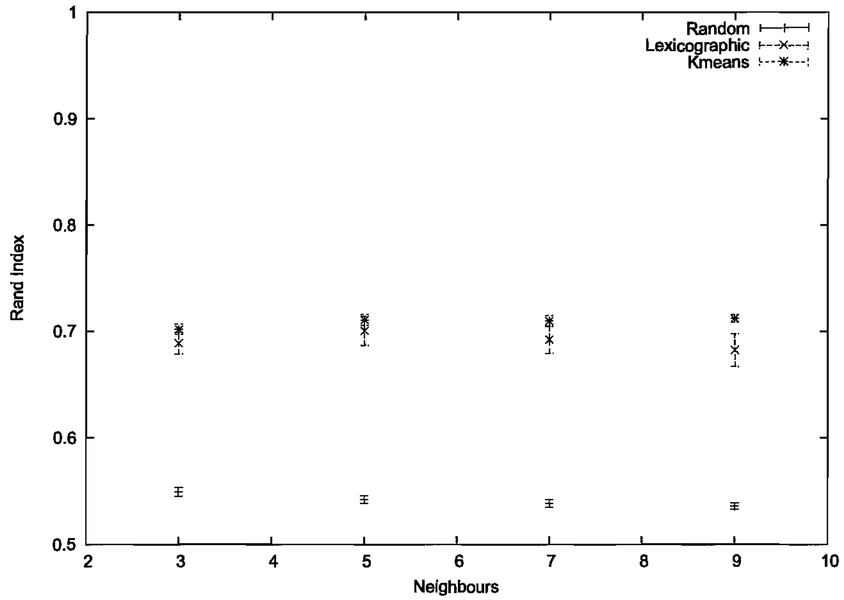


Figure 7.21: Comparison of Neighbours for Code #4, 0%/50% — Verification

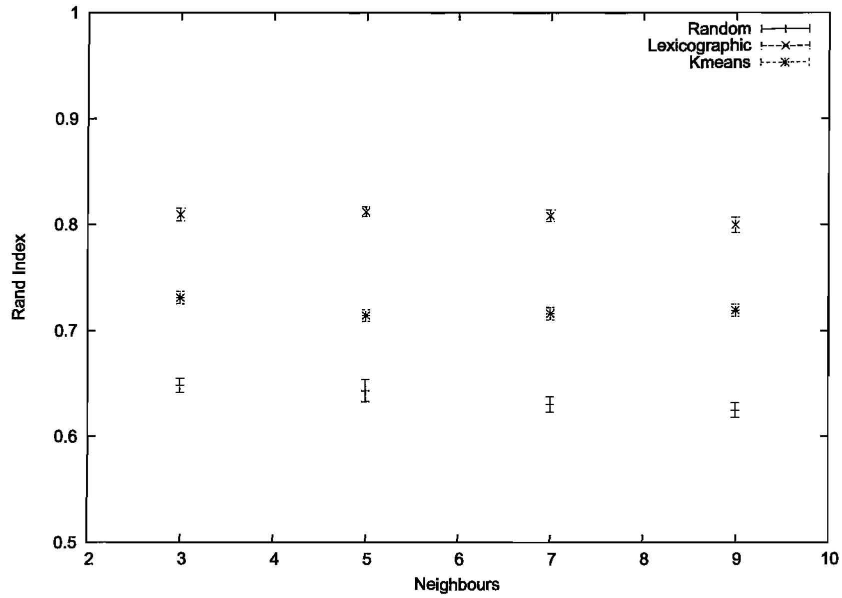


Figure 7.22: Comparison of Neighbours for Code #5, 90%/10% — Training

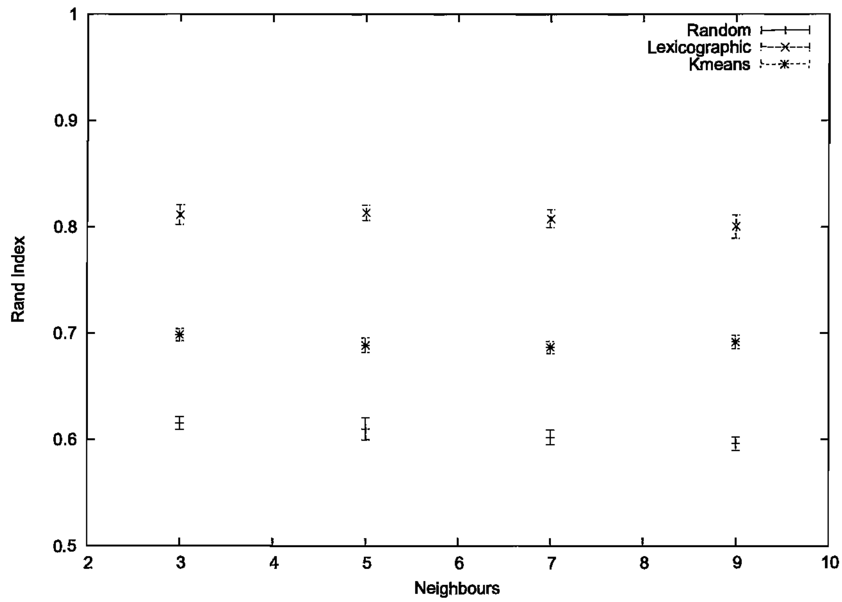


Figure 7.23: Comparison of Neighbours for Code #5, 90%/10% — Verification

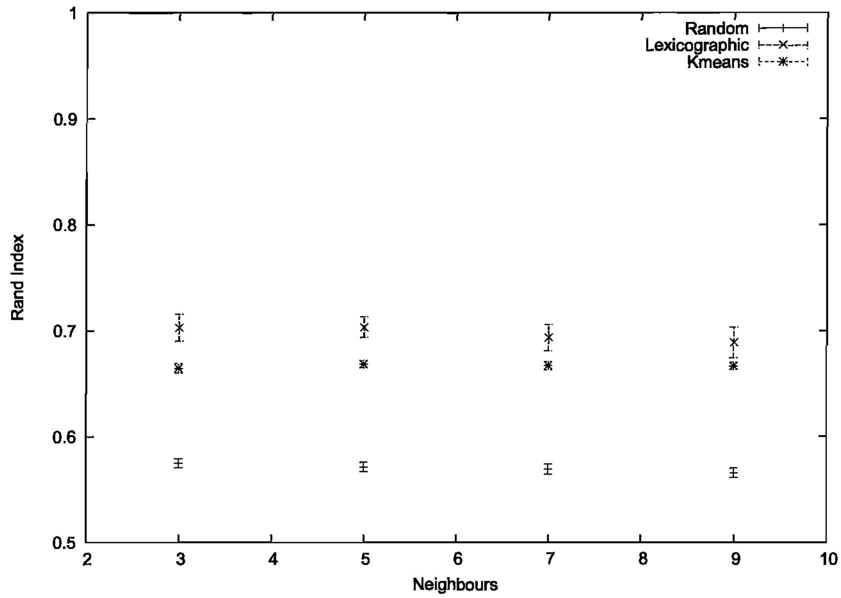


Figure 7.24: Comparison of Neighbours for Code #5, 0%/50% — Training

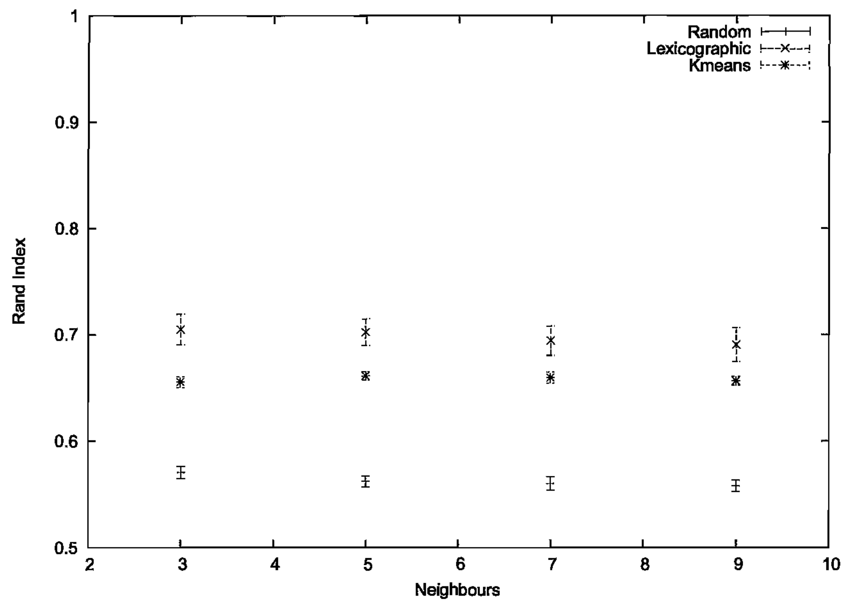


Figure 7.25: Comparison of Neighbours for Code #5, 0%/50% — Verification

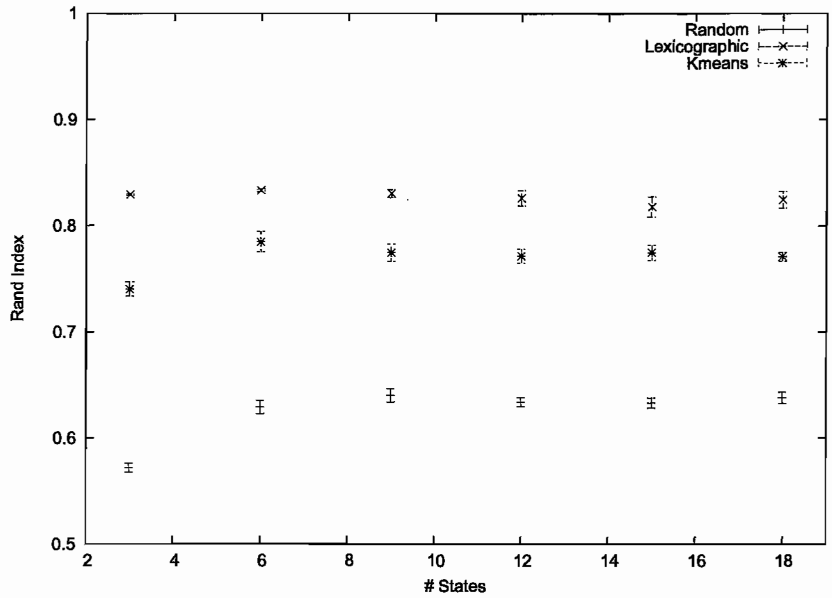


Figure 7.26: Comparison of States for Code #1, 90%/10% — Training

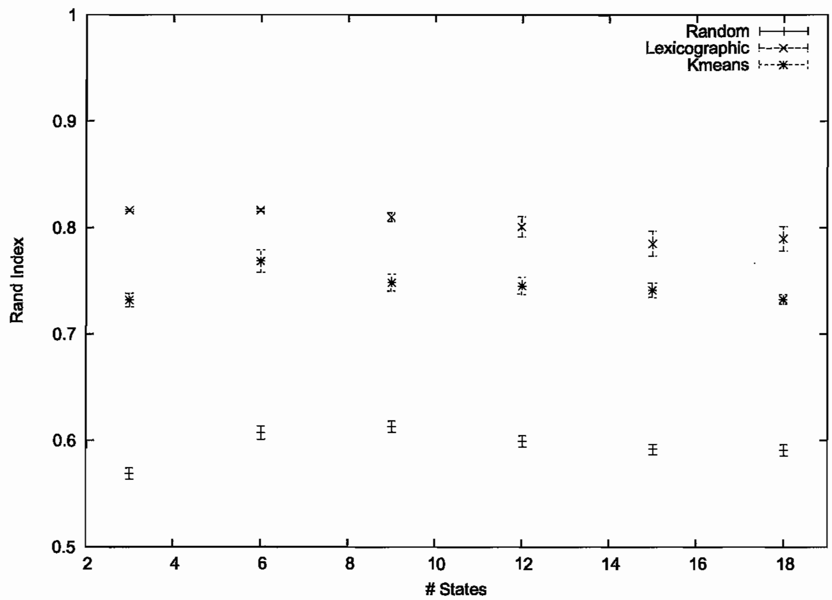


Figure 7.27: Comparison of States for Code #1, 90%/10% — Verification

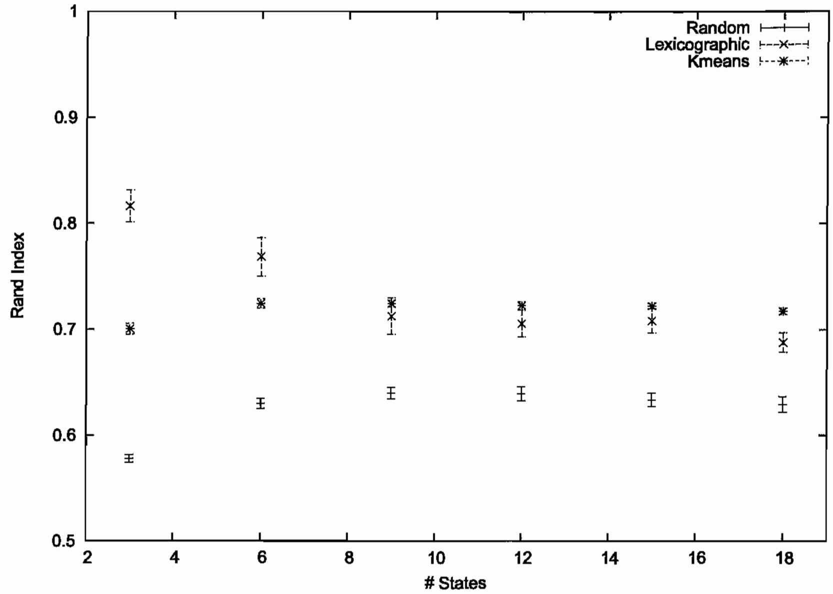


Figure 7.28: Comparison of States for Code #1, 0%/50% — Training

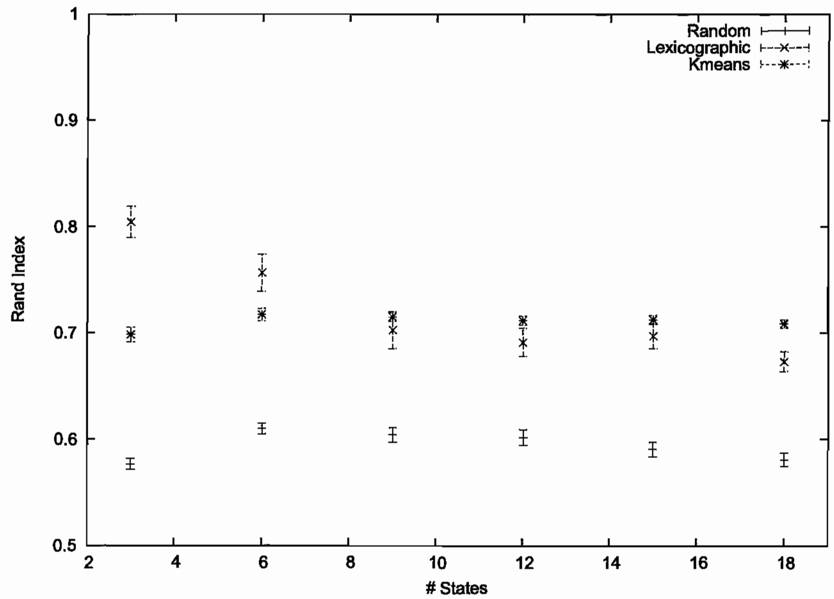


Figure 7.29: Comparison of States for Code #1, 0%/50% — Verification

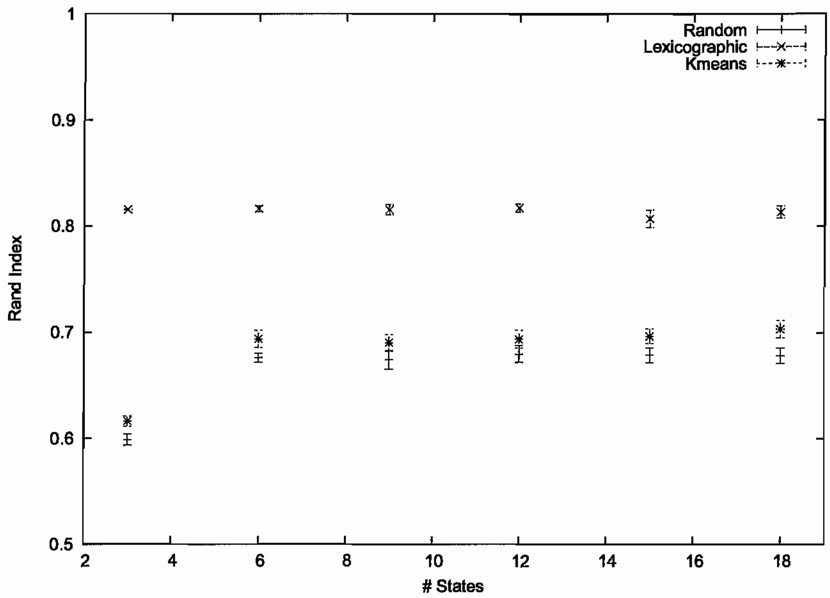


Figure 7.30: Comparison of States for Code #2, 90%/10% — Training

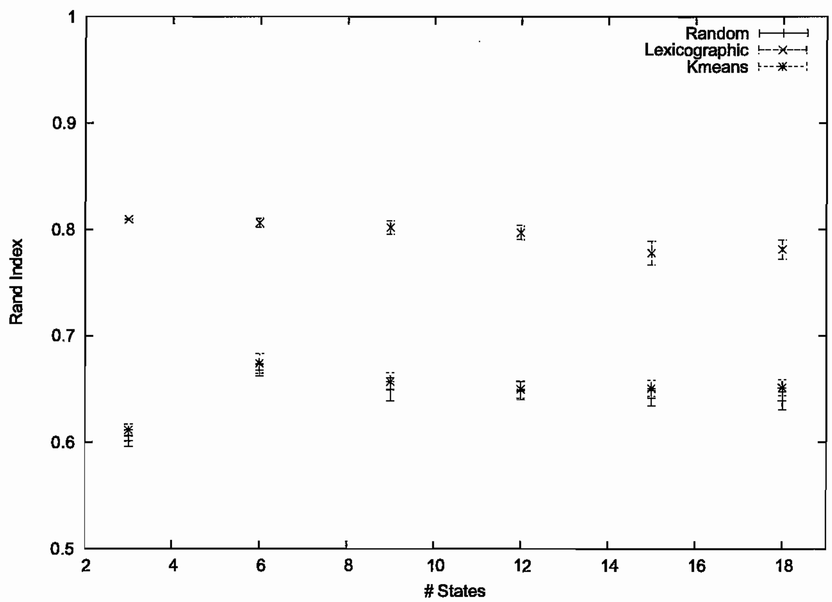


Figure 7.31: Comparison of States for Code #2, 90%/10% — Verification

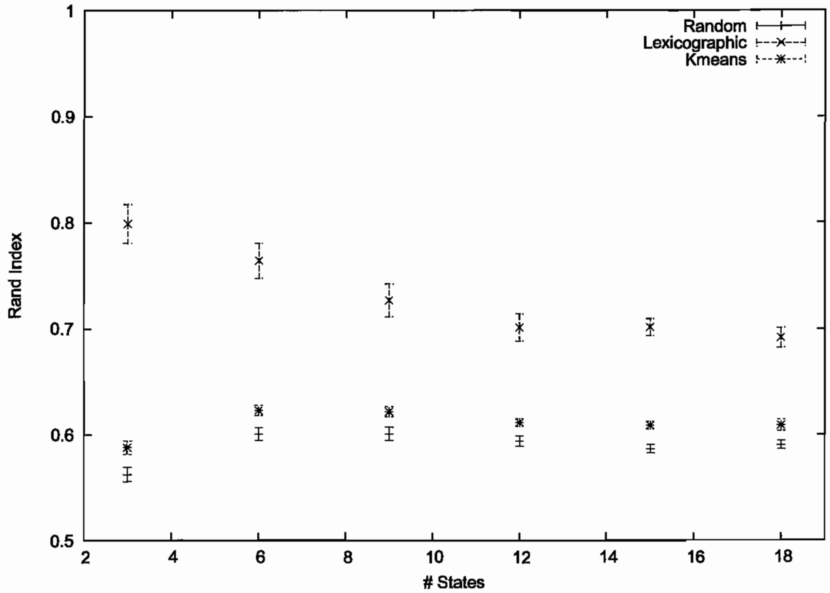


Figure 7.32: Comparison of States for Code #2, 0%/50% — Training

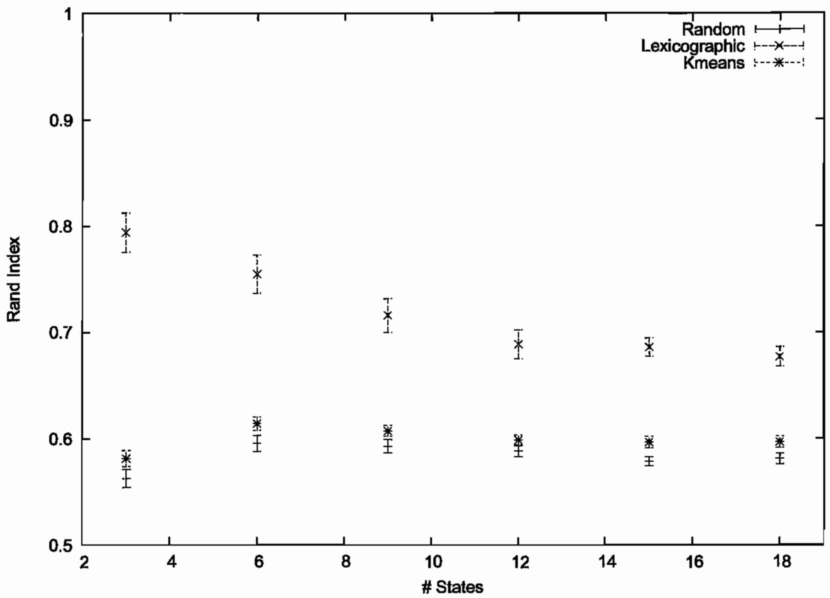


Figure 7.33: Comparison of States for Code #2, 0%/50% — Verification

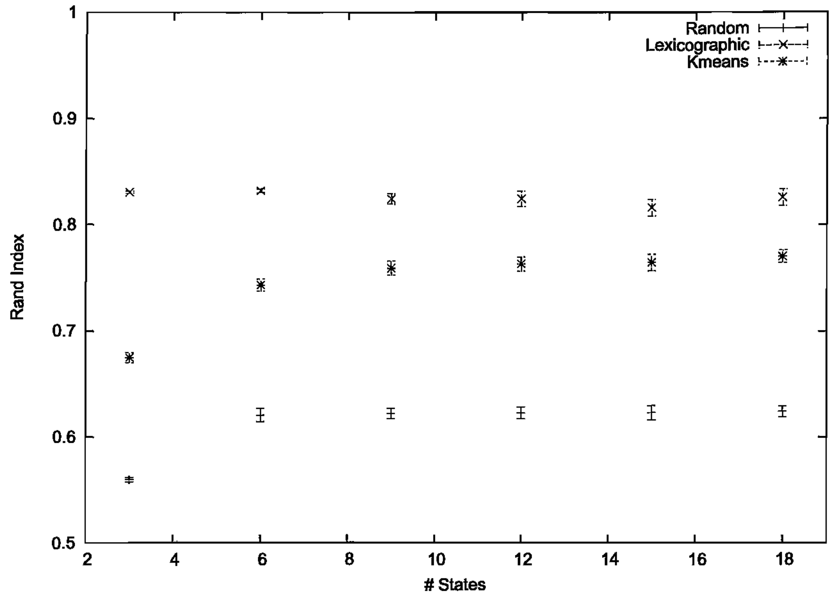


Figure 7.34: Comparison of States for Code #3, 90%/10% — Training

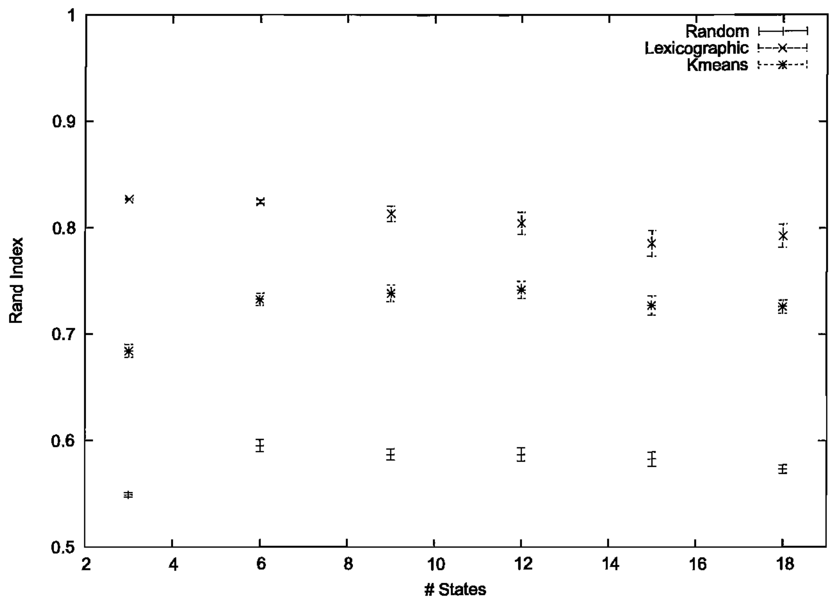


Figure 7.35: Comparison of States for Code #3, 90%/10% — Verification

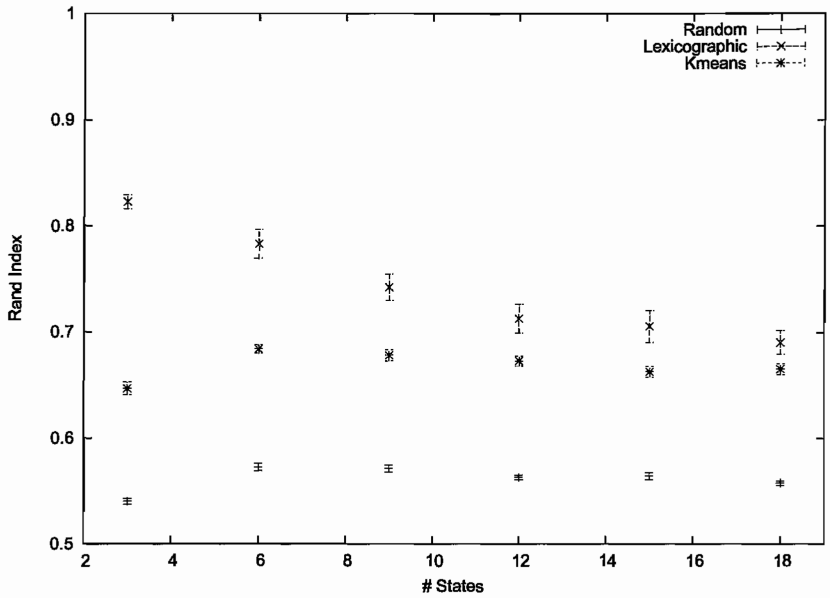


Figure 7.36: Comparison of States for Code #3, 0%/50% — Training

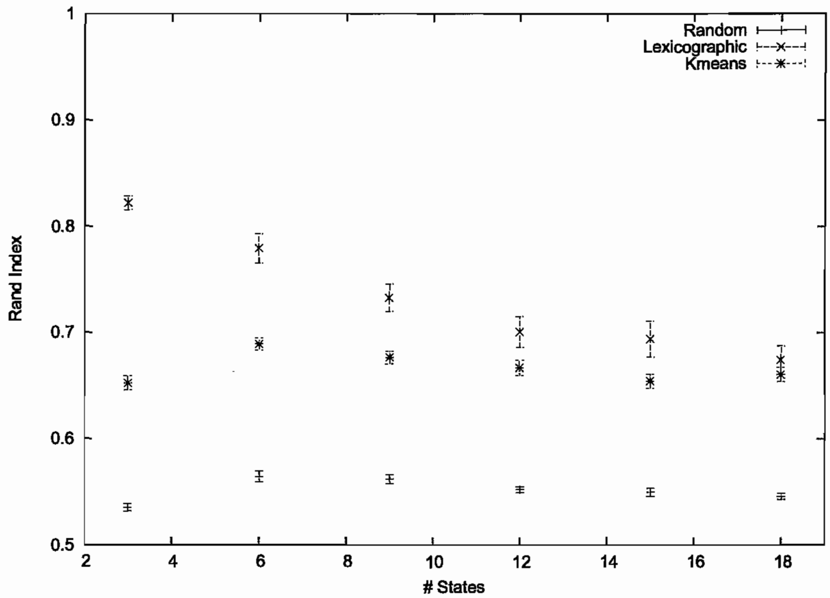


Figure 7.37: Comparison of States for Code #3, 0%/50% — Verification

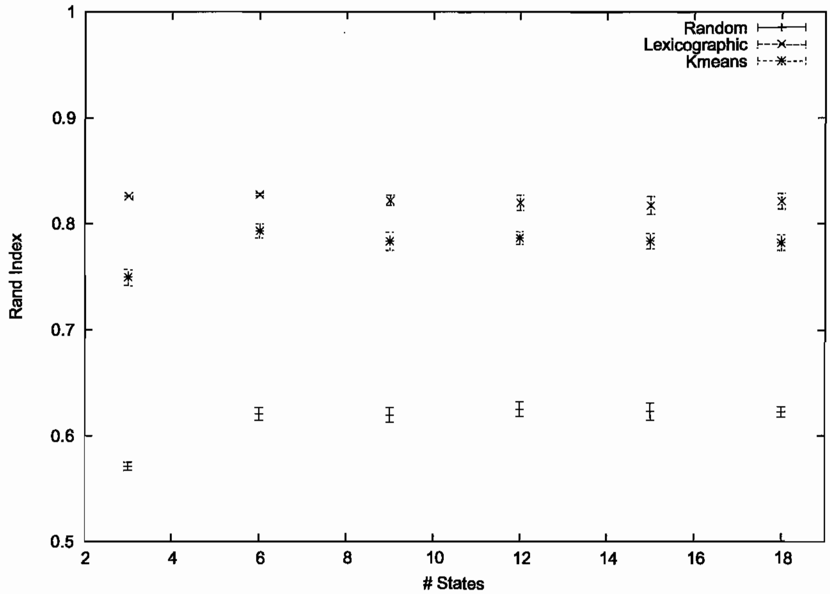


Figure 7.38: Comparison of States for Code #4, 90%/10% — Training

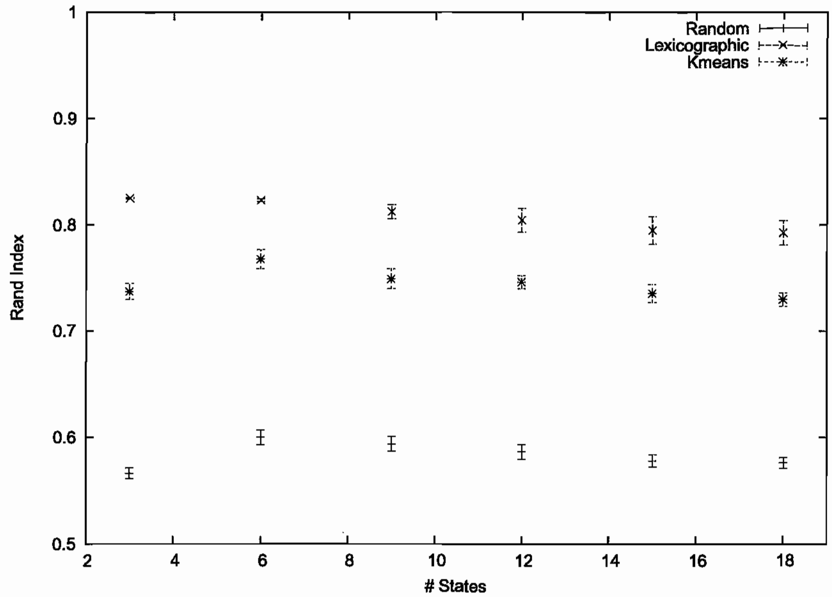


Figure 7.39: Comparison of States for Code #4, 90%/10% — Verification

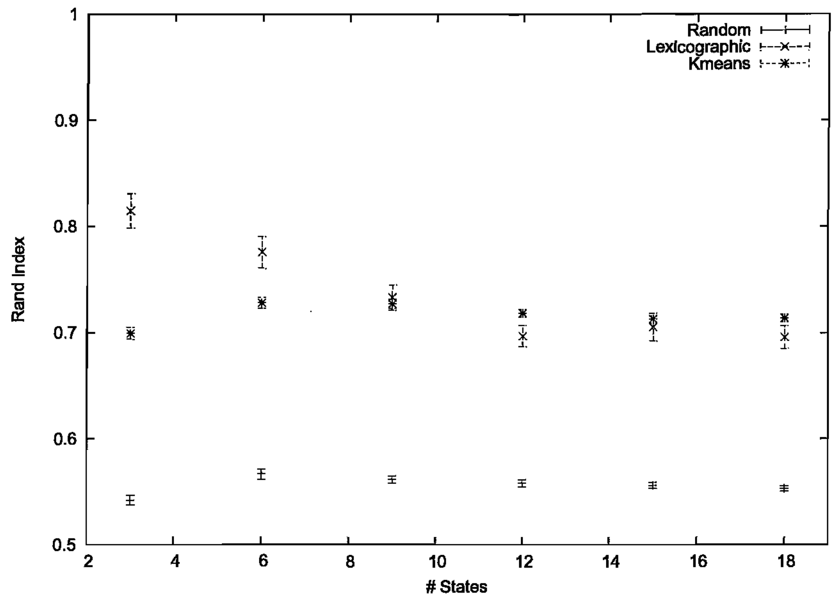


Figure 7.40: Comparison of States for Code #4, 0%/50% — Training

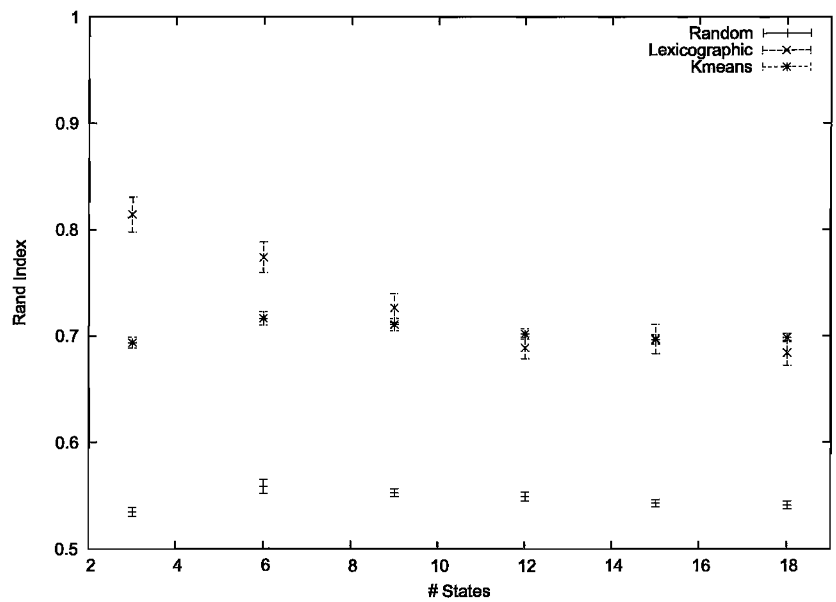


Figure 7.41: Comparison of States for Code #4, 0%/50% — Verification

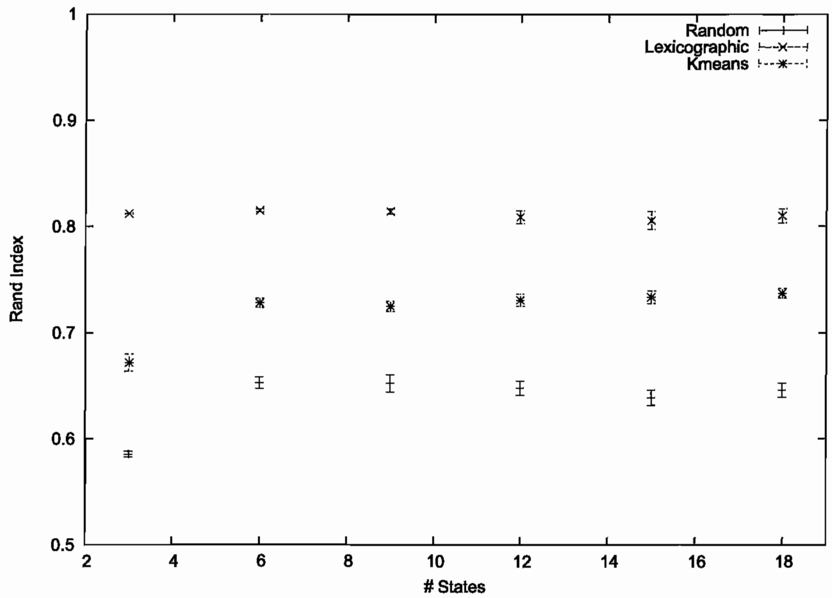


Figure 7.42: Comparison of States for Code #5, 90%/10% — Training

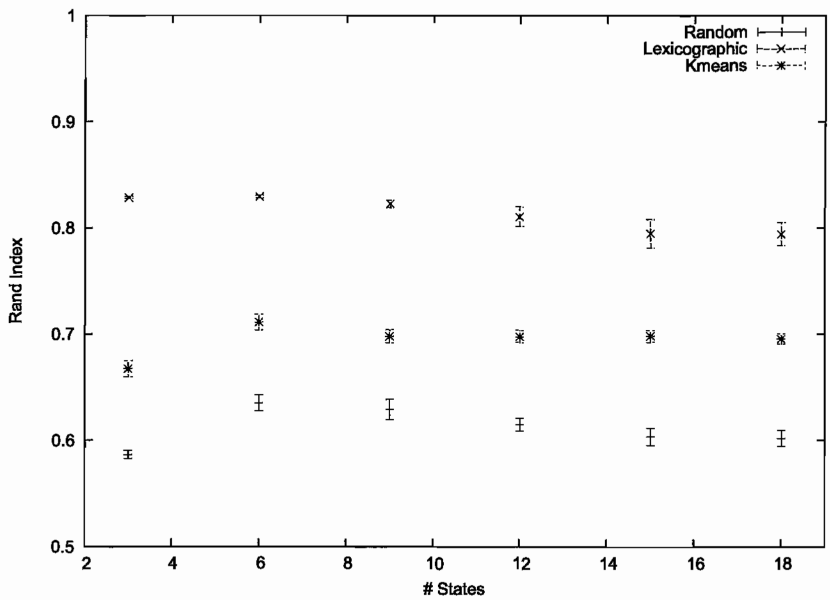


Figure 7.43: Comparison of States for Code #5, 90%/10% — Verification

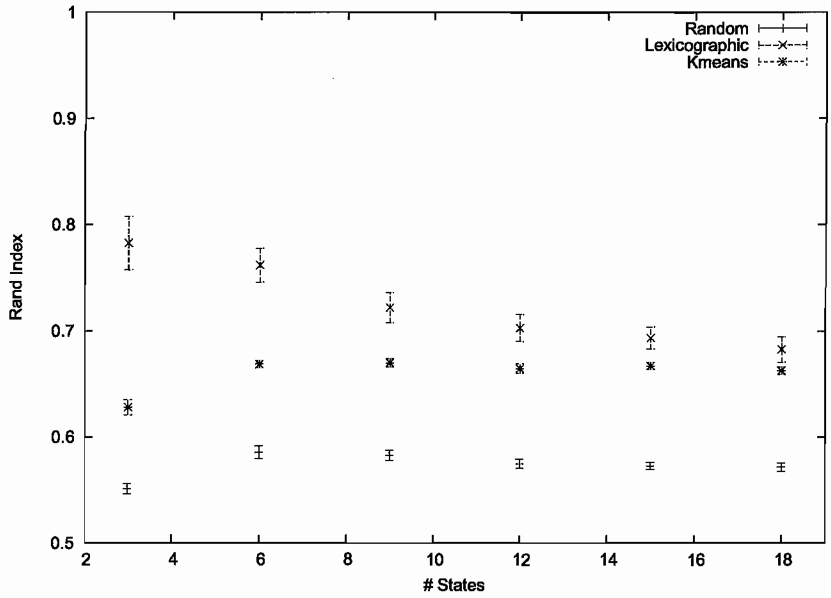


Figure 7.44: Comparison of States for Code #5, 0%/50% — Training

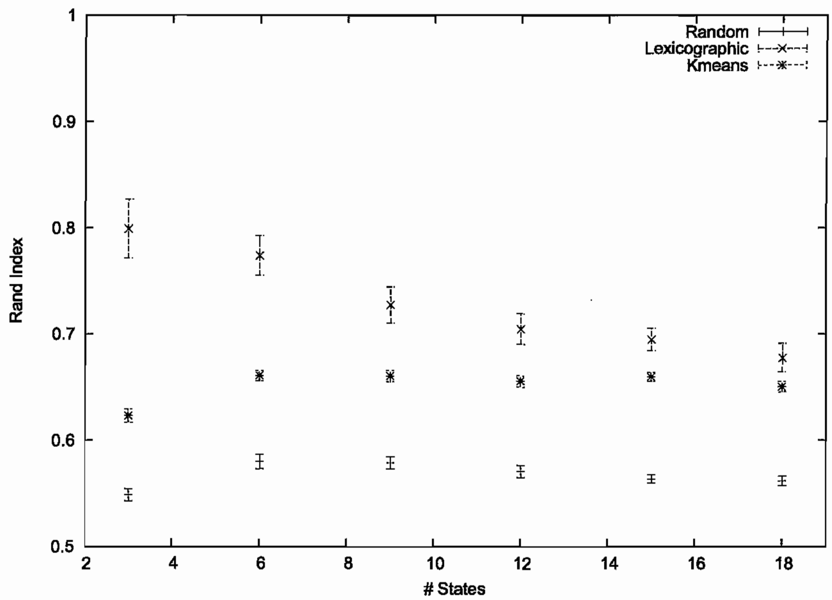


Figure 7.45: Comparison of States for Code #5, 0%/50% — Verification

Chapter 8

Conclusion

8.1 Side Effect Machines for Decoding

Side effect machines are small, efficient, and most importantly simple to understand. They are generalizations of finite state machines. They are also powerful when used for bioinformatics applications. This has directed their use towards the decoding of error correcting codes.

Error correcting codes for use in bioinformatics do not always use the well-understood and well-used Hamming metric, but instead sometimes use the edit metric in order to represent errors caused by the insertion, deletion and substitution of base pairs. These errors were caused either while the organism is out in the field or during the sequencing. Error correcting codes for these uses have been examined. Research into the decoders is lacking, especially since codes which take into consideration biological restrictions will not often have well defined stuctures.

This thesis aimed to contribute to these missing areas. Two new types of generalized decoders were examined: Single Classifier Machines and Locking Side Effect Machines. Both show promise for correcting the type of errors shown by biology while taking into account the restrictions. They are generalized decoders for the edit metric.

Single classifier machines work by using a single side effect machine to classify all words within a code. As they use the Euclidean metric, instead of the edit metric, there is a reduction in the runtime to properly classify a code. This runtime however has the cost of making the decoder probabilistic. There is a gain in speed for a loss to the correction ability. Therefore, fuzziness is added with the idea that the single classifier machine will work as a sorting algorithm for the codewords inside of a linear search. This sorting allows for gains in runtime as it moves the most likely codewords forward at less than the cost of measuring the edit metric of one codeword. Classifications were found that are correct over eighty percent of the time on distance one errors; the runtime is substantially reduced for the most frequent of errors. All codewords also are correctly classified in the minimal amount of runtime.

Locking side effect machines use the idea of breaking the code into subclassifications. Multiple side effect machines work together in a tree structure, each making a classification as to what side effect machine to use next, until the result is fully classified. This requires an exponential number of machines to the problem instance, and so usually this tree structure would end at some point and the final classification would be done by single classifier machines or a linear search. The hard work of breaking down the code would allow for a better final classification.

