Bounds on Edit Metric Codes with Combinatorial DNA Constraints

by Jing Sun

Department of Computer Science

Submitted in Partial Fulfillment

of the Requirements for the Degree of

Master of Science

Faculty of Mathematics and Science, Brock University

St. Catharines, Ontario, Canada

Approved for the Committee:

Dr. S. Houghten

Dr. K. Qiu

Dr. D. McCarthy

Department of Computer Science

Supervisor Dr. S. Houghten

Acknowledgements

I would like to express my gratitude to my supervisor, Dr. Sheridan Houghten, for her support, patience and encouragement throughout my graduate studies. With her enthusiasm, guidance and great effort to explain things clearly, I am devoted to the research field that I am interested in and overcame lots of difficulties that unavoidably crop up in the processes of performing research. Her technical and editorial advice were essential to the completion of this dissertation and have taught me innumerable lessons and insights on the workings of academic research in general.

My thanks also go to the members of my committee, Dr. Ke Qiu and Dr. Dave McCarthy for reading my research proposal and previous draft of this dissertation and providing valuable comments and suggestions that help with the completion of this dissertation.

The friendship of my fellow students is much appreciated and has led to many interesting and inspirited discussions relating to this research. My thanks go to Robert Flack and Yi Feng Li for their helping with various ideas of coding implementation.

Most importantly, I wish to thank my parents, Zhigang Sun and Hong Yu. They bore me, raised me, supported me, taught me, and loved me. To them I dedicate this thesis. I am also grateful to my aunt Ping Patel Yu, my uncle Narsh Patel and my grandma Xingzhi Li for their dedication, encouragement, love and support. And I would also like to thank my husband, Xiang Yin, for his understanding and love during past few years. His support and encouragement were in the end what made this dissertation possible.

Finally, I appreciate the financial support from our department, Computer Science Department of Brock University for my graduate studies and the great working environment it provided me.

This document was prepared using TeXnicCenter.

Abstract

The design of a large and reliable DNA codeword library is a key problem in DNA based computing. DNA codes, namely sets of fixed length edit metric codewords over the alphabet $\{A, C, G, T\}$, satisfy certain combinatorial constraints with respect to biological and chemical restrictions of DNA strands. The primary constraints that we consider are the reverse–complement constraint and the fixed GC–content constraint, as well as the basic edit distance constraint between codewords.

We focus on exploring the theory underlying DNA codes and discuss several approaches to searching for optimal DNA codes. We use Conway's lexicode algorithm and an exhaustive search algorithm to produce provably optimal DNA codes for codes with small parameter values. And a genetic algorithm is proposed to search for some sub–optimal DNA codes with relatively large parameter values, where we can consider their sizes as reasonable lower bounds of DNA codes. Furthermore, we provide tables of bounds on sizes of DNA codes with length from 1 to 9 and minimum distance from 1 to 9.

Contents

List of Committee Members ii

Acknowledgements iii

Abstract v

List of Tables ix

List of Figures x

1 Introduction 1

1.1 Fundamentals of Coding Theory 1

1.2 Biological Background of DNA Molecules 3

1.3 Applications of DNA Codes 5

1.3.1 DNA Computing 5

1.4 DNA Code Design Problem 6

1.5 Concept of DNA Codes 7

1.6 Organization of the Thesis 9

2 Previous Work 10

2.1 Previous Work on Error–Correcting Codes 10

2.2 Previous Work on Edit Metric Codes 11

2.3 Previous Work on Various DNA Codes 12

3 DNA Code Space **15**

3.1 DNA Word Space 15

3.1.1 Graph of DNA Word Space 15

3.1.2 Words of Length 1 16

3.1.3 Words of Length 2 17

3.1.4 Words of Length 3 18

3.1.5 Graph for Words of Different Lengths 19

3.2 Permutations that Preserve the Fixed GC–content 21

3.3 Definition of Equivalent DNA Codes 26

3.4 Properties of Equivalent DNA Codes 28

4 Properties of DNA Codes **30**

4.1 Theorems of Optimal DNA codes 30

4.1.1 Binary Reverse–Complement Codes 35

4.2 Lower and Upper Bounds 42

5 Algorithms **43**

5.1 Conway's Lexicode Algorithm 43

5.1.1 Outline of Conway's Lexicode 44

5.1.2 Table of Bounds on DNA Codes by Conway's Lexicode 44

5.2 Exhaustive Search Algorithm 45

5.2.1 Outline of Exhaustive Search 46
5.2.2 Optimizations of Exhaustive Search 48
5.2.3 Table of Sizes of Optimal DNA Codes by Exhaustive Search 50

6 Genetic Algorithm **52**

6.1 Background of Genetic Algorithm 52
6.2 The Operators of Genetic Algorithm 53
6.3 Parameters of Genetic Algorithm 54
6.4 Outline of General Genetic Algorithm 55
6.5 Genetic Algorithm for DNA Codes 55
6.5.1 Parameter Sets 56
6.5.2 Representation 56
6.5.3 Fitness Function 57
6.5.4 Initialization 57
6.5.5 Selection operator 57
6.5.6 Repair Operator 58
6.5.7 Crossover Operator 58
6.5.8 Mutation Operator 60
6.5.9 Replacement Operator 60
6.6 Examples of Experiments 60
6.7 Tables of Sizes of DNA Codes by Randomized Genetic Algorithm 63

7 Conclusion **66**

Bibliography **69**

List of Tables

5.1 Bounds on DNA Codes by Conway's Lexicode Algorithm 45

5.2 Bounds on DNA Codes by Exhaustive Search Algorithm 50

6.1 Sizes of DNA Codes by RGA of Population Size 50 62

6.2 Sizes of DNA Codes by RGA of Population Size 100 62

6.3 Sizes of DNA Codes by RGA of Population Size 200 63

6.4 Bounds on DNA Codes by RGA under $R_c = 1.0$, $R_m = 0.0$ 64

6.5 Bounds on DNA Codes by RGA under $R_c = 0.0$, $R_m = 1.0$ 65

6.6 Bounds on DNA Codes by RGA with Elitism under $R_c = 0.9$, $R_m = 0.1$. 65

7.1 Best–known Bounds on DNA Codes 67

List of Figures

1.1 Image of a DNA double helix 3

3.1 DNA code space for length of 1 16

3.2 DNA code space for length of 2 17

3.3 DNA code space for length of 3 19

3.4 DNA code space for length of 0 to 2 20

4.1 Template–Map construction for a $(4, 2, 2)_4$ DNA code 39

5.1 Exhaustive Search procedure for $(2, 1, 1)_4$ codes 47

5.2 Pruning method of Exhaustive Search for $(2, 1, 1)_4$ codes 49

6.1 Repair of a $(4, 3)$ DNA code 58

6.2 Crossover of two $(4, 3)$ DNA codes 59

6.3 Randomized GA on $(7, 3, 3)_4$ DNA codes 61

6.4 Different population sizes on $(7, 3, 3)_4$ DNA codes 61

Chapter 1

Introduction

In this chapter, we first introduce the basic concepts of coding theory and biological background of DNA. Various applications using DNA are presented. We briefly define DNA codes over edit distance, restricted with certain biological constraints. We consider the problem of finding optimal DNA codes in this thesis.

1.1 Fundamentals of Coding Theory

Communication channels are used to transmit messages from a source to a receiver. However, errors can occur during the transmission due to noise, and distort the message so that what is received is not always the same as what was sent. Error–correcting codes are designed to correct certain possible errors that occur when the channel is noisy. An *error–correcting code* is a set of strings over an alphabet set, in which each string is called a *codeword.* The codewords are separated from each other by a *distance*, thus it is capable of avoiding the errors that occur in a code when transmitting.

Let x, y be two codewords. The *Hamming distance* $d_H(x, y)$ between two words x and

y is defined to be the number of coordinates in which x and y differ. For example, if we have two binary words $x = 000111$ and $y = 101010$, then $d_H(000111, 101010) = 4$. Codes over *Hamming distance* can be used to correct substitution errors, which have been extensively studied, see more in books [18, 29]. Synchronization errors can cause part of the message to be stripped off or cause additional symbols to be inserted into the message from the sender. Codes over the *edit metric* are designed to correct both substitution errors and synchronization errors. Edit distance, also known as *Levenshtein distance*, is the minimum number of substitutions, deletions or insertions that are required to transform one word into another. For example, the edit distance between two words 011100 and 001110 equals $d_{edit}(011100, 001110) = 2$, since we first delete the last symbol 0 in the word 011100 and insert it in front of this word, obtaining word 001110.

A q–ary code with pre–determined *minimum distance* d and *length* n is denoted as an $(n, d)_q$ code, where the minimum distance is the smallest distance between any two distinct codewords, and the length is the number of symbols in a codeword. The notation $(n, M, d)_q$ is an $(n, d)_q$ code with M *codewords*, where M is defined as the *size* of the $(n, d)_q$ code.

The maximum size of a q–ary Hamming distance code of length n and minimum distance d is denoted by $A_q(n, d)$. An $(n, M, d)_q$ code is *optimal* if $M = A_q(n, d)$ for Hamming distance codes. Similarly, we denote $E_q(n, d)$ as the maximum size of a q–ary edit distance code of length n and minimum edit distance d. We define an $(n, M, d)_q$ edit metric code to be *optimal* if $M = E_q(n, d)$. Exact values and bounds for $A_q(n, d)$ can be found in [18, 22]. Tables of bounds for edit metric codes are investigated in [17], for quaternary codes with the length n from 1 to 10 and minimum distance d from 1 to 10.

1.2 Biological Background of DNA Molecules

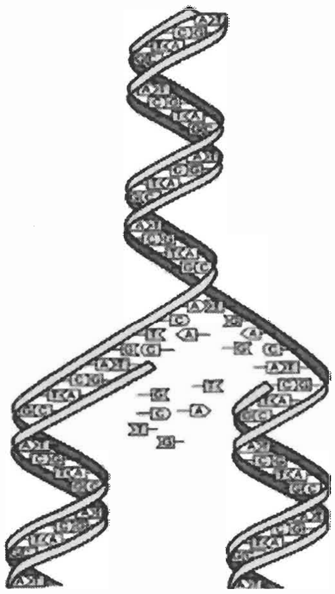

Figure 1.1: Image of a DNA double helix

DNA (deoxyribonucleic acid), shown in an Figure 1.1 in [33], consists of two strands that form a spiral called a double helix. A single DNA strand is a sequence of nucleotides, which has chemically distinct ends as the $3'$ and $5'$ ends. Each nucleotide contains one base, one phosphate molecule and the sugar molecule deoxyribose. The $3'$ end has a terminal hydroxyl group and the $5'$ end has a terminal phosphate group. The $3'$ end and $5'$ end are called the asymmetric ends of DNA strands, which indicate the direction of two DNA strands in the DNA double helix. These DNA strands are antiparallel; the direction of nucleotides in one DNA strand is opposite to the direction of the nucleotides in the other strand. The information in DNA is stored as a code made up of four bases: adenine (A), guanine (G), cytosine (C), and thymine (T). Abiding by the Watson–Crick law in base paring, A pairs up with T and C with G.

Hybridization, known as base pairing, occurs when a single strand bonds to another single strand, to form a double strand. The Watson–Crick complementary strand of a DNA strand is obtained by replacing every A with a T and every C with a G and vice versa, while also switching the $3'$ and $5'$ ends. For instance, the Watson–Crick complementary strand of $3' - ACTGAA - 5'$ is $5' - TGACTT - 3'$. In specific hybridization, a strand bonds to its Watson–Crick complementary strand. Specific hybridization can be used to identify and retrieve target DNA words from a set of DNA words: if we wish to obtain a particular DNA word in the DNA words pool, we can use its Watson–Crick complementary word to bond to it, forming a double helix by specific hybridization. Then, we split this DNA double helix, thus obtaining the desirable word. Non–specific hybridization occurs when hybridization between a DNA strand and its Watson–Crick complement does not take place as intended; for example, a DNA strand may bond to the Watson–Crick complement of some other DNA strand, or to the reverse–complement of some other strand. For example, for a DNA strand $ACTGAA$, specific hybridization occurs when it bonds to its Watson–Crick complementary strand $TGACTT$. However, non–specific hybridization takes place if it bonds to some other strand like $TCCCAT$ or to the Watson–Crick complementary strand of $TCCCAT$, $ATGGGA$. Non–specific hybridizations can occur in a self–assembly process, in a polymerase chain reaction or in an extraction operation in lab experiments.

1.3 Applications of DNA Codes

The DNA molecule is widely used in lab applications such as probe selection for DNA microarrays [27], or primer design for PCR [5, 7]. It is also applied in nanotechnology as three dimensional structural materials and as an information storage medium.

From a computer science point of view, one of the most significant application for DNA molecules is DNA computing, which is based on innovative work by Adleman. In Adleman's paper [1], he proposed using DNA molecules to solve a well–known mathematical problem, called the 'Hamiltonian Path Problem'. The goal of this problem is to find a path that starts from a given starting city and terminates at a given ending city by going through the intermediary cities exactly once. His experiment was performed on a seven cities and fourteen flights map. He encoded the seven cities and possible paths into DNA sequences, synthesized the Watson–Crick complementary DNA sequences of each city and mixed them in a test tube. Hence, all of the possible combinations of DNA sequences were created in the test tube during the chemical reaction. Then, he eliminated the wrong molecules and took only those in which the paths connected all seven cities as the final solution. The feasibility of using DNA molecules for computation is shown through Adleman's experiments.