The use of genetic algorithms is required as there is a large search space and because a deterministic creation was shown to have intractable cost for its creation. The genetic algorithms do not require complete enumerations of the errors, and the subset of errors it requires can be created in minimal time by taking codewords and causing errors up to a bound, using a random number generator.

The decoder is allowed to have a longer time to be created as its runtime cost is offset by the number of corrections it makes; it only needs to be created once while it can be indefinitely used as a decoder. However, as these biological codes are generalized and are used for specific purposes, we need to be able to generate decoders within a reasonable runtime. This is provided by using decoders created by evolution as we can take the best decoder we find within a set runtime and know that it is a reasonable approximation for the amount of time available for a search.

Side effect machines have been shown to have the ability to be used in bioinformatics as a tool for solving problems in decoding. Their use adds new tools that are used to understand the code of life itself. It has not escaped attention, that in the future these roles will need to be expanded upon for use in bioinformatics and other applications.

8.2 Future Work

8.2.1 Side Effect Machines for Decoding

While this marks the end of this thesis on the topic of Side Effect Machines for decoding, we have barely scratched the surface in terms of the capabilites of these machines. The creation of the Side Effect Machines shows the most room for new techniques. This includes, introducing an ability for a SEM to have feature selection and extraction. This can be accomplished by adding to the chromosome another vector which acts to flag which states counters would be used in the classification vector. Modifiying the SEM by adding to a chromosome a vector of values giving the increment by which a counter for that state will increase would also allow for a variation of feature selection. Both modifications would increase the size of the search space, and therefore, the representational ability of a smaller SEM. This might lead to better solutions and better runtimes by allowing for a smaller machine.

From using LSEMs, we have a subclassification at each level, and we must measure the Euclidean distance to a subset of those M words at each level when we use KNN. However, by using K-means points instead of the codeword's classification vectors, we would only need to measure a codeword to the saved K centroids. This would reduce the runtime, because we only look at a constant number of calculations of the Euclidean distance, and also reduce space complexity, since we only need a constant number to save the Centroid classification vector. This idea was not implemented currently due to concerns over how the K-means would work when the space is not necessarily easily separable. If we could find an algorithm which acts like K-means but which does not have these concerns of hyperplane-separability, then there can be a change to the number of Euclidean measures to a constant value of the K value.

Perhaps more importantly, the SCM and LSEMs should be applied to other codes. Only a small subset of the various codes that exist have been tested and there are infinitely many codes. It would be of interest to find an underlying structure relating good SEM decoders to their codes. This leads to a further idea for future work, namely creating the code in conjuction with its decoder.

8.2.2 Error Correcting Codes and Decoders

Error correcting codes allow for transmission of data even when there is noise by introducing redundant data. This data allows us to repair or detect errors made during transmission or transcription. How we add this redundancy has effects on the ability of the code to be used in various applications.

Error correcting codes have a number of properties which define their usefulness for a particular task. These properties include: the number of errors that can be corrected, the number of codewords and the ease of decoding. All these properties may be in conflict. For example, looking at two codes of a given length, the one which corrects more errors will usually have fewer codewords. Sometimes, codes with useful properties are very difficult to decode based on their structure. Conversely, good decoders do not necessarily correspond to useful codes.

The idea of generating a code or decoder first has a potential flaw as we sacrifice some properties for others. By using evolutionary computation techniques side-effect machine decoders will be created and the code they require will be extracted from the decoder. Seeing the code and decoder, the technique will score the decoder and code based on the set of useful properties: the number of codewords, error correction ability, ease of decoding, etc. As these decoders are being used for bioinformatics, the created code and decoder may need to take into account biological restrictions; for example, some DNA strings, or combination of strings, cannot be used in applications. These restrictions vary depending on the application. A wide variety of codes are required.

8.2.3 Side Effect Machines for Data Mining

The Side Effect Machine acts as a general classifier and could have uses beyond those in bioinformatics. One such place is in data mining, with the idea of classification of a consumer on an Internet shopping website. Using a SEM to track pages and moves between them would be natural — page 'hits' and links have similar functions to the counter on a state and the links between states. Such a classifier would allow the monitoring of behaviours allowing the site to predict, based on past profiles, how a consumer will act. This would allow for the introduction of targeted advertisements or promotions.