1.3.1 DNA Computing

DNA computing is highly parallelized, selecting a large amount of special–purpose DNA strands to hybridize and filtering the final solution. With a given setup and enough DNA strands, DNA computing can potentially solve huge problems and perform calculations much faster than the most powerful human–built supercomputers. It accomplishes massive parallelism by allowing multiple strands to hybridize simultaneously in a large pool of DNA strands. DNA computing also has a powerful storage capability because biochips are much smaller than traditional chips but hold enormous numbers of DNA molecules. For these reasons, there is a growing interest in using DNA computing. The reliability of DNA computing requires the availability of a large set of DNA strands which can achieve specific hybridizations and avoid non–specific hybridizations.

1.4 DNA Code Design Problem

The DNA sequence design problem is to construct a large set of single DNA strands that are likely to hybridize to their target strands in the predetermined way, minimizing the chances that errors occur in unintended hybridizations. Various metrics have been proposed. There are simple measures such as comparing two DNA sequences coordinate by coordinate, the GC–content restriction on DNA sequences in which the number of Gs and Cs is determined, and more sophisticated and realistic models that focus on thermodynamic factors like estimating free–energy [7] to produce DNA strands with similar melting temperatures[20]. The Gibbs energy and nearest neighbor thermodynamics scheme considers both factors of avoiding secondary structure of DNA strands from non–specific hybridization and thermodynamic factors of DNA strands. It provides a more accurate measurement in most research work, stated in [6, 7]. However, it is computationally time consuming. Simpler metrics like Hamming distance, as used in most conventional error–correcting codes, have also been used, allowing the correction of substitution errors occurring in DNA sequences. The authors of [30] argued about several types of metrics, and proposed the use of a robust metric called edit distance, which is not only capable of correcting combinations of substitution, insertion and deletion errors, but is also feasible in terms of computational complexity. In this thesis, we use edit distance to measure the similarity of DNA strands.

1.5 Concept of DNA Codes

We define an $(n, d, w)_4$ DNA code to be a set of codewords over a quaternary alphabet $\{A, C, G, T\}$, that has fixed length n, minimum edit distance d and fixed GC–weight w where w is the total number of Gs and Cs in a word. In addition, the DNA Code must satisfy certain combinatorial biological constraints.

In this thesis, three major constraints on DNA codes are considered:

1. **Edit Distance Constraint**

   The edit distance constraint for a DNA code $\mathcal{C}$ is that $d_{edit}(x, y) \geq d$ for all $x, y \in \mathcal{C}$, with $x \neq y$, for some prescribed minimum distance d. For example, if we have two DNA words $TACCTG$ and $ACCTGT$, then $d_{edit}(TACCTG, ACCTGT) = 2$, because we can simply remove the first T of word $TACCTG$ and append a T at the end of this word to obtain the word $ACCTGT$.

   The edit distance constraint can reduce non–specific hybridizations between distinct codewords, as well as correct the insertion, deletion and substitution errors in codewords. The computation of the edit distance can be simply solved by applying a dynamic programming algorithm with time complexity of $O(n^2)$ for words length n, see [34].

2. **Reverse–Complement Constraint**

The reverse–complement constraint is that $d_{edit}(x^{RC}, y) \geq d$ for all $x, y \in \mathcal{C}$, including $x = y$ for some prescribed minimum distance d. The Watson–Crick complement of each nucleotide is denoted by $\overline{A} = T$, $\overline{T} = A$, $\overline{C} = G$ and $\overline{G} = C$. The reverse–complement of $x = (x_1, x_2, ..., x_n)$ is denoted as $x^{RC} = (\overline{x}_n, \overline{x}_{n-1}, ..., \overline{x}_1)$. For example, the DNA word $z = AACCGGTT$ does not satisfy the reverse–complement constraint since it is *self–complementary* and thus $d_{edit}(z^{RC}, z) = 0$. So, such words can not be selected for constructing our DNA words set.

The reverse–complement constraint limits non–specific hybridizations between codewords and the reverse–complements of codewords, including the reverse–complement of itself.

3. **Fixed GC–Content Constraint**

   The fixed GC–content constraint requires that each DNA strand contain the same total number of Gs and Cs, denoted by w for a code $\mathcal{C}$ or $W_{GC}(x)$ for some codeword x.

   It guarantees that all DNA words in the set have similar melting temperature, therefore hybridization of multiple words can take place simultaneously, see [7]. In this thesis, we present results for a fixed GC weight of $\lfloor \frac{n}{2} \rfloor$ for DNA codes of length n since this has been demonstrated as the best value to retain each DNA word in a proper temperature range in lab experiments.

It is noted that the constraints we consider may not concern about certain issues which could be crucial in specific practical applications. One of the important constraints is the forbidden subwords constraint, that a class of substrings must not occur in any codeword or concatenation of codewords, mentioned in [31]. The consecutive–bases constraint forbids long runs of the same base. The frame-shift constraint, also mentioned in [31], requires that concatenation of two or more codewords should not properly contain another codeword. Those constraints will not be discussed in this thesis, but may possibly be examined in future research.

An $(n, d, w)_4$ DNA code is *optimal* if it has the maximum possible number of codewords for given parameters of n, d and w. In this thesis, we consider the problem of finding DNA codes which contain a large number of codewords. Exhaustive search is used to obtain optimal DNA codes with relatively small values of n and d. However, for those with larger values of n and d, it is not computationally feasible. Thus, a genetic algorithm is used to search for large codes within a reasonable period of time.

1.6 Organization of the Thesis

The remainder of this thesis is organized as follows:

Chapter 2 gives a literature overview of methodologies and algorithms that have been previously utilized to find good predefined DNA codes. Chapter 3 describes the DNA word space according to the above constraints and gives a definition of equivalent codes. Chapter 4 focuses on optimal DNA codes, and establishes the size of optimal codes for some specific parameter values. Chapters 5 and 6 present several algorithms that we designed for searching for good DNA codes and tables of lower bounds are given. In Chapter 7, we analyze the approaches that we applied to design good DNA codes and consider further future work.

Chapter 2

Previous Work

In this chapter, we first show some early work on edit metric codes. Then, we give a overview of previous work which involves various approaches to finding bounds on types of DNA codes, addressing different experimental issues.

2.1 Previous Work on Error–Correcting Codes

Research on error–correcting codes can be traced back to the early 1950s. Enormous work has been done and different types of codes have been presented. The fundamentals of error–correcting codes may be found in books [18, 22, 29], etc. Different types of upper bounds and lower bounds of both linear and non–linear codes have been investigated. Tables of bounds are given in [8] for binary Hamming codes, in [10] for ternary Hamming codes and in [9] for quaternary Hamming codes.

The Lexicode algorithm is first used for binary linear codes. The earliest work using genetic algorithms is presented in [24], employing a general genetic algorithm to search for linear binary codes with given n and d. In [16], the authors transform the problem of finding bounds on sizes of optimal error–correcting code over Hamming distance into the Maximum Clique Problem. Heuristics as Hill Climbing, Simulated Annealing, greedy methods and Genetic Algorithms were applied and compared. The empirical results showed that the Genetic Algorithm outperformed the other algorithms.

2.2 Previous Work on Edit Metric Codes

The theory of edit metric codes over small alphabet sets was explored and some theoretical upper bounds were derived in [17]. The authors also indicated that the structure of the edit space is heavily dependent on the *block structure* of the words, which partitions words into maximal subwords of a single symbol. Therefore, the theory of edit metric codes is significantly different from that of the conventional Hamming distance codes. An exhaustive search method was employed to search for exact values and upper bounds on sizes of optimal edit codes for small parameter values. A similar approach is also used in this project, to find optimal DNA codes for small parameter sets.

Different strategies have been applied in constructing good edit codes for larger parameter values. Ashlock, Guo and Qiu [3] developed a program that used a genetic algorithm to control a greedy algorithm, known as Conway's Lexicode Algorithm [18]. They designed their chromosome to be a list of codewords of a given small size m, termed a 'seed'. A seed consists of m codewords which are compatible with each other. Then they used Conway's algorithm to finish this partial code and defined the fitness of a chromosome to be the size of the code obtained after applying the greedy algorithm to the corresponding seed. The GA operators such as crossover and mutation were employed only on seeds, and those modifications had a huge effect on completing the partial codes. They demonstrated that this deflection of the greedy algorithm potentially exploits the search space and a large number of good edit codes were found.

Optimal variable–length insertion–deletion codes and edit metric codes have been studied in [4]. The theory of such codes has been investigated and an exhaustive search has been performed to obtain exact values of sizes of codes.

2.3 Previous Work on Various DNA Codes

King [20] provided theoretical upper bounds and lower bounds on sizes of DNA codes with fixed GC–content over Hamming distance. Codes both with and without the additional reverse–complement constraint were discussed and tables of lower bounds on sizes of codes are given respectively. Gaborit and King in [14] proposed a new construction for Hamming DNA codes, that are derived from additive and linear codes over $GF(4)$. Using a combination of lexicographic techniques and stochastic search, they provided a table of bounds up to length 20. Chee and Ling designed a stochastic local search in [11], using an 'acceptance' probability function to determine a random neighborhood search. The 'acceptance' probability function is used to accept a move to a solution at least as large as the best solution found so far, and reject a move to a solution that has a size three or more smaller than the best known solution. Their algorithm improved one third of the values of lower bounds for Hamming DNA codes with constant GC–content constraint and reverse–complement constraints in [14]. Tulpan and Hoos in [32] utilized different types of neighborhood functions to improve a general stochastic local search. They start a code with a set of randomly selected DNA sequences, then repeatedly replace a conflicting word x that violates more than one constraint with a 'neighborhood' word until a valid code is found. They proposed three types of neighborhood mechanism: v–Mutation Neighborhood, Pure Random Neighborhoods and Hybrid Randomized Neighborhoods. The neighborhood words of a given word x in the first neighborhood method can be obtained by modifying up to v bases from x, while satisfying the fixed GC–content constraint as well. The Pure Random Neighborhood selects a fixed number of random words with the same GC–content as x to be the neighborhood of x. The Hybrid Randomized Neighborhood combines the first two methods, adding random words to the set of v–mutation neighborhoods, which allows the search to escape from local optima. The impact of those three neighborhoods on the performance of stochastic local search have been analyzed and compared.

In [25], a modified genetic algorithm with mutation–only heuristic was used to produce some initial DNA codes, then exhaustive search was applied to extend codes across the entire search space of possible words, thus yielding codes of maximum size. They also implemented their hybrid architecture by using parallel genetic algorithms, incorporating hardware accelerators to calculate edit distance and speed up the exhaustive search. They significantly improved the computational time required to generate good DNA codes up to length 16.

Several approaches to designing good DNA sequences for other experimental purposes have been investigated. In [15], the authors proposed a new metric, called H–measure, as a measurement for hybridization likelihood. They indicated that the H–measure allows the development of encoding criteria and guarantees reliability for DNA computing. In [19], a multi–objective genetic algorithm was used to generate sets of DNA sequences which reflect more realistic characteristics of DNA for practical lab applications, considering various constraints including fixed GC content, H–measure and melting temperature, etc. The fitness of a code is evaluated by Pareto optimization. Feldkamp, Rauhe and Banzhaf in [13] implemented a software tool called the DNA Sequence Generator and the DNA Sequence Compiler. The approach of measuring dissimilarity between DNA sequences is based on the uniqueness of overlapping subsequences, see [26, 28] and a graph based algorithm was applied for DNA sequence generation. Furthermore, constraints such as melting temperature and forbidden subsequences were also considered in their software.

Chapter 3

DNA Code Space

In this chapter, we describe the search space of DNA codes with small length n. We discuss how constraints affect DNA words, and finally define the equivalence of DNA codes.

3.1 DNA Word Space

3.1.1 Graph of DNA Word Space

We represent the DNA word space by a graph $\mathcal{G}$ in which *words* are represented by vertices for our alphabet $\mathcal{D} = \{A, C, G, T\}$, and the vertex set at 'level' n is $\mathcal{D}^n$, representing all possible words of length n. The edge set E is $\{(x, y) : \min(d_{edit}(x, y), d_{edit}(x, y^{RC})) = 1, x, y \in \mathcal{D}^n\}$. Two vertices x and y are connected by an edge if and only if x can be transformed into y or y^{RC} by a single operation of insertion, deletion or substitution.

In graph $\mathcal{G}$, we classify all 4^n words of length n into 3 subsets,

1. The words in the set $c_0 : \{x \in \mathcal{D}^n : d_{edit}(x, x^{RC}) \geq 1 \text{ and } W_{GC}(x) = \lfloor \frac{n}{2} \rfloor\}$ are valid words in the space, that is, they can be chosen as codewords in a code of length n. In

the following graphs, they are denoted by black dots.

2. The words in the set c_1 : $\{x \in \mathcal{D}^n : d_{edit}(x, x^{RC}) = 0\}$ are words that are *self–complementary*, and thus can never be chosen as codewords for a code. These are denoted by squares in the following graphs. It is noted that self–complementary words only exist for codes with even length.

3. The remaining words, in the set c_2, are those that are not self–complementary and do not satisfy $W_{GC}(x) = \lfloor \frac{n}{2} \rfloor$. Since we wish to find codes with a GC–content of 50%, these words can not be included in our codes. However, if an application requires a different GC content or if this restriction is not required, then these words could be valid. They are marked as empty dots in the following graphs.