Bibliography

[1] Alfred V. Aho and Margaret J. Corasick. Efficient string matching: an aid to bibliographic search. *Commun. ACM*, 18(6):333–340, 1975.

[2] D. Ashlock, Ling Guo, and Fang Qiu. Greedy closure evolutionary algorithms. In *CEC '02: Proceedings of the Evolutionary Computation on 2002. CEC '02. Proceedings of the 2002 Congress*, pages 1296–1301, Washington, DC, USA, 2002. IEEE Computer Society.

[3] Daniel Ashlock. *Evolutionary Computation for Modeling and Optimization.* Springer, 2006.

[4] Daniel Ashlock and Sheridan Houghten. DNA error-correcting codes : No crossover. In *Proceedings of the 2009 IEEE Symposium on Computational Intelligence in Bioinformatics and Computational Biology*, pages 38–45, 2009.

[5] Daniel Ashlock and Elizabeth Warner. Classifying synthetic and biological DNA sequences with side effect machines. In *Proceedings of the 2008 IEEE Symposium on Computational Intelligence in Bioinformatics and Computational Biology*, pages 22–29, 2008.

[6] Daniel Ashlock and Elizabeth Warner. Side effect machines for sequence classification. In *Proceedings of the Canadian Conference on Electrical & Computer Engineering 2008*, pages 1453–1456, 2008.

[7] Stephen Baker, Robert Flack, and Sheridan Houghten. Optimal variable-length insertion-deletion correcting codes and edit metric codes. *Congressus Numerantium*, 186:65–80, 2007.

[8] Joseph A. Brown, Sheridan K. Houghten, and Daniel A. Ashlock. Edit metric decoding: a new hope. In *C3S2E '09: Proceedings of the 2009 C3S2E conference*, pages 233–242, New York, NY, USA, 2009. ACM.

[9] Jessie Katherine Campbell. *Enumeration and Symmetry of Edit Metric Spaces.* PhD thesis, Iowa State University, 2005.

[10] J. H. Conway and N. J. A. Sloane. Lexicographic codes: Error-correcting codes from game theory. *IEEE Trans. Inf. Theor.*, 32(3):337–348, 1986.

[11] T. Cover and P. Hart. Nearest neighbor pattern classification. *Information Theory, IEEE Transactions on*, 13(1):21–27, 1967.

[12] F. H. C. Crick, J. S. Griffith, and L. E. Orgel. Codes without commas. In *Proceedings of the National Academy of the Sciences of the USA*, volume 43, pages 416–421, 1957.

[13] Charles Darwin. *On Natural Selection.* Penguin Books, London, 2004.

[14] Matthew C. Davey, David J. C. Mackay, and Cavendish Laboratory. Watermark codes: Reliable communication over insertion /deletion channels. In *ISIT 2000*, page 47, 2000.

[15] Evelyn Fix and J. L. Hodges, Jr. Discriminatory analysis: Nonparametric discrimination: Consistency properties. Technical Report Project 21-49-004, Report Number 4, USAF School of Aviation Medicine, Randolf Field, Texas, 1951.

[16] Lawarence J. Fogel, Alvin J. Owens, and Michael J. Walsh. *Artifical Inteligence through Simulated Evolution.* John Wiley & Sons, New York, 1966.

[17] Errol C. Friedberg, Graham C. Walker, and Wolfram Siede. *DNA Repair and Mutagenesis*, chapter 2. American Society for Microbiology Press, 1995.

[18] David E. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning.* Addison-Wesley Professional, January 1989.

[19] R. W. Hamming. Error detecting and error correcting codes. *Bell System Tech. J.*, 29:147–160, 1950.

[20] Douglas R. Hofstadter. *Godel Escher Bach: An Eternal Golden Braid.* Basic Books, Inc., New York, NY, USA, 1999.

[21] John H. Holland. *Adaptation in natural and artificial systems.* MIT Press, Cambridge, MA, USA, 1992.

[22] John E. Hopcroft, Rajeev Motwaniand, and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages, and Computation (3rd Edition).* Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2006.

[23] L. Hubert and P. Arabie. Comparing partitions. *Journal of Classication*, 2(1):193–218, 1985.

[24] W. C. Huffman and V. Pless. *Fundamentals of Error-Correcting Codes.* Cambridge University Press, USA, 2003.

[25] Sheridan K. Houghten, Dan Ashlock, and Jessie Lenarz. Construction of optimal edit metric codes. In *Proceedings of the 2006 IEEE Workshop on Information Theory (ITW 2006)*, pages 259–263, 2006.

[26] David Kahn. *The Codebreakers: The Comprehensive History of Secret Communication from Ancient Times to the Internet.* Scribner, rev sub edition, December 1996.

[27] Stavros Konstantinidis. Computing the edit distance of a regular language. *Inf. Comput.*, 205(9):1307–1316, 2007.

[28] Vladimir I. Levenshtein. Binary codes capable of correcting deletions, insertions, and reversals. *Soviet Physics Doklady*, 10(8):707–710, 1966.

[29] Peter Linz. *An Introduction to Formal Language and Automata.* Jones and Bartlett Publishers, Inc., USA, 2006.

[30] David J. C. Mackay. *Information Theory, Inference & Learning Algorithms.* Cambridge University Press, June 2002.

[31] John. R. Pierce. *An Introduction to Information Theory - Symbols, Signals and Noise, 2nd rev. ed.* Dover, New York, 1980.

[32] William M. Rand. Objective criteria for the evaluation of clustering methods. *Journal of the American Statistical Association*, 66(336):846–850, December 1971.

[33] Edward A. Ratzer and David J. C. Mackay. Codes for channels with insertions, deletions and substitutions. In *2nd International Symposium on Turbo Codes and Related Topics*, pages 149–156, 2000.

[34] J. F. F. Sellers. Bit loss and gain correction code. In *IEEE Transactions on Information Theory*, volume 8, pages 35–38, 1962.

[35] Claude E. Shannon. *A Mathematical Theory of Communication.* CSLI Publications, 1948.

[36] Roberto Togneri and Christopher J. S. DeSilva. *Fundamentals of Information Theory and Coding Design.* CRC Press, Inc., Boca Raton, FL, USA, 2003.

[37] Robert A. Wagner and Michael J. Fischer. The string-to-string correction problem. *J. ACM*, 21(1):168–173, 1974.

Appendix A

Edit Metric Error Correcting Codes

A.1 $(12, M, 7)_4$ Codes

2 0 3 3 2 2 2 0 0 0 3 0	1 2 2 2 0 1 3 2 3 1 1 1	1 2 1 0 0 2 1 0 3 2 2 2	2 0 1 0 2 2 0 2 2 3 2 3	3 2 3 1 2 1 3 2 2 2 2 2
0 0 2 1 2 3 1 1 2 1 2 2	3 2 2 3 3 3 1 1 2 3 3 3	1 1 2 0 3 1 1 2 2 2 0 1	3 3 3 0 3 3 3 2 3 0 0 0	3 3 0 0 2 0 3 1 2 0 3 1
3 2 2 2 0 0 0 0 0 3 3 1	2 1 0 2 1 1 1 1 1 1 1 1	1 1 1 2 0 0 3 1 3 3 0 3	3 3 3 2 2 2 2 1 0 1 2 1	0 0 3 2 2 1 3 2 0 3 3 1
2 0 0 1 1 1 0 2 3 3 0 1	0 0 1 0 0 3 0 0 0 0 1 1	1 3 0 0 0 0 0 2 0 0 2 2	3 1 1 0 3 0 1 1 1 3 1 0	2 2 0 0 1 2 1 2 0 1 3 2
2 1 1 1 2 3 2 1 2 3 3 1	1 3 1 2 1 1 0 2 1 1 2 3	1 2 2 3 2 2 0 2 2 2 1 0	1 0 0 2 1 3 3 0 2 0 0 3	0 2 2 2 3 0 3 2 0 0 0 2
2 2 2 2 1 1 3 1 2 2 3 0	3 0 1 2 1 2 1 2 3 0 2 0	3 3 2 1 1 1 1 0 3 1 3 0	1 0 3 0 3 1 2 3 3 3 3 0	2 2 1 1 0 0 0 3 0 1 0 2
3 2 3 1 0 2 1 0 1 0 0 0	3 1 2 2 2 2 3 3 3 3 3 2	0 1 1 1 3 1 3 0 0 3 0 0	2 0 0 0 0 3 3 3 3 2 0 2	3 3 3 0 1 1 0 0 2 2 1 1
0 3 3 1 1 1 1 2 2 0 0 3	0 3 0 1 3 2 3 2 0 2 1 3	1 0 0 3 3 3 2 2 2 2 2 2	2 1 2 3 0 2 3 0 2 0 1 1	0 1 2 0 0 3 2 2 1 1 0 0
2 0 3 3 0 1 0 3 1 1 2 1	2 2 3 3 3 3 3 3 0 2 2 3	1 1 1 3 2 2 2 1 1 2 0 2	0 0 0 0 0 1 3 1 1 1 3 3	3 3 1 3 3 0 0 0 2 1 0 0
1 1 0 0 3 3 0 0 1 2 3 3	2 3 2 3 3 2 2 3 3 1 3 1	0 1 1 3 1 0 3 3 2 1 3 2	1 3 3 3 1 0 0 0 1 3 3 2	0 0 0 2 2 0 0 0 1 2 2 0
0 2 1 3 3 2 0 1 3 3 2 0	3 1 1 2 3 3 0 3 1 0 2 2	1 2 3 1 1 1 1 3 0 3 2 1	2 0 2 2 3 1 0 0 1 3 1 3	0 1 3 3 3 3 3 1 3 1 1 1

Table A.1: $(12, 55, 7)_4$ Code — Code #1

0 0 3 0 2 3 1 2 2 3 3 3	2 2 2 3 1 0 1 1 0 2 2 2	1 0 0 0 2 1 3 1 3 0 3 2	2 2 3 2 2 1 3 3 1 1 3 3	2 3 0 0 0 2 2 0 3 3 3 3
0 3 1 3 1 3 0 2 1 1 1 1	0 3 2 2 2 0 3 2 3 3 1 1	1 2 1 2 0 0 1 0 3 2 3 2	0 1 0 2 2 0 0 3 3 0 2 2	0 0 2 0 3 1 3 1 0 2 1 0
3 3 3 3 1 2 1 3 3 3 0 3	0 3 1 1 1 1 1 1 3 3 2 0	0 3 2 3 2 1 2 2 0 1 0 2	0 3 1 2 0 2 3 1 1 2 2 2	2 2 2 1 2 3 2 2 2 0 1 3
1 1 0 3 3 3 3 3 3 2 0 1	0 0 3 0 0 0 0 1 1 1 3 2	0 1 2 3 3 3 0 2 0 0 3 3	2 1 3 3 1 2 3 2 1 3 1 0	1 1 2 1 0 0 0 3 3 1 1 3
3 0 2 1 3 3 3 0 3 3 0 0	0 3 0 1 1 0 0 0 0 2 3 3	2 2 3 3 0 0 1 2 2 1 0 3	0 0 0 1 0 0 2 2 3 1 0 1	0 2 2 2 1 1 0 0 0 3 0 1
2 3 3 0 3 3 3 0 0 1 3 0	2 0 1 1 1 3 0 2 3 3 3 2	1 2 2 0 1 2 1 1 2 3 3 1	0 2 1 0 3 3 2 2 0 2 2 1	1 0 1 3 1 2 2 2 2 2 0 2
1 1 2 2 2 2 3 3 2 0 3 0	1 1 0 3 3 0 1 1 1 3 3 3	3 3 3 2 2 3 0 2 3 1 2 3	1 1 1 2 1 1 1 1 2 2 1 3	3 3 2 2 1 3 1 2 1 0 0 0
1 2 3 3 3 2 0 3 1 1 2 0	3 2 3 1 1 3 3 0 0 2 1 2	2 1 1 0 2 0 0 2 2 2 2 0	1 1 1 1 1 2 2 3 0 1 2 1	3 0 0 1 1 1 2 3 2 0 3 0
3 3 0 0 0 3 3 2 2 3 2 0	3 0 0 0 0 1 2 0 0 2 2 2	1 0 2 1 0 0 1 1 2 0 0 0	1 3 0 1 2 0 1 3 0 0 1 1	3 1 1 3 2 3 3 1 2 3 3 1
2 0 1 3 1 1 1 1 1 0 0 1	2 2 0 2 2 3 0 0 3 2 1 1	3 2 2 0 2 0 0 0 1 0 0 3	1 2 1 3 1 3 3 1 3 0 2 3	3 1 0 0 2 1 1 0 1 1 3 1
2 3 1 0 1 0 0 3 0 1 0 2	1 1 1 3 2 2 1 0 0 0 0 3	2 2 0 0 3 3 1 3 3 1 2 2	3 3 3 3 0 0 2 0 2 0 2 1	3 0 1 2 2 2 2 1 1 1 1 0
3 0 3 0 3 2 1 1 3 2 2 1				

Table A.2: $(12, 56, 7)_4$ Code — Code #2

1 1 2 3 3 1 0 0 1 2 1 2	0 0 2 0 2 0 2 1 2 2 1 1	1 3 1 2 0 3 2 2 2 2 3 1	3 3 0 0 2 3 3 3 1 1 2 2	0 3 3 3 3 2 3 3 2 3 0 1
0 1 3 1 1 1 2 2 2 1 1 3	1 0 0 3 3 2 1 1 1 3 1 1	1 1 1 1 0 3 1 1 1 0 0 1	1 3 2 0 0 1 1 1 1 1 3 2	3 0 0 1 0 3 0 1 3 0 1 2
1 2 0 0 3 1 3 3 2 1 0 0	0 3 1 2 1 3 3 2 0 0 0 2	3 0 1 2 2 1 3 2 2 3 1 2	0 0 0 3 2 2 3 0 3 2 2 2	3 2 2 3 0 2 0 2 0 1 1 3
3 1 1 3 3 3 3 2 1 3 3 3	2 2 3 2 1 3 3 2 1 1 2 3	3 3 3 0 2 1 1 1 3 3 0 0	2 0 3 3 0 3 1 1 2 3 2 2	0 2 2 3 0 0 2 2 3 2 2 0
2 1 3 0 3 0 0 2 1 1 0 1	0 2 1 2 3 2 1 1 1 0 2 2	1 2 1 2 1 1 1 0 0 0 3 3	3 3 2 2 2 2 2 1 0 1 2 1	0 1 1 1 2 0 2 3 3 3 0 2
1 1 3 2 2 2 1 1 3 2 2 3	2 2 1 0 2 3 3 3 3 3 3 0	3 1 1 3 3 0 1 0 2 2 0 0	0 1 2 2 2 1 3 0 3 0 0 1	2 2 2 1 0 0 0 3 1 0 2 2
0 0 0 3 1 1 0 0 0 1 1 0	0 1 2 0 3 3 0 0 3 1 3 3	1 0 1 0 1 1 0 3 2 3 2 1	0 2 0 0 0 0 0 1 0 0 0 3	3 3 2 0 1 0 0 2 0 2 2 2
0 3 0 2 2 2 2 2 3 3 3 3	1 1 0 0 0 0 1 2 0 3 1 0	2 3 3 1 1 0 2 3 0 3 3 1	2 0 0 0 0 2 2 3 1 1 3 3	1 1 3 1 2 3 3 3 3 1 1 1
1 0 3 3 0 2 0 0 0 0 3 2	3 3 3 1 2 2 0 2 2 1 3 0	1 1 1 0 2 2 3 0 2 1 3 2	1 0 0 1 3 1 2 2 0 2 3 3	1 3 2 3 1 3 2 3 3 0 2 2
2 3 0 3 3 3 0 3 0 2 1 3	2 0 0 2 3 2 2 2 2 0 0 2	2 0 2 1 1 1 1 2 0 1 2 1	3 0 1 0 1 2 0 1 1 2 0 3	0 0 2 2 1 0 1 1 1 3 3 0
0 3 1 3 0 0 1 3 3 0 3 3	2 2 1 1 3 0 1 2 2 3 1 3	3 3 2 3 3 1 3 0 0 0 3 0	3 1 2 1 1 2 1 1 3 1 1 0	3 2 3 2 0 0 3 3 1 2 0 1
2 2 0 0 1 2 3 3 0 0 1 1				

Table A.3: $(12, 56, 7)_4$ Code — Code #3

1 2 0 1 3 1 1 3 3 2 0 1	2 0 1 0 3 2 2 0 0 3 1 3	1 3 3 1 1 1 0 0 0 2 1 3	0 0 3 3 3 0 1 2 2 2 1 0	3 1 0 3 0 3 1 1 1 2 1 1
0 3 2 3 0 3 0 3 3 3 1 0	0 2 2 0 0 0 0 2 0 2 1 3	2 0 0 1 1 1 3 3 3 1 3 3	0 2 3 2 2 2 1 3 2 1 1 1	2 1 1 1 2 1 1 3 0 3 2 2
3 0 0 2 2 2 3 3 3 2 2 0	3 2 1 0 1 2 2 1 1 1 3 3	1 2 3 3 2 2 0 1 1 1 2 0	0 2 2 1 1 2 2 3 3 0 0 0	3 1 0 0 2 0 3 0 0 1 0 3
3 0 2 3 1 3 0 2 0 0 3 0	2 1 2 0 2 2 2 2 3 2 0 3	1 1 1 1 3 2 0 2 3 3 3 0	2 3 3 3 3 3 3 0 2 1 1 2	3 1 1 3 2 2 2 3 1 0 2 1
2 2 1 0 1 0 1 0 0 2 2 0	1 0 2 0 0 3 3 3 1 1 3 1	0 0 1 1 2 1 0 1 1 2 0 3	2 2 2 1 1 2 0 0 2 3 1 1	0 1 3 3 0 0 3 2 1 0 0 1
0 2 2 3 3 2 3 0 0 1 3 3	1 1 2 1 3 0 3 0 0 0 1 1	2 3 0 2 1 3 2 1 0 2 2 3	1 0 1 2 0 1 2 1 1 0 1 2	3 3 3 3 3 2 2 3 2 2 3 3
1 1 2 3 1 1 0 1 2 2 3 2	2 2 2 2 3 0 2 3 3 2 3 2	3 3 1 2 0 0 3 2 2 3 1 2	0 0 0 2 1 0 0 0 3 3 0 2	1 0 1 3 0 0 2 0 3 2 2 3
0 3 1 2 2 2 0 2 0 1 3 2	1 0 3 0 3 3 2 2 2 0 2 2	3 3 3 2 0 1 1 0 3 1 3 3	0 2 2 1 1 3 3 2 2 1 2 2	2 3 3 1 1 0 2 2 0 1 0 0
3 2 0 1 1 3 1 0 0 0 0 2	2 1 3 0 1 0 0 3 3 1 2 2	3 2 0 3 1 0 2 2 2 2 2 1	1 1 2 2 0 2 1 2 0 0 0 0	2 2 3 1 3 3 1 1 2 3 3 3
1 1 0 2 1 1 1 3 1 1 1 0	1 1 0 0 0 1 2 2 2 3 3 1	0 2 2 2 3 1 1 1 1 0 3 0	1 1 3 3 3 1 3 3 0 3 0 0	2 0 0 0 0 1 1 1 2 2 2 2
0 0 3 0 1 0 2 0 1 1 1 1	0 3 1 1 0 3 3 1 0 2 3 0	2 3 0 0 0 0 1 3 0 3 0 1	0 3 0 3 1 2 1 3 1 3 3 2	

Table A.4: $(12, 54, 7)_4$ Code — Code #4

1 2 2 2 2 2 3 3 0 0 0 0	3 2 0 1 1 1 2 0 2 0 1 1	1 3 1 3 2 0 3 1 1 1 0 2	1 0 3 1 1 2 3 3 0 2 2 2	1 0 0 2 0 3 2 1 0 0 0 1
0 0 0 2 0 0 3 1 3 1 3 3	3 1 1 3 3 3 0 0 0 1 1 2	1 1 1 2 0 0 2 2 3 3 2 2	1 3 2 2 3 1 3 2 2 3 3 3	1 3 3 3 2 1 3 0 3 2 3 1
0 2 1 2 1 3 3 3 1 2 1 0	1 1 1 1 2 2 1 3 1 3 3 0	0 2 2 3 2 0 1 1 0 0 3 3	3 2 2 0 3 3 2 3 0 2 0 2	2 0 0 1 3 3 2 2 1 3 2 1
1 1 0 3 1 3 3 1 3 2 0 0	1 2 3 0 0 2 2 2 2 2 3 1	1 3 2 0 1 3 3 0 1 1 2 3	1 1 1 3 1 1 2 0 0 0 0 3	2 1 1 0 2 3 1 2 3 2 2 1
1 1 2 3 2 2 1 0 2 1 1 1	3 3 3 1 2 1 2 1 1 0 3 3	2 2 0 1 1 1 1 3 1 2 3 3	0 3 2 3 1 0 1 0 3 2 0 2	3 2 1 3 0 2 1 3 2 0 0 3
2 2 0 3 0 0 3 3 3 3 1 2	0 0 0 0 2 2 2 0 3 2 0 3	1 1 0 0 2 1 2 0 1 1 3 2	2 0 2 3 0 3 3 1 0 2 2 3	3 1 2 2 1 1 2 1 2 2 3 2
0 1 2 1 0 3 3 2 0 3 3 1	0 0 0 3 3 1 1 0 1 1 1 3	2 3 3 2 1 0 0 2 0 0 2 1	2 0 1 2 2 2 2 0 0 0 2 2	3 3 3 0 0 0 0 0 2 3 3 2
2 2 1 2 2 3 0 1 0 1 3 1	3 3 2 2 3 3 3 1 3 1 1 1	2 1 2 3 3 0 0 2 0 3 0 0	3 3 1 3 3 1 2 2 3 1 2 0	0 0 3 1 3 0 0 0 1 2 3 1
1 1 1 1 1 0 3 2 2 2 1 3	0 3 2 0 2 3 0 2 2 1 0 1	3 0 3 0 3 2 3 3 2 1 3 0	0 0 2 3 1 2 3 2 3 0 3 0	0 0 0 0 1 1 1 2 2 2 2 2
3 3 0 2 2 2 2 3 3 3 3 2	0 1 3 1 1 1 1 1 0 0 1 2	2 3 3 0 3 1 3 0 1 0 1 0	1 2 1 1 0 0 0 3 3 1 0 1	2 2 0 1 0 3 0 0 2 2 2 2
3 3 0 0 0 2 3 1 0 1 2 2	3 2 2 2 2 0 0 2 3 2 3 0	3 0 0 0 1 1 3 0 0 3 3 3	2 0 3 2 1 0 1 1 2 1 1 0	3 0 2 0 0 1 0 0 1 2 0 0
0 3 3 3 3 3 2 2 0 0 3 2	0 3 0 0 1 0 2 1 3 3 1 1	2 2 2 0 2 0 2 1 1 1 2 0	2 0 3 1 1 1 0 3 3 3 0 0	

Table A.5: $(12, 59, 7)_4$ Code — Code #5

Appendix B

Results of SCM Decoders

B.1 Distance Two Decoders

Measured is the average number of corrections for the best machines found during 30 evolutions.