In the graph $\mathcal{G}$, we connect pairs of words x, y such that $\min(d_{edit}(x, y), d_{edit}(x, y^{RC})) = 1$, as adjacent words in the space. A solid line connects any two adjacent valid words contained in c_0, while a dashed line indicates the adjacency of a valid word from c_0 with an invalid word which lies in set c_1 or c_2, or between any two invalid words from c_1 or c_2.

3.1.2 Words of Length 1

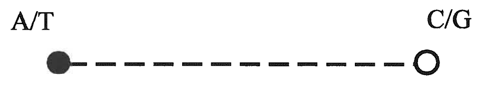

Figure 3.1: DNA code space for length of 1

Figure 3.1 shows the DNA code graph for words of length 1. Consider the words $x = A$ and $y = T$. Since they are reverse–complements of each other, we have $d_{edit}(x, y^{RC}) = 0$.

Furthermore, the edge set connecting x to other vertices is exactly the same as the edge set connecting y to other vertices in the graph. Therefore, A and T 'share' a vertex, denoted by $\{A/T\}$. Similarly, C and G also share a vertex, denoted by $\{C/G\}$. In the graph, vertex $\{A/T\}$, is marked black since it satisfies $d_{edit}(x, x^{RC}) \geq 1$ and $W_{GC}(x) = \lfloor\frac{1}{2}\rfloor = 0$ where $x \in \{A, T\}$, while vertex $\{C/G\}$ is marked by an empty dot. Vertices $\{A/T\}$ and $\{C/G\}$ are connected by a dashed line since the minimum distance of both of $\{A, T\}$ with both of $\{C, G\}$ equals 1.

3.1.3 Words of Length 2

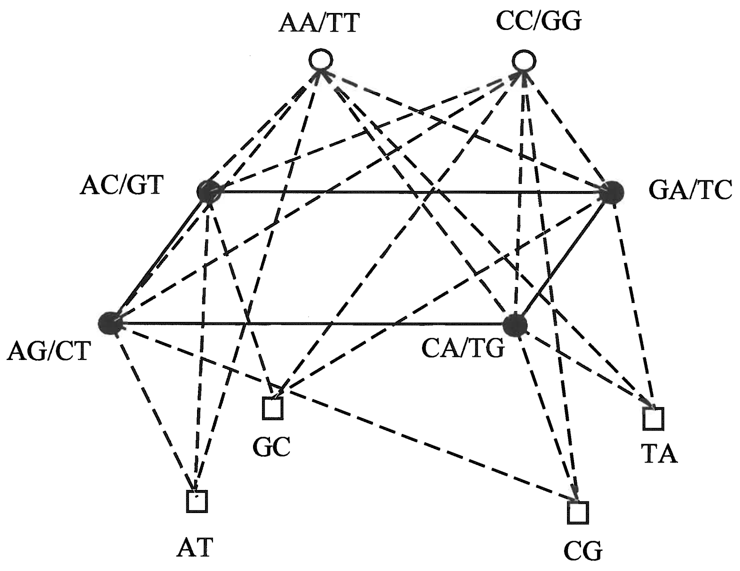

Figure 3.2: DNA code space for length of 2

Figure 3.2 shows the DNA code graph for codewords of length 2. All the 4^2 words are partitioned into three sets c_0, c_1 and c_2 as described before. As is defined, every word is grouped together with its reverse–complement word to form a single vertex. Thus, the set

c_0 consists of vertices $\{AG/CT\}$, $\{CA/TG\}$, $\{AC/GT\}$, $\{GA/TC\}$ since these words obey the constraints $d_{edit}(x, x^{RC}) \geq 1$ and $W_{GC}(x) = \lfloor \frac{2}{2} \rfloor = 1$. The set c_1 consists of vertices AT, TA, CG and GC, since for these words, the distance between a word and its reverse–complement is 0. The vertices $\{AA/TT\}$, $\{CC/GG\}$ are contained in set c_2 because they are neither self–complementary nor satisfy $W_{GC}(x) = \lfloor \frac{2}{2} \rfloor = 1$. As indicated, any two adjacent valid words are connected by a solid line, so that $\{AG/CT\}$ is adjacent to $\{AC/GT\}$ and $\{CA/TG\}$, and GA/TC is adjacent to $\{AC/GT\}$ and $\{CA/TG\}$, and vice versa. Then, the vertex $\{AA/TT\}$ is connected with $\{AG/CT\}$, $\{CA/TG\}$, $\{GA/TC\}$, $\{AC/GT\}$, $\{AT/TA\}$ by a dashed line since $\{AA/TT\}$ does not belong to the valid word set. The same applies to the edges for vertex $\{CC/GG\}$. The word AT is self–complementary and thus does not share a vertex with any other words; it is connected by a dashed line with vertices $\{AA/TT\}$, $\{AC/GT\}$ and $\{AG/CT\}$. This also applies to the edges for vertices TA, CG and GC.

3.1.4 Words of Length 3

Figure 3.3 shows the DNA code subgraph for valid words of length 3. Since there are 4^3 words on 'level' 3, we only present the graph that contains the valid words, in c_0. These words are connected by a solid line if they are adjacent to each other. For example, word $\{AAC/GTT\}$ is adjacent to $\{AAG/CTT\}$, $\{ATC/GAT\}$ and $\{TAC/GTA\}$. The same also applies to the remaining 11 vertices. Two vertices that are disconnected implies that the edit distance between them is greater than one.