Parameters				Exact		Fuzzy	
Population	# Mutations	Type	Distance	Average	Std Dev	Average	Std Dev
11	1	Training	1	403.167	26.1891	508.633	18.3049
			2	316.467	22.6955	470.067	19.3443
		Verification	1	395.967	26.1356	506.333	17.3609
			2	288.033	28.6831	447.233	20.8702
	2	Training	1	415.633	17.6742	520.133	28.0563
			2	323.033	12.9707	476.533	29.3219
		Verification	1	408.3	18.3851	513.867	28.3704
			2	298.533	19.0837	454.867	30.9279
	7	Training	1	414.633	18.8487	515.367	25.3778
			2	332.333	14.0107	474.433	29.2866
		Verification	1	402	20.4872	507.467	26.4975
			2	299.567	12.6455	451.933	25.6447
	12	Training	1	419	18.7929	517	16.8605
			2	330	14.0614	472.333	20.0161
		Verification	1	407.3	19.1548	508.067	15.66
			2	295.3	19.3376	448.233	22.1464
25	1	Training	1	403.467	23.3589	515.867	28.5449
			2	321.5	20.3025	474.3	36.9773
		Verification	1	394.7	25.2247	511.7	30.2975
			2	297.5	22.5813	455.833	33.5611
	2	Training	1	422.467	19.5391	519.4	25.0924
			2	334.067	17.1181	478.367	26.6697
		Verification	1	410.433	19.242	511.933	25.9441
			2	304.767	19.8506	456.267	25.8389
	7	Training	1	417.833	23.9065	523.067	21.5422
			2	329.467	17.9746	478.067	26.92
		Verification	1	409.433	25.3944	518.033	22.9113
			2	305.333	23.0327	462.167	28.8385
	12	Training	1	415.433	16.7305	513.367	27.6711
			2	325.033	11.4876	466.333	26.8654
		Verification	1	403.733	18.9408	503.233	30.9824
			2	297.8	16.2362	447.533	28.0317
51	1	Training	1	409.2	20.8184	511.3	28.3709
			2	322.2	15.6545	464.1	29.3755
		Verification	1	401.667	21.0243	504.233	28.9455
			2	296.6	21.8215	448.7	29.3447
	2	Training	1	410.467	19.0656	511.2	32.8438
			2	327.4	17.6998	468.867	31.4541
		Verification	1	398.333	19.8274	505.3	31.6774
			2	301.033	19.7335	449.833	37.2819
	7	Training	1	417.933	18.0515	516.767	24.5338
			2	332.867	14.4144	475.8	24.626
		Verification	1	408.033	16.0419	509.633	22.0587
			2	303.933	16.4776	454	30.1811
	12	Training	1	424.067	11.6971	520.867	26.844
			2	332.1	14.0721	476.433	27.8267
		Verification	1	410.567	15.0532	511.833	26.7828
			2	303.1	12.7424	456.5	25.8147
101	1	Training	1	410.767	20.0253	515.9	21.2868
			2	322.567	13.302	466.7	24.9788
		Verification	1	395.267	21.6109	505.667	25.8701
			2	293.767	21.1785	450	29.6892
	2	Training	1	416.233	20.4647	512.133	24.5634
			2	326.067	14.1347	466.4	28.9561
		Verification	1	405.667	23.228	504.367	25.143
			2	297.933	17.8479	446.133	29.9939
	7	Training	1	426.333	18.9506	516.267	21.2634
			2	330.467	14.5098	463.8	27.8015
		Verification	1	410.9	21.9173	504.367	25.5903
			2	307.967	15.253	448	28.836
	12	Training	1	419.933	18.1468	509.7	22.0237
			2	332.9	16.2509	466.4	22.1057
		Verification	1	408.333	21.5908	504	22.4638
			2	304.433	20.7143	446.233	24.6712

Table B.1: 6 States – $(12, 55, 7)_4$ – Code #1 – Perfect Score is 660

Parameters				Exact		Fuzzy	
Population	# Mutations	Type	Distance	Average	Std Dev	Average	Std Dev
11	1	Training	1	537.5	28.1446	587.9	28.773
			2	456.467	22.6514	542.067	35.0644
		Verification	1	525.9	30.5968	582.367	29.8334
			2	410.067	23.0516	520	36.8782
	2	Training	1	544.567	25.4256	586.4	27.6637
			2	462.367	21.3888	543.6	33.1201
		Verification	1	531.1	28.1931	581.567	31.511
			2	413.267	21.7128	518.7	37.7671
	7	Training	1	539.433	24.2468	588.167	24.1776
			2	451.7	21.7083	540.567	32.6884
		Verification	1	527.067	27.9037	581.3	25.8379
			2	411.267	22.2027	522.667	32.9402
	12	Training	1	531.167	23.0263	580.067	28.2525
			2	440.667	19.2467	529.633	36.4488
		Verification	1	515.167	27.8717	571.667	32.5665
			2	402.367	21.0672	510.533	38.2962
25	1	Training	1	540.567	29.6039	593.3	30.0989
			2	453.633	26.2842	544	33.721
		Verification	1	524.167	36.3907	583.467	34.4581
			2	409.733	28.8503	526.267	37.4478
	2	Training	1	545.833	22.7658	596.533	28.1275
			2	462.067	23.2393	548.467	29.5013
		Verification	1	531.167	30.2166	589.567	33.4599
			2	415.067	22.8834	526.933	35.2684
	7	Training	1	542.1	20.4642	589.267	23.4637
			2	455.867	16.8681	542	27.1636
		Verification	1	526.1	24.9653	582.867	23.7788
			2	411.367	22.132	523.133	31.3432
	12	Training	1	526.433	25.1255	577.867	28.0624
			2	438	21.3945	525.9	31.7364
		Verification	1	513.4	29.056	571.567	30.2423
			2	397.433	24.332	506.067	35.7924
51	1	Training	1	532.833	29.7508	577.6	32.2004
			2	450.567	28.4662	531.467	38.7456
		Verification	1	517.833	32.6719	570.4	35.4572
			2	400.8	25.9581	508.633	41.7245
	2	Training	1	552.2	15.4795	600.1	18.0657
			2	460.6	16.7838	552.5	22.3202
		Verification	1	535.133	18.7759	592.7	19.6261
			2	416.933	18.5861	534.6	29.0096
	7	Training	1	535.367	25.7059	580.5	24.1343
			2	453.533	23.0138	530.933	33.5764
		Verification	1	524.2	30.1484	574.167	27.1243
			2	412.6	17.9781	511.733	34.0466
	12	Training	1	535.333	19.9332	582.9	26.0535
			2	449.567	15.7714	537	30.8422
		Verification	1	521.767	24.1614	574.4	27.9909
			2	404	15.9525	514.733	33.5969
101	1	Training	1	544.667	20.9043	588.8	21.7769
			2	461.2	20.3239	544.5	26.4624
		Verification	1	528.9	27.7704	580.467	25.5245
			2	414.8	22.2624	524.6	29.6539
	2	Training	1	535.933	29.71	579.767	32.9437
			2	451.233	24.7479	533.2	35.4278
		Verification	1	520.567	33.8533	571.967	34.0886
			2	404.067	29.4594	512.367	42.7273
	7	Training	1	532.933	25.8963	573.233	30.8944
			2	444.967	21.3242	520.7	37.8893
		Verification	1	518.2	28.4586	565.833	33.1539
			2	407.067	24.2671	503.033	38.4882
	12	Training	1	531.667	20.295	575.9	22.8644
			2	449.767	18.9458	531.1	28.8054
		Verification	1	515.767	26.8825	567.267	27.958
			2	403.133	25.0417	505.933	33.4828

Table B.2: 12 States – $(12, 55, 7)_4$ – Code #1 – Perfect Score is 660

Parameters				Exact		Fuzzy	
Population	# Mutations	Type	Distance	Average	Std Dev	Average	Std Dev
11	1	Training	1	557.6	20.4174	596.433	23.8887
			2	471.5	18.2695	531.133	26.9005
		Verification	1	538.1	23.7521	587.133	26.299
			2	415.633	21.503	499.1	32.7492
	2	Training	1	560.9	21.1438	593.767	27.5639
			2	468.433	20.7492	524.433	32.9029
		Verification	1	541.167	27.5845	581.4	33.3969
			2	413.633	22.2749	493.9	37.9449
	7	Training	1	538.867	24.1657	577.4	27.2265
			2	450.567	23.2003	505.333	30.7777
		Verification	1	517.3	27.6906	565.567	30.7321
			2	397.067	27.2143	474.833	35.8821
	12	Training	1	537.433	24.7814	576.133	29.1414
			2	446.633	30.8841	509.8	36.8533
		Verification	1	516.6	31.6539	564.833	36.4267
			2	394.9	34.1552	480.6	43.4032
25	1	Training	1	547.733	24.5412	583.033	26.8399
			2	461.3	24.7416	515.433	29.417
		Verification	1	524.967	29.8946	571.067	30.4324
			2	406.033	30.893	484.033	34.4829
	2	Training	1	553.533	23.0558	583	24.5216
			2	469.767	24.2539	518.4	32.6059
		Verification	1	534.367	31.2746	573.9	31.3999
			2	415.533	27.1455	489	34.3602
	7	Training	1	535.767	23.2048	576.033	26.1184
			2	437.633	24.2309	499.8	31.2403
		Verification	1	509.233	25.1021	559.933	30.8321
			2	390.1	29.0699	472.767	38.8078
	12	Training	1	532.667	26.2722	571.267	31.0938
			2	441.767	29.3301	497.667	39.237
		Verification	1	510.033	33.4195	556.2	38.2536
			2	391.067	30.0103	467.667	40.8372
51	1	Training	1	551.333	20.1089	584.8	22.8781
			2	464.533	15.8956	515.4	26.1146
		Verification	1	527.367	23.3142	571.867	25.9412
			2	411.933	19.2943	488.8	27.1755
	2	Training	1	559.733	17.0434	595.267	20.6163
			2	469.133	16.8947	527.4	27.4887
		Verification	1	536.633	22.4614	584.867	26.0593
			2	413.267	18.9372	499.233	32.6018
	7	Training	1	537.667	24.6441	579.567	23.3115
			2	450.633	24.2565	508.5	32.6388
		Verification	1	514.6	26.6285	565.167	26.3414
			2	397.567	25.6686	480.333	34.7606
	12	Training	1	529.8	29.4576	571.933	33.1277
			2	441.833	29.0054	503.2	39.052
		Verification	1	509.167	33.9351	560.3	41.0585
			2	390.667	32.1562	475.333	43.219
101	1	Training	1	552.6	26.3695	584.2	28.5807
			2	464.9	21.6243	513.133	28.8728
		Verification	1	529.6	32.4628	573.267	33.8434
			2	409	23.5255	486.6	33.2551
	2	Training	1	544.067	22.1965	570.433	25.9012
			2	463.733	19.0044	504.7	25.3855
		Verification	1	518.8	23.5949	553.667	31.1186
			2	403.633	18.576	467.867	29.534
	7	Training	1	537.167	20.4738	573.567	22.4402
			2	454.567	19.9183	507	26.4106
		Verification	1	517.367	25.6911	562.8	25.9448
			2	402.9	21.2836	479.667	34.6801
	12	Training	1	526.533	29.5223	560	34.4343
			2	438.9	29.8356	490.567	36.7336
		Verification	1	503.9	31.6787	549	37.4497
			2	392.533	30.8542	463.467	43.8231

Table B.3: 18 States – $(12, 55, 7)_4$ – Code #1 – Perfect Score is 660

Parameters				Exact		Fuzzy	
Population	# Mutations	Type	Distance	Average	Std Dev	Average	Std Dev
11	1	Training	1	411.967	27.2175	518.533	27.1188
			2	322	21.7383	473.2	30.7576
		Verification	1	409.5	24.3392	520.267	25.6904
			2	301.5	24.5747	457.7	31.8976
	2	Training	1	427.633	32.0457	525.867	20.6577
			2	335.267	31.6118	482	19.4032
		Verification	1	417.933	28.8156	523.767	18.2637
			2	313.333	33.0499	466.567	24.2866
	7	Training	1	430.8	19.3344	523.133	18.1122
			2	339.6	14.6913	478.8	20.8895
		Verification	1	418.733	19.0606	521.667	17.3907
			2	315.867	18.0931	465.367	20.5636
	12	Training	1	433.233	20.5438	526.2	10.0358
			2	337.3	14.2397	479.6	13.5407
		Verification	1	422.533	18.7557	523.4	12.3891
			2	314.1	21.4337	465.9	15.7115
25	1	Training	1	417.967	22.079	527.233	20.9757
			2	331.433	14.4501	485.233	21.3132
		Verification	1	409.333	20.0763	525.4	22.6497
			2	306.533	15.6838	466.733	24.5215
	2	Training	1	431.233	18.7592	531.733	15.9632
			2	338.667	14.8657	481.967	20.3444
		Verification	1	422.867	17.8552	529.3	16.869
			2	318.3	15.013	469.633	20.9852
	7	Training	1	430.633	19.2542	526.767	14.6398
			2	336.8	19.1894	479.467	17.9438
		Verification	1	415.433	23.2552	521.567	14.1828
			2	315.267	18.379	468.1	19.0015
	12	Training	1	425.9	20.652	522.7	22.0159
			2	340.4	16.9331	480.1	22.7874
		Verification	1	415.467	19.9183	521.1	22.3072
			2	315.633	17.3732	466.467	26.6066
51	1	Training	1	421.867	21.8423	524.9	19.6896
			2	330.8	18.1268	478	23.3194
		Verification	1	412.333	20.7702	524	19.6065
			2	313.3	19.6366	467.767	21.4905
	2	Training	1	431.433	16.0381	524.567	16.9476
			2	339.667	16.2658	476.367	17.9338
		Verification	1	419.133	17.5945	522.1	15.3271
			2	316.7	19.6927	466.3	19.4726
	7	Training	1	431.7	20.4251	527.633	18.4606
			2	336.033	15.4373	481.467	22.2024
		Verification	1	419.033	16.841	523.9	22.3443
			2	315.133	20.701	470.967	22.9384
	12	Training	1	437.167	18.0938	524.467	18.8711
			2	336.5	16.387	474.933	21.0696
		Verification	1	428.1	20.2422	521.433	20.3057
			2	321.267	20.7795	464.8	19.1625
101	1	Training	1	423	25.5329	518.267	17.8575
			2	331.667	15.457	473.167	17.9752
		Verification	1	412.4	24.2723	515.833	15.9094
			2	306.667	21.6752	456.4	22.5612
	2	Training	1	429.033	13.074	522.367	19.5263
			2	338.667	14.5349	475.567	23.1899
		Verification	1	419.033	18.0793	519.467	18.9641
			2	314.533	15.3505	461.4	22.5978
	7	Training	1	430.4	11.996	521	15.378
			2	337.867	14.0706	475.233	14.7851
		Verification	1	421.733	19.104	518.133	14.5951
			2	316.5	15.0442	459.533	20.4311
	12	Training	1	428.567	14.1194	521.2	18.438
			2	336.967	14.0209	475.6	20.0802
		Verification	1	418.367	17.6117	518.567	18.5727
			2	315.133	20.8818	459.933	20.8474

Table B.4: 6 States – $(12, 56, 7)_4$ – Code #2 – Perfect Score is 672

Parameters				Exact		Fuzzy	
Population	# Mutations	Type	Distance	Average	Std Dev	Average	Std Dev
11	1	Training	1	543.133	31.3212	584.467	28.1103
			2	455.733	27.5204	537.9	32.009
		Verification	1	529.367	35.6946	579.133	30.9591
			2	425.533	28.874	523.167	36.9212
	2	Training	1	548.7	23.0249	592.4	21.8783
			2	454.733	19.4386	540.967	26.2619
		Verification	1	537.233	25.4107	590.167	24.7777
			2	423.833	25.1013	530.6	32.2828
	7	Training	1	528.7	25.6463	579.367	25.543
			2	441.2	25.4469	528.2	30.9007
		Verification	1	517.6	27.6762	578.167	24.4429
			2	408.167	30.402	511.6	33.2686
	12	Training	1	532.767	22.8785	585.933	27.4364
			2	439.5	19.6306	532.933	24.4384
		Verification	1	523.267	25.5261	581.567	26.8645
			2	411.133	26.3029	521.933	34.1447
25	1	Training	1	537.067	28.4641	579.533	25.8587
			2	450.067	21.1153	525.233	26.9478
		Verification	1	521.067	31.9913	571.833	28.9412
			2	419.467	26.9709	510.067	31.9751
	2	Training	1	549.5	19.8698	591.433	20.6259
			2	459.167	19.4335	542.767	24.5212
		Verification	1	536.533	22.8801	586.033	22.4369
			2	428.6	26.5494	527.967	28.9297
	7	Training	1	530.667	27.561	580.233	25.087
			2	447.267	26.9686	528.433	30.2936
		Verification	1	518.867	27.9442	576.9	26.8654
			2	410.3	33.3168	510.8	36.5828
	12	Training	1	524.933	26.1428	574.2	25.6009
			2	439.267	23.1426	520.567	30.7253
		Verification	1	512.533	28.0181	569.3	29.2388
			2	406.4	29.6887	504.233	39.5489
51	1	Training	1	542.967	26.6011	584.667	23.9184
			2	455.6	21.4711	537.733	29.4372
		Verification	1	530.4	28.5604	582.667	26.142
			2	421	24.6884	520.033	29.7594
	2	Training	1	541.167	23.4375	587.133	22.6224
			2	455.333	15.6917	539.7	21.1923
		Verification	1	527.367	24.3728	583.967	23.3511
			2	420.9	21.8022	520.6	25.2922
	7	Training	1	526.567	27.2304	577.433	32.7105
			2	439.467	25.542	524.667	37.7773
		Verification	1	514	30.8847	571.867	37.2038
			2	407.867	26.2294	509.033	37.7866
	12	Training	1	517.733	25.9282	569.567	29.1674
			2	435.5	18.7428	516.767	29.6167
		Verification	1	503.1	25.9633	565.433	29.5118
			2	402.933	25.4259	500.3	36.5052
101	1	Training	1	539.833	29.0589	580.4	26.6207
			2	454.6	23.8698	529.2	28.5372
		Verification	1	522.867	33.2086	574.3	29.5788
			2	420.8	25.5564	512.867	34.2151
	2	Training	1	544.267	26.3582	585.233	23.2508
			2	455.733	22.4422	536.933	25.6514
		Verification	1	532.867	30.3096	583.467	24.0771
			2	426.1	28.3736	523.1	31.7993
	7	Training	1	532.4	29.5385	573.3	31.8998
			2	454.033	25.1951	527.133	34.5889
		Verification	1	523.333	31.0265	569.933	35.2899
			2	418.833	36.3935	505.3	43.8573
	12	Training	1	515.567	23.6434	561.567	25.5905
			2	435.767	18.207	508.567	27.426
		Verification	1	504.2	24.0422	557.833	26.6886
			2	405.4	20.1299	490.867	30.08

Table B.5: 12 States – $(12, 56, 7)_4$ – Code #2 – Perfect Score is 672

Parameters				Exact		Fuzzy	
Population	# Mutations	Type	Distance	Average	Std Dev	Average	Std Dev
11	1	Training	1	553.333	21.933	594.333	29.0556
			2	459	20.9614	523.533	31.2374
		Verification	1	536.033	25.3166	588.467	30.3448
			2	405.8	26.1658	491.967	40.4078
	2	Training	1	561.1	26.4488	597.133	27.2975
			2	465.8	35.3226	523.4	37.8359
		Verification	1	545.567	34.5031	592.3	33.3127
			2	418	36.437	502.4	44.5108
	7	Training	1	535.933	27.6467	574.667	33.7418
			2	444.7	26.3349	507.633	39.5234
		Verification	1	519.767	31.0113	571.433	33.7421
			2	401.8	32.6511	484.767	46.8833
	12	Training	1	530.867	25.7089	580.033	27.3779
			2	431.6	28.7289	502.133	28.9014
		Verification	1	517.2	26.4984	575.633	27.4383
			2	386.933	33.2264	477.233	32.7168
25	1	Training	1	555.1	25.136	587.1	27.0139
			2	464.933	21.5662	516.6	34.4059
		Verification	1	538	27.4327	580.667	28.2065
			2	419.967	29.4366	495.833	36.5354
	2	Training	1	559.5	26.4455	595.733	25.134
			2	468.933	22.8125	525.133	27.2482
		Verification	1	544.3	33.2412	589.5	29.2937
			2	424.067	29.429	501.567	34.2322
	7	Training	1	536.7	27.0824	577.333	26.4762
			2	441.433	28.5388	504.067	30.9916
		Verification	1	517.633	25.5093	570	27.3357
			2	399.8	33.6405	482.833	37.8437
	12	Training	1	526.933	31.1503	567.633	41.0101
			2	430.933	26.078	492.333	46.2432
		Verification	1	511.667	34.8893	564.133	42.3171
			2	385.367	35.5833	468.933	54.7867
51	1	Training	1	539.233	25.5851	568.733	29.4395
			2	447.733	24.5805	497.7	29.2765
		Verification	1	519.467	30.5216	559.567	31.1821
			2	403.633	25.817	471.933	36.4038
	2	Training	1	548.3	25.0009	586.1	23.9386
			2	462.933	24.2187	519.8	23.6489
		Verification	1	531.5	26.422	578.633	27.1795
			2	417.333	27.0317	494.533	28.1789
	7	Training	1	540.267	20.5476	577.033	24.5532
			2	451.967	17.7443	504.733	26.0436
		Verification	1	519.733	23.6526	567.833	28.3355
			2	408.633	24.7212	482.467	31.777
	12	Training	1	528.367	25.7809	574.4	27.1618
			2	440.3	25.252	502.333	29.8194
		Verification	1	513.6	28.5483	568.767	30.465
			2	393.833	26.8535	477.467	30.9758
101	1	Training	1	553.167	25.163	583.2	26.0323
			2	468.367	21.6149	520.633	25.3139
		Verification	1	536.1	26.3927	576.033	29.4647
			2	424.2	25.3287	497.633	34.0947
	2	Training	1	533.267	29.5739	562.5	30.3187
			2	451.067	30.6627	498.6	29.5374
		Verification	1	516.133	32.7011	554.567	32.2893
			2	404.333	34.913	471.867	35.0799
	7	Training	1	531.867	28.54	563.833	29.863
			2	450.9	23.9948	500.133	29.1426
		Verification	1	513.167	33.9097	556.633	34.8133
			2	404.133	29.8534	470.4	36.1983
	12	Training	1	526.567	23.0945	563.967	26.6723
			2	442.233	17.3914	500.3	30.0725
		Verification	1	511.7	26.4368	558.067	30.5252
			2	402.633	20.7189	475.967	31.1132

Table B.6: 18 States – $(12, 56, 7)_4$ – Code#2 – Perfect Score is 672

Parameters				Exact		Fuzzy	
Population	# Mutations	Type	Distance	Average	Std Dev	Average	Std Dev
11	1	Training	1	418.167	17.6168	514.467	26.0275
			2	333.567	16.1836	470.267	25.4625
		Verification	1	400.9	17.779	504.633	28.17
			2	311.833	22.4101	463.333	30.2431
	2	Training	1	420.933	18.2831	525.233	22.8755
			2	345.333	17.2013	487.633	25.8423
		Verification	1	412.933	14.3261	520.9	22.8403
			2	316.1	18.0867	478.367	27.6761
	7	Training	1	418.633	20.2918	521.2	25.4564
			2	339.867	16.9334	481.567	27.353
		Verification	1	405.133	23.0886	514.9	24.2051
			2	316.533	21.2485	475.733	28.8096
	12	Training	1	420.667	19.3397	519.133	22.6818
			2	343.667	13.7448	486.5	24.4861
		Verification	1	411.633	16.8942	514.2	25.6117
			2	313.633	21.5174	473.033	27.7681
25	1	Training	1	415.933	21.2667	524.4	25.7917
			2	332.233	24.3915	486	30.2962
		Verification	1	405	23.7124	518.233	24.8147
			2	304.233	26.0632	474.3	28.5067
	2	Training	1	424.967	16.7053	521.1	19.5031
			2	344.467	15.8086	482.7	24.5766
		Verification	1	410	15.4875	511.767	23.566
			2	317.533	16.4542	471.2	26.6179
	7	Training	1	415.9	17.0341	519.467	24.4664
			2	342.733	15.9955	484.067	27.1737
		Verification	1	403.9	18.6258	513.733	27.5467
			2	313.033	16.6433	468.667	31.4876
	12	Training	1	421.2	19.1858	525.9	23.5025
			2	345.467	18.3843	485.633	29.284
		Verification	1	406.967	18.5834	518.567	25.8613
			2	312.167	24.3113	472.667	24.5727
51	1	Training	1	422.267	18.2906	519.233	16.7078
			2	339.1	19.2432	476.867	16.094
		Verification	1	410.6	16.5396	513.633	15.4283
			2	318.467	20.1164	467.267	19.9619
	2	Training	1	418.267	21.2521	516.367	28.4962
			2	341.9	20.5801	482.667	31.4471
		Verification	1	408.767	26.701	515.233	29.5631
			2	316.4	16.988	470.633	32.7071
	7	Training	1	425.167	15.1295	528.1	25.0645
			2	350.033	15.3297	491.167	29.6358
		Verification	1	410.867	16.7409	520.733	25.0763
			2	317.567	21.5658	483.2	29.0189
	12	Training	1	428.567	17.3099	531.2	19.6353
			2	338.867	25.6955	483.5	30.6974
		Verification	1	409.667	17.7829	518.667	21.7467
			2	318.8	23.8218	481.4	27.4699
101	1	Training	1	416.367	16.3844	517.667	23.0117
			2	338.167	16.8033	480.6	21.0395
		Verification	1	403.733	21.5677	511.6	21.1751
			2	314.033	21.0131	471.833	26.9586
	2	Training	1	426.767	18.4591	525.867	19.4045
			2	349.067	16.7763	486.5	26.7527
		Verification	1	411.633	17.2576	517.967	20.8335
			2	324.833	21.4204	483.433	25.2951
	7	Training	1	422.467	13.0641	523.9	21.9268
			2	347.833	15.6516	487.9	25.1387
		Verification	1	411.967	18.5667	521.1	23.348
			2	316.867	17.8996	476.067	27.1597
	12	Training	1	425.6	14.7545	523.3	22.1019
			2	347.167	17.3545	489.867	25.9638
		Verification	1	411.867	14.936	516.833	23.628
			2	316.667	18.7953	475.7	29.1431

Table B.7: 6 States – $(12, 56, 7)_4$ – Code #3 – Perfect Score is 672

Parameters				Exact		Fuzzy	
Population	# Mutations	Type	Distance	Average	Std Dev	Average	Std Dev
11	1	Training	1	552.933	19.9481	604.3	28.1843
			2	479.467	19.3154	570.733	31.6706
		Verification	1	535.867	24.8634	601.2	30.9242
			2	424.1	23.6473	552.667	36.5026
	2	Training	1	550.3	27.4856	599.833	29.6335
			2	470.933	21.5037	560.5	33.2698
		Verification	1	534.133	30.1247	596.1	29.7267
			2	421.133	22.3479	543	35.7559
	7	Training	1	542.9	22.884	597.533	27.7498
			2	464.367	22.4399	556.5	31.8842
		Verification	1	525.433	29.178	592.533	30.6917
			2	418.467	25.3945	539.567	33.8166
	12	Training	1	537.233	25.6981	593.767	26.8594
			2	457.867	22.3433	549.067	34.778
		Verification	1	520.5	26.0394	587.533	30.2184
			2	414.3	24.7388	535.3	37.2347
25	1	Training	1	552.433	24.83	602	26.0927
			2	475.133	18.621	565.2	33.3977
		Verification	1	535.067	30.5512	599.2	31.3483
			2	427.033	25.1252	547.833	34.1367
	2	Training	1	547.833	25.8031	600.467	31.223
			2	470.8	27.3526	558.2	36.3919
		Verification	1	530.867	31.2969	594.333	35.5793
			2	425.233	27.2393	543.6	37.3138
	7	Training	1	536.5	25.0706	594.467	30.9268
			2	455.6	27.7918	545.567	38.7215
		Verification	1	517.867	29.8106	585.5	36.0342
			2	415.167	21.8697	534.833	37.0108
	12	Training	1	529.733	25.4273	580.267	31.4477
			2	456.833	20.194	539.067	35.1714
		Verification	1	510.933	28.6536	574.233	32.7042
			2	410.8	21.5573	522.4	36.2877
51	1	Training	1	543.067	29.61	582.3	34.4565
			2	470.6	27.315	542.9	37.4593
		Verification	1	524.967	33.4112	575.1	36.8813
			2	427.567	23.0033	528.4	37.2453
	2	Training	1	546.1	27.5735	594.8	27.2882
			2	469.267	24.3791	554.067	34.1588
		Verification	1	526.733	32.6644	590.233	32.4215
			2	418.133	26.3906	536.167	38.3155
	7	Training	1	524.967	23.959	576.367	30.2945
			2	457.067	20.6513	535.267	31.879
		Verification	1	508.7	28.7284	571.833	33.4665
			2	409.333	22.7268	515	40.0818
	12	Training	1	536.8	22.6661	593.067	22.7171
			2	459.2	22.6158	550.467	30.4877
		Verification	1	519.667	26.5425	588.567	26.7848
			2	416.033	23.018	535.9	30.9475
101	1	Training	1	539.533	30.3642	583	31.2178
			2	466.833	26.4107	545.6	35.8585
		Verification	1	520	34.0719	577	36.0383
			2	419.6	27.7571	528.733	39.664
	2	Training	1	542.367	23.7465	591.033	23.3393
			2	468.767	21.2565	551.233	26.5845
		Verification	1	524.867	24.3958	584.167	25.9629
			2	416.9	20.9932	529.067	33.9472
	7	Training	1	530.333	24.842	576.5	27.7982
			2	457.333	23.6167	532.867	36.9583
		Verification	1	510.3	26.5942	568.867	30.5577
			2	410.733	20.1425	514.167	34.9513
	12	Training	1	520.4	27.7409	571.8	33.6948
			2	447.8	24.3061	526.533	39.4756
		Verification	1	502.067	31.3775	565.467	37.4789
			2	405.833	27.9422	510.433	38.0822

Table B.8: 12 States – $(12, 56, 7)_4$ – Code #3 – Perfect Score is 672

Parameters				Exact		Fuzzy	
Population	# Mutations	Type	Distance	Average	Std Dev	Average	Std Dev
11	1	Training	1	572.333	22.5409	612.6	24.8063
			2	477.733	27.2763	548.967	29.592
		Verification	1	553.133	24.2384	607	26.2087
			2	422.533	28.8262	518	32.2512
	2	Training	1	564.3	29.4842	604.633	27.8264
			2	476.7	33.2173	543.467	36.1508
		Verification	1	543.667	33.1385	597.467	32.1952
			2	422.533	35.1399	519.433	37.396
	7	Training	1	555.067	19.3176	600	19.7694
			2	465.3	23.7924	535.967	25.8049
		Verification	1	533.8	23.5671	593.233	24.7354
			2	419.733	23.6599	515.633	29.2628
	12	Training	1	533.467	31.1368	582.967	33.6631
			2	446.7	29.6592	518.8	33.7623
		Verification	1	510.433	36.0705	573	37.97
			2	391.633	31.2471	492.167	37.8765
25	1	Training	1	555.933	23.6802	595.033	30.8215
			2	472.867	22.0622	533.5	32.3832
		Verification	1	535.367	26.5739	586	34.6101
			2	415.3	26.9356	506.467	39.4075
	2	Training	1	561.333	24.6105	603.133	22.7971
			2	478.167	29.3705	544	31.8174
		Verification	1	541.833	30.2793	595.233	26.1965
			2	428	26.5278	524.3	33.4315
	7	Training	1	547.967	28.6471	585.567	33.9824
			2	461.8	25.6117	523.5	38.3107
		Verification	1	526.4	29.9604	577.8	36.1791
			2	413.267	22.5464	504.967	37.985
	12	Training	1	529.5	30.8609	570.633	35.2855
			2	449.767	29.1248	507.2	37.5614
		Verification	1	509.433	37.6432	560.533	38.0759
			2	395.133	29.7388	477.2	44.2223
51	1	Training	1	553.433	26.5131	591.067	25.8403
			2	475.533	26.0619	531.133	28.5231
		Verification	1	533.033	26.5284	583.733	26.4665
			2	422.633	29.5255	503.767	31.028
	2	Training	1	559.167	27.2562	599.633	27.6099
			2	482.067	25.8576	545.033	33.8399
		Verification	1	536.2	29.7129	591.2	31.3846
			2	422.333	26.7882	520.733	36.5569
	7	Training	1	538.133	33.5114	578.867	39.2136
			2	455.933	30.1433	513.5	42.7267
		Verification	1	515.933	41.3204	569.467	45.6665
			2	402.733	28.7785	488.133	46.6408
	12	Training	1	528.5	30.1545	571.4	27.2771
			2	449.767	25.9438	507.2	26.0178
		Verification	1	506.633	30.6093	560.767	27.3568
			2	400.167	30.0311	483.3	27.6482
101	1	Training	1	539.433	28.335	569.9	34.0956
			2	470.033	28.3555	520.4	40.8509
		Verification	1	518.633	34.0755	563.7	36.8484
			2	414.367	30.185	490.6	44.9495
	2	Training	1	552.9	32.4785	588.267	32.945
			2	479.667	28.2969	533.9	32.9758
		Verification	1	533.667	37.3228	582.5	34.2493
			2	427.067	29.8397	510.467	38.0315
	7	Training	1	531.1	29.5382	566.9	37.598
			2	463.333	22.6538	517.367	36.2658
		Verification	1	513.833	30.496	561.5	37.8734
			2	408.767	26.6868	490.3	43.9146
	12	Training	1	537.9	23.1953	575.6	32.9886
			2	459.267	22.6593	515.833	34.2768
		Verification	1	516.3	27.1092	565	35.3329
			2	410.467	24.291	492	41.0844

Table B.9: 18 States – $(12, 56, 7)_4$ – Code #3 – Perfect Score is 672

Parameters				Exact		Fuzzy	
Population	# Mutations	Type	Distance	Average	Std Dev	Average	Std Dev
11	1	Training	1	397.367	31.4637	501.4	30.4094
			2	303.5	20.5473	453.267	30.5534
		Verification	1	388.5	32.0127	499.667	30.6001
			2	294.667	23.1447	451.367	28.1578
	2	Training	1	404.2	19.4376	502.733	19.6993
			2	309.633	16.7527	457.767	22.5949
		Verification	1	397.333	19.1695	499.4	21.4502
			2	297.767	16.7161	457.133	21.1395
	7	Training	1	404.167	26.5214	507.233	20.015
			2	316.033	20.9901	461.1	26.3718
		Verification	1	397.7	26.6279	503.733	25.3947
			2	304.733	16.2924	460.633	26.2803
	12	Training	1	402.433	21.561	499.7	24.2489
			2	315.333	15.6323	455.667	27.4607
		Verification	1	402.267	23.9827	501.1	25.8595
			2	305.367	17.4899	454.667	27.7977
25	1	Training	1	397.4	28.5109	498.333	24.2136
			2	305.1	22.5547	457.867	28.4359
		Verification	1	398.1	29.7082	502.233	25.6349
			2	300	24.4089	456.733	25.7989
	2	Training	1	403.333	28.0016	503.133	26.4741
			2	313.633	20.6071	454.267	29.9539
		Verification	1	395.167	32.0819	499.667	29.433
			2	304.667	19.6141	458.533	31.3641
	7	Training	1	402.267	19.8163	492.5	19.8525
			2	316.533	13.2059	451.1	16.9875
		Verification	1	398.9	17.8119	492.067	20.0756
			2	308.7	16.191	447.867	20.485
	12	Training	1	409.367	21.4869	509.767	18.3429
			2	320.1	14.2837	467.367	22.8103
		Verification	1	406.8	19.2253	511.867	17.9899
			2	307.867	17.1036	463.633	20.4981
51	1	Training	1	405.4	25.5527	495.767	30.0295
			2	312.7	15.996	451.633	32.0231
		Verification	1	401.1	25.8221	495.667	31.6242
			2	300.4	18.2693	448.8	36.0281
	2	Training	1	403.6	27.3428	504.7	32.3506
			2	309.733	18.1582	455.333	33.0531
		Verification	1	394.567	31.8322	500.567	34.3518
			2	294.733	20.4382	455.4	38.0259
	7	Training	1	410.333	21.191	504.867	22.8061
			2	317.1	16.6637	457.267	25.2422
		Verification	1	401.433	22.2132	503.5	22.7304
			2	311.333	20.6219	458.367	26.2816
	12	Training	1	410.667	24.47	505.167	26.0438
			2	312.767	14.576	455.433	31.0236
		Verification	1	403.467	24.1243	503.367	28.9857
			2	305.667	13.8273	456.533	31.9997
101	1	Training	1	405.633	25.6105	492.567	24.2639
			2	305.9	14.5634	444.4	24.8591
		Verification	1	405	23.3947	492.667	24.8004
			2	302.733	20.7812	445.167	29.0352
	2	Training	1	407.533	21.3052	503.5	23.629
			2	307.467	17.0996	456.933	30.4324
		Verification	1	399.833	26.757	500.633	29.4753
			2	298.9	21.4353	455.233	31.5438
	7	Training	1	412.533	19.3991	501.633	25.011
			2	319.1	13.6011	453.4	27.3478
		Verification	1	410.733	20.1801	502.933	25.0516
			2	311.267	11.4649	455.567	28.5641
	12	Training	1	410.567	19.6411	500.433	21.3649
			2	311.633	15.1691	455.1	23.7332
		Verification	1	404.767	20.7525	498.133	23.184
			2	302.167	18.0804	451.867	25.246