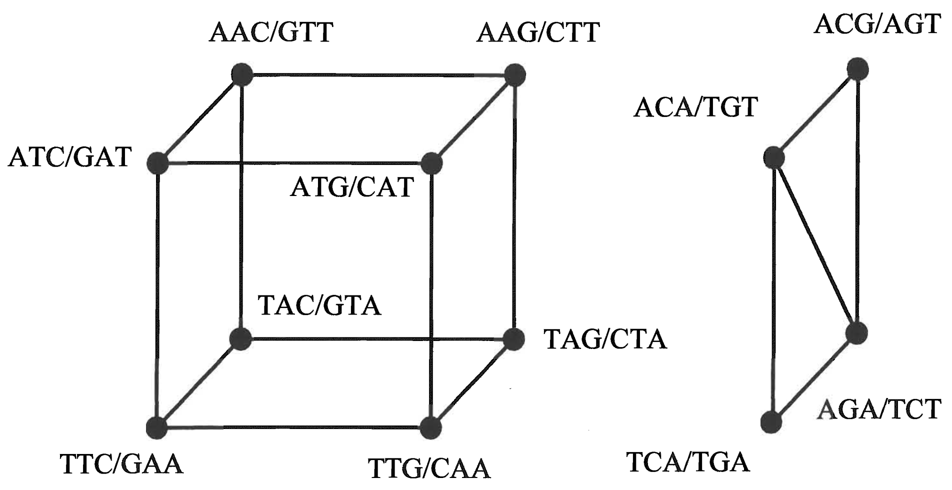

Figure 3.3: DNA code space for length of 3

3.1.5 Graph for Words of Different Lengths

To describe the insertion and deletion space on DNA words, we define a graph $\mathcal{G}$ that has vertex set V^* which consists of all words $\mathcal{D}^n$ at each level n, $n \geq 0$ and edge set E which is $\{(x, y) : \min(d_{edit}(x, y), d_{edit}(x, y^{RC})) = 1, x, y \in V^*\}$. Thus, the adjacent words of a given vertex either share the same length or differ in length by one. Figure 3.4 shows the DNA word space for words of lengths 0 to 2, where λ represents the *empty* word, which is self–complementary, at level 0. For example, the graph indicates that the distance between word GC to word CG is 2. When we delete C from word GC, we obtain a word G on level 1. Then, we insert C at the start of G, and obtain a word CG which is on level 2. It is observed from the graph that the size of DNA word space grows exponentially when the length of words increases.

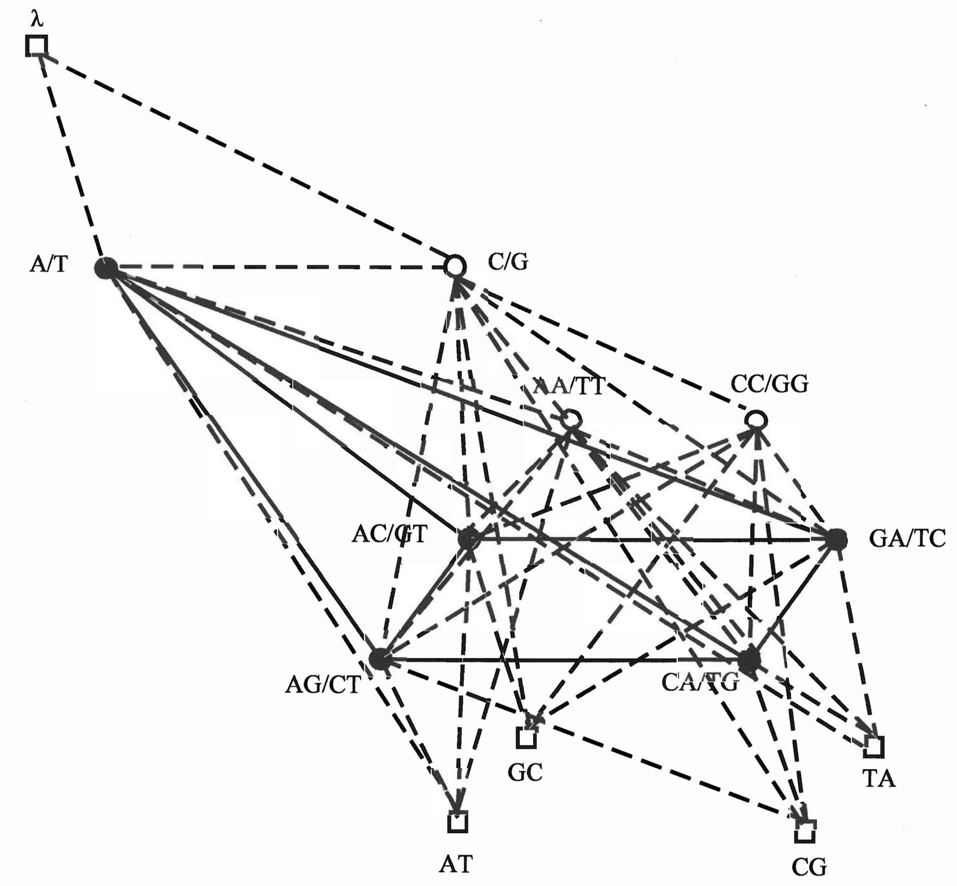

Figure 3.4: DNA code space for length of 0 to 2

3.2 Permutations that Preserve the Fixed GC–content

Given alphabet set $\mathcal{D} = \{A, C, G, T\}$, let $Sym(\mathcal{D}^o)$ be the set of all permutations on $\mathcal{D}$ and $Sym(\mathcal{D})$ be the set of all permutations on $\mathcal{D}$ that maintain the same GC weight for every word in the DNA space. Since $|\mathcal{D}| = 4$, then $Sym(\mathcal{D}^o)$ has $4!$ permutations.

We represent a permutation in *one line* notation or *cycle* notation. For example, consider $[2, 3, 1, 4]$, one of the permutations of $[1, 2, 3, 4]$ in one line notation. In this notation, the symbol in the first position is changed into the symbol in the second position. Similarly, the symbol in the second position is changed into the third position and the symbol in the third position is changed into the symbol in the first position, while the last symbol remains the same. In cycle notation, this permutation is represented as as $(123)4$. For example, apply $[T, G, C, A]$, as a permutation of $[A, C, G, T]$, on a word $ACGGTA$, and thus obtain a new word $TGCCAT$.

Among the 24 permutations in $Sym(\mathcal{D}^o)$, some permutations also preserve the same total number of Gs and Cs in every word of DNA word space; only those permutations are in $Sym(\mathcal{D})$. The cases will be enumerated here and counterexamples will be given for those cases that do not satisfy the fixed GC–content constraint.

1. The identity permutation is $[A, C, G, T]$. Clearly, this retains the same GC weight in all DNA words, as all words are unchanged.

2. Now consider those permutations that only exchange Cs with Gs and/or As with Ts. All of those maintain the same GC weight in every DNA word to which they are applied. These permutations are $[A, G, C, T]$, $[T, C, G, A]$, $[T, G, C, A]$.

   First consider $[A, G, C, T]$. Exchanging Cs with Gs maintains the same total number of Cs and Gs in every word in the space, so that the GC weight remains the same.

Therefore, it is an element of $Sym(\mathcal{D})$. Similarly, permutation $[T, C, G, A]$ is an element of $Sym(\mathcal{D})$ since exchanging As with Ts also maintains the same total number of Cs and Gs in any word in the space.

Now consider $[T, G, C, A]$. It is easy to identify that this permutation is the combination of $[A, G, C, T]$ and $[T, G, C, A]$. It is apparent that simultaneously exchanging character A with character T and character C with character G can not change the GC weight in a word.

3. Now consider those permutations that replaces As with either Gs or Cs, and simultaneously replace Ts with the remaining symbol Cs or Gs; furthermore, Cs are replaced with either As or Ts and Gs are replaced with the remaining symbol Ts or As. These are $[C, A, T, G]$, $[C, T, A, G]$, $[G, A, T, C]$ and $[G, T, A, C]$. All of these only preserve the same GC weight after they are applied to a word with even length and GC weight of $\frac{n}{2}$.