Table B.10: 6 States – $(12, 54, 7)_4$ – Code #4 – Perfect Score is 648

Parameters				Exact		Fuzzy	
Population	# Mutations	Type	Distance	Average	Std Dev	Average	Std Dev
11	1	Training	1	515.4	24.3106	563.9	25.7191
			2	432.9	22.5944	518.7	29.1809
		Verification	1	504.6	29.3065	561.433	29.2394
			2	406.533	25.0541	504.067	31.1481
	2	Training	1	525.033	30.8125	571.433	28.4904
			2	440.967	32.5698	527.633	37.8932
		Verification	1	518.733	37.0321	569.733	31.906
			2	415	33.0183	514.967	36.678
	7	Training	1	515.933	24.3508	564	21.7842
			2	426.633	22.466	514.333	32.259
		Verification	1	501.433	32.5372	558.867	29.2088
			2	403.567	24.9091	502.9	31.7233
	12	Training	1	510.4	26.444	562.4	26.6272
			2	423.033	25.7099	509.9	37.7088
		Verification	1	497.333	31.6809	555.167	32.673
			2	396.167	28.5731	492.7	39.2623
25	1	Training	1	527.633	27.9254	574.867	21.002
			2	442.3	27.7142	526.933	31.6902
		Verification	1	515.3	30.9484	571.567	23.7104
			2	412.433	29.3982	513.633	30.2501
	2	Training	1	520.867	28.0636	566.3	26.9407
			2	437.633	26.7923	520.733	34.9462
		Verification	1	506	34.6957	558.533	30.4877
			2	409.533	29.755	509.233	38.5475
	7	Training	1	520.267	22.0531	570.867	23.2598
			2	435.333	17.5014	524.433	29.2371
		Verification	1	507.233	26.9926	563.467	27.6664
			2	408.8	21.8038	507.867	34.0372
	12	Training	1	511.433	27.3593	563	28.0983
			2	433.567	23.175	517.333	33.1957
		Verification	1	501.6	30.9612	559.067	29.054
			2	407.5	24.8606	505.3	35.449
51	1	Training	1	523.633	21.3549	568.8	18.5089
			2	441.667	20.5247	521.1	29.635
		Verification	1	513.3	28.3234	564.9	24.3485
			2	412.5	24.9285	508.333	30.7911
	2	Training	1	517.1	32.1273	563.9	32.594
			2	440.333	28.3395	516.833	40.8074
		Verification	1	503.733	37.9963	557.9	38.0121
			2	409.267	32.6992	503.9	43.2757
	7	Training	1	517.833	19.759	566.133	25.908
			2	432.367	17.2576	518.433	31.2186
		Verification	1	502.7	25.2725	559.8	29.9694
			2	405.833	17.1426	505.767	34.3488
	12	Training	1	517.067	22.6745	567.233	25.1337
			2	430.667	17.8738	520.133	29.1189
		Verification	1	508.433	24.741	566.233	26.0724
			2	404.233	19.6288	507.8	28.0755
101	1	Training	1	521.433	24.7103	571.067	24.1474
			2	439.9	21.586	522.567	28.2607
		Verification	1	509.433	31.1699	565	26.1982
			2	412.067	27.0197	511.2	28.4586
	2	Training	1	527.333	26.4879	570.3	24.4232
			2	441.267	21.2667	525.967	29.8369
		Verification	1	516.9	29.6059	568.367	27.5312
			2	415	22.7838	515.067	30.3246
	7	Training	1	513.167	29.3811	557.233	30.2218
			2	430.033	24.8991	507.833	40.548
		Verification	1	501.8	34.72	549.967	35.8709
			2	408.167	26.2811	497.267	41.3754
	12	Training	1	509.6	24.0568	556.233	28.8357
			2	428.4	19.406	509.833	32.6202
		Verification	1	499.667	24.8933	553.167	28.5996
			2	400.933	21.6316	494.3	34.8802

Table B.11: 12 States – $(12, 54, 7)_4$ – Code #4 – Perfect Score is 648

Parameters				Exact		Fuzzy	
Population	# Mutations	Type	Distance	Average	Std Dev	Average	Std Dev
11	1	Training	1	547.367	19.4962	590.233	21.9084
			2	450.967	24.7058	518.167	25.1068
		Verification	1	528.9	24.2691	582.667	24.9653
			2	416.133	27.7746	506.2	29.1883
	2	Training	1	536.267	18.0515	576.3	20.5446
			2	447.633	24.1525	506.033	28.5494
		Verification	1	521.767	25.2227	571.167	25.1219
			2	413	26.3727	490.933	34.4343
	7	Training	1	520.633	22.3151	556.933	30.4958
			2	433.9	23.1582	486.333	35.6248
		Verification	1	497	25.7535	545.033	36.0081
			2	396.367	27.2631	466.2	43.9227
	12	Training	1	513.8	27.2263	559	29.7275
			2	422.4	31.2571	490.1	36.1676
		Verification	1	497.6	36.4631	551.467	36.5379
			2	385.267	34.1881	474.6	37.7885
25	1	Training	1	534.933	25.064	571.867	26.999
			2	445.167	29.5683	502.5	35.2956
		Verification	1	516.7	28.1267	564.3	33.7906
			2	407.433	29.37	485.867	37.7101
	2	Training	1	537.9	28.5975	574.033	31.6723
			2	443.833	28.6442	501.233	43.454
		Verification	1	516.433	33.2915	567.167	35.8484
			2	407.133	31.298	488.233	46.3909
	7	Training	1	528.167	27.8618	560.933	31.2498
			2	437.833	32.4623	491.533	37.8178
		Verification	1	513.267	34.5113	555.167	35.5829
			2	405.933	33.8271	476.167	41.2612
	12	Training	1	514.867	23.6712	559.933	28.0786
			2	428.233	26.3984	490.867	30.5611
		Verification	1	498.667	27.2641	552.433	30.9445
			2	389.4	27.772	472.567	37.016
51	1	Training	1	531.5	21.9352	567.7	25.6866
			2	444.433	25.4717	497.5	26.5587
		Verification	1	511	26.1283	559	28.233
			2	403.8	26.7032	481.4	33.1377
	2	Training	1	538.067	28.4313	575	26.9789
			2	445	31.6086	505.233	34.1717
		Verification	1	514.267	32.5597	565.8	33.4843
			2	407.167	31.1383	489.5	38.0922
	7	Training	1	522.833	22.3315	559.433	23.906
			2	433.2	21.8181	490.3	36.2569
		Verification	1	502.167	27.0084	548.6	29.2747
			2	396.7	28.732	471.633	38.1679
	12	Training	1	509.767	27.1327	556	31.7653
			2	422.9	30.5618	485.867	38.791
		Verification	1	489.5	35.9211	547.567	37.6762
			2	386.933	33.3166	471.3	40.2314
101	1	Training	1	537.967	25.6374	571.967	30.3991
			2	455.1	22.573	508.167	34.9966
		Verification	1	519.8	30.6441	562.833	35.3847
			2	419	27.9223	492	35.4343
	2	Training	1	524.167	30.1971	554	31.9828
			2	442.3	30.4553	485.767	39.7914
		Verification	1	507.1	36.2143	546.867	36.8433
			2	406.1	34.9939	467.8	44.0645
	7	Training	1	509.533	23.19	541.033	26.5037
			2	426.033	20.1725	473.233	26.7217
		Verification	1	491.433	23.1452	532.867	30.0112
			2	392.567	21.2598	451.5	28.6859
	12	Training	1	507.233	22.8393	545.8	31.4976
			2	425.533	20.1302	480.1	34.4977
		Verification	1	493.467	25.808	539.1	34.7219
			2	390.433	23.9679	465.4	37.1368

Table B.12: 18 States – $(12, 54, 7)_4$ – Code #4 – Perfect Score is 648

Parameters				Exact		Fuzzy	
Population	# Mutations	Type	Distance	Average	Std Dev	Average	Std Dev
11	1	Training	1	426.167	23.7938	537.4	23.8394
			2	331.933	24.4878	487.567	24.3391
		Verification	1	404.933	25.0309	520.533	22.7925
			2	327.3	23.8864	486.633	25.5268
	2	Training	1	438.867	24.5494	550.367	26.9066
			2	343.967	19.1284	505.867	25.6632
		Verification	1	420.2	24.5531	536.033	24.4434
			2	330.167	20.3488	496.833	26.0425
	7	Training	1	451.133	27.8255	551.067	22.4145
			2	351.267	12.8544	500.8	24.0694
		Verification	1	427.767	26.2385	532	25.1025
			2	345.733	21.3557	500.2	25.5186
	12	Training	1	439.067	23.6482	544.133	23.6114
			2	342.667	19.4959	499.3	27.7142
		Verification	1	418.867	23.3884	528.5	26.5223
			2	334.633	24.1825	497.333	24.8795
25	1	Training	1	426.6	31.0568	548.767	20.6643
			2	333.667	25.2741	502.467	29.8487
		Verification	1	407.3	32.8981	535.3	27.2082
			2	323.1	30.3524	499.467	30.3573
	2	Training	1	443.433	20.1557	546.933	21.9481
			2	345.733	18.0878	501.933	24.7734
		Verification	1	422.667	18.9251	529.9	27.3437
			2	332.7	23.4096	493.567	29.8175
	7	Training	1	439.067	24.338	551.3	20.0106
			2	343.733	14.3741	507.2	20.2508
		Verification	1	417.933	21.7714	536.133	22.5492
			2	330.733	21.2991	500.733	25.2927
	12	Training	1	450.733	19.3817	553.8	16.4828
			2	348.8	18.2519	506.533	20.6193
		Verification	1	427.9	22.5516	535.767	21.6264
			2	340.1	20.1586	502.767	20.9477
51	1	Training	1	434.1	24.0507	543.133	25.2556
			2	338.067	19.0425	496.367	30.3866
		Verification	1	414.2	22.4075	526.2	29.3556
			2	326.6	20.8353	492.933	25.6501
	2	Training	1	435.367	27.8512	541.767	27.4938
			2	345.033	18.7423	493.567	33.2562
		Verification	1	411.9	31.1596	525.533	35.1438
			2	331.8	26.3536	489.5	32.679
	7	Training	1	440.933	22.2787	546.967	20.8517
			2	351.033	14.8567	502.333	20.0333
		Verification	1	418.733	21.0663	529.967	22.5793
			2	332.133	18.7501	498.2	23.9747
	12	Training	1	445	20.8029	547.333	15.7553
			2	345.8	16.7567	503.567	17.0105
		Verification	1	425.567	24.0283	532	20.9975
			2	334.767	22.3092	500.1	17.0422
101	1	Training	1	430	22.7944	539.1	22.4612
			2	338.867	11.5929	493.1	23.0282
		Verification	1	412.9	24.2876	525.367	22.6221
			2	332.2	21.6419	494.4	25.0965
	2	Training	1	440.967	25.6454	545.767	18.829
			2	348.6	18.4252	500.167	20.2707
		Verification	1	420.3	26.8664	532.2	22.4075
			2	337.533	17.5081	496.733	23.4858
	7	Training	1	442.367	23.0479	548.5	18.6598
			2	348.133	14.5383	503.667	24.9473
		Verification	1	422.567	24.9201	532.6	24.295
			2	336.5	18.9314	501.433	26.418
	12	Training	1	438.267	17.9941	542.4	15.6218
			2	346.933	14.1712	495.3	21.1222
		Verification	1	416.467	22.4726	523.867	21.9525
			2	334.467	19.062	490.767	22.2132

Table B.13: 6 States – $(12, 59, 7)_4$ – Code #5 – Perfect Score is 708

Parameters				Exact		Fuzzy	
Population	# Mutations	Type	Distance	Average	Std Dev	Average	Std Dev
11	1	Training	1	569.933	28.4919	621.3	24.9816
			2	478.767	22.2054	573.267	30.7469
		Verification	1	543.933	30.3178	610.033	28.2018
			2	444.5	29.0561	562.3	35.6149
	2	Training	1	571.033	27.0383	621.167	25.5803
			2	483.267	21.004	575.1	28.8556
		Verification	1	547.367	29.5115	611.633	26.9386
			2	450.667	27.1932	563.233	31.9554
	7	Training	1	558.9	22.7707	616.167	29.9541
			2	471.3	22.4701	569.7	34.3754
		Verification	1	535.033	27.1655	605.467	35.3434
			2	436.333	25.4143	554.833	37.2513
	12	Training	1	547.333	32.7607	608.167	30.9751
			2	461.867	29.0478	558.667	36.0549
		Verification	1	526	34.9413	593.533	33.9388
			2	430.8	26.1658	544.267	38.255
25	1	Training	1	564.967	25.2429	621	27.6879
			2	481.733	23.8616	576.267	30.8846
		Verification	1	539.633	30.2124	609.533	33.5803
			2	439.167	27.5294	557.467	40.8519
	2	Training	1	568.467	28.0538	621.167	29.1537
			2	481.4	24.1184	575.4	35.3188
		Verification	1	545.6	34.6117	609.733	32.602
			2	447.167	32.6191	560.767	39.0708
	7	Training	1	554.767	29.7932	614.367	32.4446
			2	465.5	21.133	561.733	34.6817
		Verification	1	529.233	34.8541	600.733	39.3945
			2	433.267	31.4061	548.033	34.929
	12	Training	1	553.733	29.3692	610.367	28.2714
			2	465.167	23.6674	564.067	29.8709
		Verification	1	529.9	27.998	599.533	28.9908
			2	434.4	29.9432	551.333	36.8457
51	1	Training	1	563.1	21.9692	615.767	28.2522
			2	478.9	19.0396	569.267	32.1279
		Verification	1	540.367	26.6904	607.1	32.9465
			2	441.967	25.6737	553.067	38.0117
	2	Training	1	574.133	17.0187	630.867	23.4311
			2	483.1	20.1055	585.467	26.9287
		Verification	1	549.733	20.6447	621.767	25.6618
			2	451.067	24.9619	574.633	28.8235
	7	Training	1	560.533	29.5667	611.233	31.7736
			2	474.333	22.4336	563.333	35.8092
		Verification	1	539.133	33.8381	598.767	36.0896
			2	441.967	28.6952	549.3	42.7681
	12	Training	1	547.5	30.8699	603.2	35.0786
			2	466.8	21.1014	554.733	37.89
		Verification	1	523.233	38.7847	589.1	42.987
			2	427.3	25.0271	536.867	44.8851
101	1	Training	1	561.867	18.4704	611.467	19.5743
			2	480.033	19.2542	565.3	21.3947
		Verification	1	539.1	22.9247	602.133	22.3726
			2	442.333	24.0622	550.5	25.3782
	2	Training	1	562.333	26.4931	610.733	31.5332
			2	476.7	18.5159	566.7	32.0389
		Verification	1	539.067	28.9505	599.633	33.8501
			2	442.1	26.2118	551.5	39.9955
	7	Training	1	552.633	26.3262	604.933	26.8134
			2	468.867	23.5968	556	33.4798
		Verification	1	529.167	33.353	594.467	34.8294
			2	438.533	25.6216	543	35.4868
	12	Training	1	542.3	25.9418	592.667	28.3966
			2	458.433	24.7131	542.733	32.6475
		Verification	1	519.667	29.4622	580.833	30.7852
			2	427.167	28.3355	525.5	36.4122

Table B.14: 12 States – $(12, 59, 7)_4$ – Code #5 – Perfect Score is 708

Parameters				Exact		Fuzzy	
Population	# Mutations	Type	Distance	Average	Std Dev	Average	Std Dev
11	1	Training	1	575.5	27.9837	617.8	30.6081
			2	481.167	26.4641	543.133	37.1667
		Verification	1	545.733	29.0421	602.567	34.1888
			2	427.467	27.0717	518.267	39.7873
	2	Training	1	583.267	30.0218	620.633	35.2484
			2	483.633	30.161	547.967	39.7965
		Verification	1	554.667	36.8514	608.7	41.7514
			2	440.567	31.2616	530.2	44.0692
	7	Training	1	570.667	27.6858	614.333	31.6046
			2	478.1	21.5668	547.867	31.4804
		Verification	1	541.8	27.3198	600.433	34.945
			2	436.267	25.0846	527.433	35.8403
	12	Training	1	560.267	30.7918	605.167	34.5464
			2	467.333	33.1708	536.033	41.6699
		Verification	1	533.467	37.9271	589.533	39.6369
			2	420.333	35.8785	513.333	42.8971
25	1	Training	1	583.033	28.3251	619.367	29.9534
			2	493.8	27.1844	554.033	35.624
		Verification	1	555.5	30.1945	605.267	33.4024
			2	447.133	29.2182	530.8	38.6812
	2	Training	1	587.933	23.7878	629.9	24.0894
			2	496.567	22.5215	562.733	26.5745
		Verification	1	563.267	27.5818	620.8	27.9092
			2	448.767	35.3189	540.767	31.7106
	7	Training	1	565.233	29.9502	613.733	33.9969
			2	468.833	30.8926	543.9	37.4979
		Verification	1	538.3	32.4177	601.633	37.9532
			2	421.967	32.6649	523.367	43.1776
	12	Training	1	558.2	25.2606	602.667	33.583
			2	468.067	24.1031	539.833	35.6951
		Verification	1	532.467	27.4059	591.033	34.8103
			2	423.833	27.4315	515.933	43.4772
51	1	Training	1	579.767	20.236	613.467	22.2768
			2	487.467	16.9273	548.667	27.6397
		Verification	1	552.633	21.2026	599.367	25.9608
			2	442.533	18.5635	524.633	32.3606
	2	Training	1	582.767	26.67	622.133	27.455
			2	490.467	22.9688	556.033	32.4638
		Verification	1	555.467	32.5203	609.1	31.4339
			2	446.233	27.0474	533.733	36.4833
	7	Training	1	562.2	25.6803	602.467	29.451
			2	474.533	25.4758	534.533	36.5162
		Verification	1	534.1	32.6194	587.567	37.287
			2	427.367	28.2116	509.767	39.0046
	12	Training	1	564.033	28.3251	607.433	29.3089
			2	472.733	25.0763	544.7	33.76
		Verification	1	537.1	33.3315	595.167	37.2263
			2	428.133	32.5754	518.7	36.2664
101	1	Training	1	573.767	30.4537	607.267	37.2178
			2	483.767	25.3876	539.633	33.8195
		Verification	1	544.5	35.3892	592.367	40.3335
			2	433.867	30.1213	511.733	43.7114
	2	Training	1	570.8	27.3564	608.433	32.9824
			2	484.833	26.4433	548.033	39.4142
		Verification	1	542.8	33.8479	593.833	36.9324
			2	434.4	29.2499	521.133	47.4071
	7	Training	1	562.867	24.281	600.833	24.9967
			2	479.367	17.8045	541.7	27.3485
		Verification	1	533.9	32.5538	586.367	30.3616
			2	433.167	28.8146	517.433	37.42
	12	Training	1	548.167	24.7025	586.867	28.0821
			2	459.533	23.3781	516.833	35.246
		Verification	1	516.567	27.7609	568.3	30.858
			2	420.467	24.4847	495.2	31.3307

Table B.15: 18 States – $(12, 59, 7)_4$ – Code #5 – Perfect Score is 708

B.2 Effect of Crossover and Mutation

Measured is the average number of corrections for the best machines found during 30 evolutions. # Mutations refers to the maximum number of edges which will be changed via the mutation operator.