   Among these permutations, we use $[C, A, T, G]$ to illustrate how it affects the GC weight in a word. For example, if we have a word $CAAAGGGC$ that has GC weight of 5, the permutation produces a new word $ACCCTTTA$ that has a GC weight of 3.

   It is obvious that all of these permutations convert the pair of G and C into the pair of A and T, therefore exchange the weight of GC with the weight of AT in a word. Thus, it is observed that when a word with even length has the GC weight of $\frac{n}{2}$, this permutation still meets the restriction of fixed GC weight.

4. All of the remaining permutations are not valid for the fixed GC–content constraint, for any length and GC weight. There are 3 subcases to consider:

(a) First consider those permutations that maintain exactly one character unchanged. Those are

  i. $[A, -, -, -]$: $[A, G, T, C]$, $[A, T, G, C]$;

  ii. $[-, C, -, -]$: $[G, C, T, A]$, $[T, C, A, G]$;

  iii. $[-, -, G, -]$: $[C, T, G, A]$, $[T, A, G, C]$;

  iv. $[-, -, -, T]$: $[C, G, A, T]$, $[G, A, C, T]$.

  Consider $[A, -, -, -]$ which consists of $[A, G, T, C]$ and $[A, T, G, C]$. In each of these, A remains unchanged while T becomes C or G. Let us take $[A, G, T, C]$ to illustrate how the GC weight is affected. For example, if we have a word $AACCCGTT$, we obtain a new word $AAGGGTCC$ after this permutation has been applied. It is obvious that $AAGGGTCC$ no longer keeps the GC weight of $\frac{n}{2}$ as $AACCCGTT$ does. The same situation occurs for permutation $[A, T, G, C]$. Similarly, the remaining cases of $[-, C, -, -]$, $[-, -, G, -]$ and $[-, -, -, T]$ also allow the GC weight to change.

  Thus, in all of these cases, there are some words for which the GC weight will change when the permutation is applied.

(b) Now consider those permutations for which exactly two characters are unchanged, and that are not already covered in case 2 above. Those are

  i. $[A, C, -, -] : [A, C, T, G]$

  ii. $[A, -, G, -] : [A, T, C, G]$

  iii. $[-, -, G, T] : [C, A, G, T]$

  iv. $[-, C, -, T] : [G, C, A, T]$

In all of the above cases, either T or A is unchanged and either C or G is unchanged, while the remaining symbols are exchanged with each other. For instance, if we apply $[A, C, T, G]$ on a word $AACCCGTT$, we can obtain a new word $AACCCTGG$ which violates the fixed GC weight, since the total number of Gs and Cs of $AACCCTGG$ does not remain the same as $AACCCGTT$. Similarly, the remaining cases of $[A, -, G, -]$, $[-, -, G, T]$ and $[-, C, -, T]$ also allow the GC weight to change.

Thus, there are some words for which the GC weight will change when the permutation is applied.

(c) Finally, consider those permutations for which all of the characters are changed, and which are not covered by case 2 and case 3 above. These permutations are $[C, G, T, A]$, $[G, T, C, A]$, $[T, A, C, G]$ and $[T, G, A, C]$. In each case, the GC weight can not be fixed if the permutation is applied.

Let us take the permutation $[C, G, T, A]$ as an example. It changes character A into character C, character C into character G, character G into character T and character T into character A. If we apply it on a word $AACCCGTT$ of GC weight 4, then this yields a new word $CCGGGTAA$ containing a GC weight of 5. Similarly, the remaining permutations of $[G, T, A, C]$, $[T, A, C, G]$ and $[T, G, A, C]$ also allow the GC weight to change.

Thus, there are some words for which the GC weight will change when the permutation is applied.

From observing all of the above permutations in $Sym(\mathcal{D}^o)$, it is clear that the following permutations, denoted as θ_i where $i = 1, 2, 3$, preserve the GC weight of every DNA word:

1. $\theta_1 : (CG)$;

2. $\theta_2 : (AT)$;

3. $\theta_3 : (CG)(AT)$.

In other words, $\theta_i \in Sym(\mathcal{D})$ where $i = 1, 2, 3$ for a general DNA word.

When a DNA word is of even length n, with a GC weight of $\frac{n}{2}$, the following permutations θ_i where $i = 4, 5, 6, 7$ also preserve the GC weight in the new word obtained:

1. $\theta_4 : (AC)(GT)$;

2. $\theta_5 : (AG)(CT)$;

3. $\theta_6 : (ACTG)$;

4. $\theta_7 : (AGTC)$.

So $\theta_i \in Sym(\mathcal{D})$ where $i = 1, 2, ..., 7$ holds for even length DNA words with GC weight $\frac{n}{2}$, which is the main problem considered in our thesis.

As is known, the set of permutations that preserve a set system is a permutation group called the automorphism group of the set system. If (u, v) is an edge of the graph $\mathcal{G} = (V, E)$, and α is a permutation of V, then define $\alpha((u, v)) = (\alpha(u), \alpha(v))$. A permutation α of V is an automorphism of $\mathcal{G} = (V, E)$ if $\alpha((u, v)) \in E$ whenever $(u, v) \in E$. In other words, the automorphism preserves the vertex set and the adjacency of vertices. The automorphism group of $\mathcal{G} = (V, E)$ is the set of all permutations on V that are automorphisms of $\mathcal{G}$.

In the next section, it is proven that $Sym(\mathcal{D})$ is an automorphism group under the operation of composition.

Therefore, we rewrite the definition of $Sym(\mathcal{D})$ as follows:

1. $\sigma_1 : (CG)$;

2. $\sigma_2 : (AT)$;

and their composition meet the fixed GC–content constraint for DNA words. Moreover,

1. $\sigma_1 : (CG)$;

2. $\sigma_2 : (AT)$;

3. $\sigma_3 : (AC)(GT)$;

4. $\sigma_4 : (ACTG)$;

and their composition meet the fixed GC–content constraint when DNA words are of even length and GC weight $\frac{n}{2}$.

3.3 Definition of Equivalent DNA Codes

If we apply any $\sigma \in Sym(\mathcal{D})$ simultaneously to every character of every word in $\mathcal{D}^n$, then we can obtain a permutation of $\mathcal{D}^n$, denoted as σ^*.

In the next few paragraphs, we show that the set of all σ^* on $\mathcal{G}$ is contained in the automorphism group of $\mathcal{G}$, denoted by $Aut(\mathcal{G})$, since an automorphism of a graph preserves the adjacency of words and the distance between words in this graph.

Let $x, y \in \mathcal{D}^n$ be any two valid adjacent words, $x = x_1x_2...x_k...x_n$ and $y = y_1y_2...y_k...y_n$ with $x_i = y_i$ for $0 \leq i \leq n$ except $x_k \neq y_k$ for some k.