Parameters					Exact		Fuzzy	
Crossover	Mutation	# Mutations	Type	Distance	Average	Std Dev	Average	Std Dev
0	10	1	Training	1	529.533	29.3043	579.133	31.3245
				2	438.367	26.8912	529.233	34.9538
			Verification	1	513.367	35.8373	570.767	37.0244
				2	393.667	29.99	509.5	39.6708
		2	Training	1	523.6	26.1621	581.767	21.431
				2	436.933	25.2586	526.233	28.9169
			Verification	1	504.633	32.5571	570.867	29.2901
				2	387.233	28.1041	502.067	32.3056
		7	Training	1	506.6	22.4984	570.9	26.0323
				2	416.633	21.9238	516.167	32.9054
			Verification	1	492.767	28.4892	561.867	33.569
				2	377.3	26.3218	495	39.1637
		12	Training	1	489.8	28.8676	557.8	32.504
				2	400.933	27.3672	503.467	36.8105
			Verification	1	473	36.5994	547.5	37.7256
				2	366.367	32.1789	484.367	41.2499
	20	1	Training	1	545.833	23.0787	593.9	26.9295
				2	455.233	24.4225	541.767	34.4891
			Verification	1	530.667	30.4714	585.6	33.2748
				2	409.833	25.9523	526.5	39.9083
		2	Training	1	534.433	25.1789	585	27.5706
				2	451.933	21.3847	537.1	32.023
			Verification	1	521.7	28.1782	578.467	29.8776
				2	411.433	28.098	522.067	35.1567
		7	Training	1	529.5	24.6587	583.133	29.4287
				2	442	16.6381	535.667	30.4861
			Verification	1	514.833	25.6462	576.533	27.574
				2	395.7	20.9961	516.3	38.0745
		12	Training	1	509.833	22.2464	567.1	25.8662
				2	426.7	19.126	511.133	36.7064
			Verification	1	495.2	24.0909	556.867	30.3676
				2	382.967	21.8766	490.6	39.2275
	50	1	Training	1	544.1	27.4997	591.133	23.8858
				2	457.267	23.6146	542.433	26.8144
			Verification	1	524.7	33.6474	581.1	28.2249
				2	411.2	26.3889	521.2	32.1413
		2	Training	1	552.067	17.2765	598.467	16.0167
				2	463.7	17.2929	554	20.7597
			Verification	1	538.6	24.2211	592.233	21.5337
				2	419.667	16.9875	532.367	22.8178
		7	Training	1	528.6	20.3446	580.5	24.2512
				2	447.367	15.262	535.333	24.9183
			Verification	1	514.3	23.9571	573.533	27.9689
				2	402.867	17.3875	514.533	33.608
		12	Training	1	521.167	23.2202	577.833	31.1903
				2	436.6	21.5784	527.8	35.9601
			Verification	1	508.333	27.4821	569.367	34.5737
				2	389.3	23.2248	503.467	39.5097

Table B.16: 12 States, Population 51, No Crossover – $(12, 55, 7)_4$

Parameters					Exact		Fuzzy	
Crossover	Mutation	# Mutations	Type	Distance	Average	Std Dev	Average	Std Dev
50	10	1	Training	1	536.5	26.4246	584.567	25.6927
				2	451	26.1969	538.9	30.9464
			Verification	1	525.033	32.3978	581.233	28.717
				2	408.7	27.2158	517.4	33.6776
		2	Training	1	518.6	29.8023	575.033	36.739
				2	435.1	24.5095	520.667	36.0128
			Verification	1	505.033	32.7556	566.733	37.2614
				2	390.867	27.7635	496.967	42.7878
		7	Training	1	516.667	23.5523	570.6	24.2751
				2	426.533	23.9579	513.667	29.2048
			Verification	1	498.8	30.1083	559.8	29.3345
				2	391.2	25.4008	497.367	31.4253
		12	Training	1	502.033	32.1349	564.1	31.2204
				2	413.767	21.7821	507.933	32.7508
			Verification	1	483.133	32.9542	554.067	31.5441
				2	376.2	29.7048	488.033	38.5661
	20	1	Training	1	539.433	23.6216	588	23.5928
				2	453.433	24.9201	539.667	34.5057
			Verification	1	524.433	31.5405	578.8	27.2604
				2	408.867	26.9914	520.167	38.1938
		2	Training	1	538.767	22.7606	583.833	25.3392
				2	453.467	20.4951	535.633	28.9357
			Verification	1	524.3	24.7806	575.467	26.6946
				2	407.567	20.4276	515.867	32.8809
		7	Training	1	536.2	27.9882	584.933	29.667
				2	451.867	24.5564	540.133	31.6214
			Verification	1	523.667	30.5392	580	30.3383
				2	407.933	25.2981	516.6	38.055
		12	Training	1	531.933	19.2639	578.833	24.662
				2	443.467	17.5926	525.733	25.1847
			Verification	1	514.267	25.3934	569	30.0287
				2	403.6	19.925	507.7	30.1618
	50	1	Training	1	551.6	19.258	598.967	21.8387
				2	461.833	19.4051	554.867	30.4254
			Verification	1	538.267	25.6635	594.233	24.8564
				2	418	17.7841	536.467	30.4243
		2	Training	1	550.4	19.2436	590.367	22.1476
				2	468.067	18.4745	546.333	21.1388
			Verification	1	535.567	19.8836	582.067	20.5341
				2	420.9	18.3027	525.133	21.497
		7	Training	1	544.033	16.9207	587.233	20.3583
				2	461	14.319	546.7	20.7766
			Verification	1	530.7	19.0845	581.1	22.335
				2	413.233	17.1679	522.433	26.6105
		12	Training	1	542.667	17.127	597.333	15.0295
				2	453.267	16.1457	552.933	22.6273
			Verification	1	528.667	23.9789	590.367	17.4366
				2	412.567	17.4942	536.5	24.3562

Table B.17: 12 States, Population 51, 50% Crossover – $(12, 55, 7)_4$

Parameters					Exact		Fuzzy	
Crossover	Mutation	# Mutations	Type	Distance	Average	Std Dev	Average	Std Dev
75	10	1	Training	1	523	27.5744	574.033	31.0044
				2	441.467	25.1735	523.633	36.3797
			Verification	1	505.867	31.9318	565.7	33.1019
				2	399.7	27.7118	501.133	35.3658
		2	Training	1	528.867	22.1915	585	22.237
				2	445.3	19.0646	533.433	28.723
			Verification	1	516.4	25.0401	579.367	24.6499
				2	400.567	21.0348	510.6	32.1286
		7	Training	1	518.233	23.2997	575.267	23.22
				2	437.367	21.0508	523.367	27.5887
			Verification	1	502	31.5802	565.667	28.025
				2	387.8	26.5205	501.5	32.429
		12	Training	1	508.333	22.6477	566.833	28.5018
				2	423.867	21.4793	509.067	29.5669
			Verification	1	493.367	23.4322	558.933	30.3894
				2	385.4	25.6308	489.233	39.9385
	20	1	Training	1	549.467	22.0168	596.067	20.3858
				2	460.067	19.6187	549.3	27.1066
			Verification	1	536.133	28.3315	590.667	25.782
				2	417.2	21.4177	528.8	29.0403
		2	Training	1	546.067	18.4502	588.9	21.9849
				2	457.4	17.065	539.567	26.2766
			Verification	1	530.267	22.5418	580.9	23.8304
				2	413.633	20.4425	517.433	29.2712
		7	Training	1	533.933	23.5649	587.233	24.805
				2	446.2	19.4624	532.467	27.6639
			Verification	1	515.533	28.9717	577.833	28.7931
				2	408.8	23.7275	521.1	33.7653
		12	Training	1	532.067	24.8789	577.967	28.0621
				2	448.9	25.8782	529.2	27.3816
			Verification	1	518.933	27.7214	572.1	28.618
				2	410.7	27.8024	509.067	34.8187
	50	1	Training	1	540.1	38.8564	583.4	33.8593
				2	458.067	31.2884	536.067	41.7884
			Verification	1	527.067	40.1437	577.733	36.3526
				2	409.133	32.4661	516.367	42.4544
		2	Training	1	553.533	21.687	593.267	19.9256
				2	466.467	15.0327	546.7	23.2025
			Verification	1	539.033	24.8686	586.3	20.8957
				2	420.667	18.8741	526.767	27.1746
		7	Training	1	548.133	27.0717	588.333	29.5289
				2	458.733	18.6953	545.8	32.3562
			Verification	1	535.267	26.2993	582.633	28.4744
				2	414.633	22.8496	523.633	34.8717
		12	Training	1	532.433	24.7173	577.767	28.3885
				2	444.2	17.0342	529.767	30.9133
			Verification	1	517.867	30.3642	568	33.4169
				2	401.967	23.4822	508.767	39.0143

Table B.18: 12 States, Population 51, 75% Crossover – $(12, 55, 7)_4$

Parameters					Exact		Fuzzy	
Crossover	Mutation	# Mutations	Type	Distance	Average	Std Dev	Average	Std Dev
80	10	1	Training	1	531.067	24.5356	582.567	22.5613
				2	448.367	23.923	531.833	30.085
			Verification	1	518.067	28.9195	575.667	29.7858
				2	405.133	25.9532	514.733	32.4292
		2	Training	1	522.4	27.808	567.9	28.7226
				2	441.467	20.5925	516.767	32.4385
			Verification	1	505.767	33.307	557.533	31.4574
				2	400.133	24.7271	494.3	34.6392
		7	Training	1	511.6	29.5023	563.033	32.783
				2	429.7	20.5781	512.3	35.7647
			Verification	1	499.533	32.7453	555.933	35.0968
				2	390.3	26.4199	488.6	41.5946
		12	Training	1	507.533	30.7591	565.067	32.0742
				2	426.2	28.116	516.833	38.4278
			Verification	1	494.4	34.2854	558.033	31.9703
				2	384.667	28.4524	494.967	39.3407
	20	1	Training	1	539.133	18.1045	584.5	16.6293
				2	456.067	15.9955	536.233	20.3702
			Verification	1	523.5	19.3047	575.3	17.858
				2	412.867	20.2599	518.1	23.8578
		2	Training	1	535.933	23.6861	586.767	25.355
				2	452	20.1186	537.233	22.7
			Verification	1	523.133	26.5924	580.333	25.3545
				2	408.433	25.1117	518.7	26.3794
		7	Training	1	536.567	25.6134	583.233	31.5252
				2	450.4	22.3832	535.267	35.3855
			Verification	1	520.433	31.213	576.933	35.5576
				2	408.1	26.0336	511.033	41.1293
		12	Training	1	529.5	19.6745	574.033	29.3886
				2	448.633	16.9654	524.6	30.93
			Verification	1	512.567	26.1978	564.6	34.2985
				2	407.133	18.8382	502.7	35.9867
	50	1	Training	1	549.9	28.6349	593	30.7089
				2	464.8	22.3073	549.067	34.4553
			Verification	1	536.733	34.7513	588.733	32.2158
				2	418.967	22.842	527.533	41.7577
		2	Training	1	552.367	16.9288	594.533	16.7079
				2	469.267	15.8982	551.4	18.7958
			Verification	1	537.633	20.7256	587.467	18.7133
				2	423.733	19.4315	532.667	26.6618
		7	Training	1	551.333	19.1263	593.733	14.3453
				2	462.933	14.5837	548.333	21.5508
			Verification	1	536	20.7198	585.6	16.9087
				2	414.2	21.6865	525.6	25.9703
		12	Training	1	535.4	25.6684	584.433	27.1492
				2	445.133	19.2044	533.733	31.4719
			Verification	1	520.567	27.9478	578.567	29.0501
				2	406.167	17.5521	515.933	32.548

Table B.19: 12 States, Population 51, 80% Crossover – $(12, 55, 7)_4$

Parameters					Exact		Fuzzy	
Crossover	Mutation	# Mutations	Type	Distance	Average	Std Dev	Average	Std Dev
90	10	1	Training	1	518.4	28.6749	565.3	31.2379
				2	435.7	28.8374	517.1	37.3468
			Verification	1	502.333	31.8697	556.133	35.5564
				2	390.7	29.7787	493.233	41.549
		2	Training	1	539.567	23.2745	592.2	24.0795
				2	447.067	17.3879	543.533	26.3802
			Verification	1	520.467	25.1008	584.2	23.9041
				2	402.733	22.2493	520.767	32.8536
		7	Training	1	515.567	29.9375	568.267	25.7212
				2	435.9	24.4799	517.367	31.7039
			Verification	1	503.7	31.8164	560.633	28.5349
				2	396	26.3308	494.8	35.5764
		12	Training	1	511.4	19.655	567.233	29.562
				2	427.267	18.2491	517.567	31.9716
			Verification	1	495.633	21.5558	557.933	32.3387
				2	381.833	15.5632	488.567	39.5681
	20	1	Training	1	545.867	21.8581	593.2	17.1774
				2	457.8	24.6778	540.7	26.6887
			Verification	1	529.167	29.6498	586	24.0302
				2	417.367	22.0509	522.367	26.2199
		2	Training	1	545.767	17.9361	592.067	20.3655
				2	462.267	16.3579	545.4	24.4563
			Verification	1	531.867	21.5946	586.433	22.7334
				2	414.767	19.9286	521.433	32.9971
		7	Training	1	535.533	22.5858	581.5	24.4142
				2	451.467	20.5707	533.3	27.92
			Verification	1	520.733	30.1078	573.467	27.614
				2	411.967	21.4757	513.5	31.6596
		12	Training	1	532.133	23.2167	584.667	25.4942
				2	449.667	17.2554	531.1	27.1894
			Verification	1	516.267	27.323	575.133	29.9237
				2	408.033	23.4366	510.167	33.4552
	50	1	Training	1	548.567	21.8611	590.833	25.8365
				2	462.067	19.9325	543.7	33.2702
			Verification	1	535.033	28.4853	583.567	33.4125
				2	415.267	24.096	520.833	39.0005
		2	Training	1	551.733	19.5729	594.967	19.5598
				2	459.2	19.63	549.267	26.3032
			Verification	1	536.4	23.6667	588.5	21.8234
				2	415	20.3148	529.5	28.1397
		7	Training	1	552.933	16.609	596.2	16.2319
				2	466.833	18.0174	554.967	23.5555
			Verification	1	543.333	24.2122	589.8	20.0455
				2	422.433	18.2939	539.067	29.118
		12	Training	1	540.1	19.5648	589.4	22.5138
				2	453.833	14.1545	543	24.5638
			Verification	1	530.867	20.8438	584.433	22.5399
				2	413.867	17.1479	523.933	28.8108

Table B.20: 12 States, Population 51, 90% Crossover – $(12, 55, 7)_4$

Parameters					Exact		Fuzzy	
Crossover	Mutation	# Mutations	Type	Distance	Average	Std Dev	Average	Std Dev
100	10	1	Training	1	536.833	20.5327	583.033	21.4452
				2	455.833	16.688	533.567	23.6857
			Verification	1	524.167	22.7461	577.033	24.6037
				2	411.4	17.5904	513	29.6996
		2	Training	1	534.233	19.2832	583.167	21.451
				2	452.933	18.5453	540.133	25.8213
			Verification	1	519.967	25.3887	576.067	23.2793
				2	408.633	21.8356	519.033	25.3887
		7	Training	1	520.533	26.0553	575.9	27.6248
				2	435.133	26.4285	524.167	36.0785
			Verification	1	506.567	33.3091	568.733	31.7696
				2	393.033	27.0153	502.167	38.6773
		12	Training	1	517.633	26.0854	571.9	33.706
				2	440.4	20.5604	524.567	31.1223
			Verification	1	506.367	26.449	566.467	32.1073
				2	398.5	23.5676	500.1	38.9786
	20	1	Training	1	545.433	18.4964	586.8	22.3057
				2	456.7	21.5201	536.1	32.7723
			Verification	1	530.8	26.3601	579.667	28.9772
				2	413.967	20.6573	516.8	32.4658
		2	Training	1	540	29.2787	581.4	30.7712
				2	457.233	24.1414	536.7	34.8367
			Verification	1	525.233	34.1666	574.2	34.3194
				2	413.133	23.1855	515.267	38.6834
		7	Training	1	539.467	19.5426	584.633	22.261
				2	452.367	19.3756	532.733	30.4596
			Verification	1	522.967	23.3112	577.167	24.7109
				2	405.467	18.7004	514.267	27.9999
		12	Training	1	542.467	29.6342	586.733	29.6612
				2	456.7	24.2375	547.167	32.9096
			Verification	1	530.333	33.6076	580.567	31.765
				2	409.367	23.3201	524.067	42.4662
	50	1	Training	1	548.4	21.9036	586.733	21.7302
				2	465.967	16.4369	545.633	24.2665
			Verification	1	534.1	25.613	581.767	23.6857
				2	419.367	16.5831	523.833	26.4576
		2	Training	1	546.4	27.3125	586.933	31.8779
				2	463.633	21.6771	542.067	34.964
			Verification	1	533.2	32.1466	581.167	35.2724
				2	417.267	23.7994	522.633	38.242
		7	Training	1	552.1	10.9964	595.5	15.328
				2	466.833	13.9434	555.567	21.5097
			Verification	1	544.367	16.9675	591.967	18.163
				2	418.167	13.5522	536.567	20.8916
		12	Training	1	540.433	14.885	586.867	17.1881
				2	447.533	14.9337	538.933	24.053
			Verification	1	530.533	17.1861	579.767	20.584
				2	408.767	15.6419	522.567	21.3714

Table B.21: 12 States, Population 51, 100% Crossover – $(12, 55, 7)_4$

“An intractable problem can only be resolved by stepping beyond **conventional solutions**.”

— Ozymandias, Watchmen.