First of all, from Lemma 2 in [17], it is proved that if we apply any permutation in $Sym(\mathcal{D}^o)$, the permutation of the graph is an automorphism which preserves the distance between words and the adjacency of words. In our graph, we have the edge set $E_{x,y} = \{(x, y) : \min(d_{edit}(x, y), d_{edit}(x, y^{RC})) = 1\}$. Since the distance between words is measured by edit distance, both $d_{edit}(x, y)$ and $d_{edit}(x, y^{RC})$ are unchanged after we apply any permutation in $Sym(\mathcal{D})$. Thus, $E_{x,y}$ is the same as it was before permutation. So, for any σ^* on graph $\mathcal{G}$, the minimum distance of any two vertices is preserved.

Secondly, it is obvious that if we apply $\sigma \in Sym(\mathcal{D})$ where $Sym(\mathcal{D})$ is the subset of $Sym(\mathcal{D}^o)$ to all characters in x and y, then these two words are still adjacent. Thus, any σ^* on graph $\mathcal{G}$ preserves the adjacency of words.

In all, we conclude that the set of σ^* are elements of $Aut(\mathcal{G})$. Furthermore, the combination of any $\sigma_1^*, \sigma_2^* \in Aut(\mathcal{G})$ is also an element of $Aut(\mathcal{G})$.

In [17], it is proved that reversing all the words in an edit code simultaneously, denoted by γ, is also an automorphism, since the minimum distance between any two words is not changed and the adjacency of words remains exactly the same. It follows that $\gamma \in Aut(G)$.

Definition 1 *In general, we can define two DNA codes $\mathcal{C}_1$ and $\mathcal{C}_2$ over the alphabet set $\{A, C, G, T\}$ to be* equivalent *if one can be obtained from the other by any combination of the following:*

1. $\sigma_1 : (GC)$;

2. $\sigma_2 : (AT)$;

3. σ_3 : *reverse all words simultaneously.*

Definition 2 *Furthermore, for two DNA codes $\mathcal{C}_1$ and $\mathcal{C}_2$ over the alphabet set $\{A, C, G, T\}$,* *with even length and GC weight* $w = \frac{n}{2}$*,* C_1 *and* C_2 *are equivalent if one can be obtained from the other by any combination of the following:*

1. $\sigma_1 : (GC)$;
2. $\sigma_2 : (AT)$;
3. $\sigma_3 : (AC)(GT)$;
4. $\sigma_4 : (ACTG)$;
5. σ_5 : *reverse all words simultaneously.*

3.4 Properties of Equivalent DNA Codes

Let C be a DNA code with length n and minimum distance d, which also contains the total number of Gs and Cs given by w. This is denoted as a $(n, d, w)_4$ code. Define C_i to be the $(n-1, d, w)_4$ subcode obtained by selecting all codewords of C containing the symbol i at the first column, where $i = \{A, C, G, T\}$.

Theorem 1 *Every DNA code* C *with length of* n*, minimum distance of* d *and GC weight of* w *where* $0 \leq w \leq n$*, is equivalent to some code* C" *with* $|C^{"}_A| \geq |C^{"}_T|$ *and* $|C^{"}_C| \geq |C^{"}_G|$.

Proof: In C, if $C_A < C_T$, then apply σ_2 in Definition 1 to obtain an equivalent code C' with $C'_A > C'_T$, otherwise let $C' = C$. Next, if $C'_C < C'_G$, then apply σ_1 in Definition 1 to obtain an equivalent code C" with $C^{"}_C > C^{"}_G$, or else let $C" = C'$. $\square$

Theorem 2 *Every DNA code* C *with even length* n *and a fixed GC weight* $w = \frac{n}{2}$*, is equivalent to some code* C" *with* $|C^{"}_A| \geq |C^{"}_C| \geq |C^{"}_G|$ *and* $|C^{"}_A| \geq |C^{"}_T|$.

Proof: For the particular case when n is even and $w = \lfloor \frac{n}{2} \rfloor$,

1. If $|\mathcal{C}_A|$ in $\mathcal{C}$ is the largest $\mathcal{C}_i$, we check whether $|\mathcal{C}_C| \geq |\mathcal{C}_G|$ in $\mathcal{C}$. If not, we apply $\sigma_1 = (GC)$ in Definition 2 that changes all Gs with Cs in $\mathcal{C}$. Therefore, we obtain a code $\mathcal{C}'$ with $|\mathcal{C}'_A| \geq |\mathcal{C}'_C| \geq |\mathcal{C}'_G|$.

2. If $|\mathcal{C}_C|$ in $\mathcal{C}$ is the largest $\mathcal{C}_i$, then apply $\sigma_3 = (AC)(GT)$ in Definition 2 that exchanges all Cs with As and Gs with Ts in $\mathcal{C}$, thus we obtain a new code $\mathcal{C}'$ such that $|\mathcal{C}'_A|$ is the largest of all. Next, if $|\mathcal{C}'_G| > |\mathcal{C}'_C|$ in $\mathcal{C}'$, then apply $\sigma_1 = (GC)$ that changes all Gs with Cs in $\mathcal{C}'$. Thus, we obtain a code $\mathcal{C}''$ with $|\mathcal{C}''_A| \geq |\mathcal{C}''_C| \geq |\mathcal{C}''_G|$.

3. If $|\mathcal{C}_G|$ in $\mathcal{C}$ is the largest $\mathcal{C}_i$, then apply the permutation $\sigma_4 = (ACTG)$ in Definition 2, and thus we obtain a new code $\mathcal{C}'$ such that $|\mathcal{C}'_A|$ is the largest of all. Next, if $|\mathcal{C}'_G| > |\mathcal{C}'_C|$ in $\mathcal{C}'$, then apply $\sigma_1 = (GC)$ that changes all Gs with Cs in $\mathcal{C}'$. Then, we obtain a code $\mathcal{C}''$ with $|\mathcal{C}''_A| \geq |\mathcal{C}''_C| \geq |\mathcal{C}''_G|$.

4. If $|\mathcal{C}_T|$ in $\mathcal{C}$ is the largest $\mathcal{C}_i$, then apply the permutation $\sigma_2 = (AT)$ in Definition 2 that exchanges all As with Ts, and thus we can obtain a DNA code $\mathcal{C}'$ such that $|\mathcal{C}'_A|$ is the largest of all. Next, if $|\mathcal{C}'_G| > |\mathcal{C}'_C|$ in $\mathcal{C}'$, then apply $\sigma_1 = (GC)$ that changes all Gs with Cs in $\mathcal{C}'$. Thus, we obtain a code $\mathcal{C}''$ with $|\mathcal{C}''_A| \geq |\mathcal{C}''_C| \geq |\mathcal{C}''_G|$.

Combined with Theorem 1, we obtain the relationship that $\mathcal{C}$ is equivalent to some code $\mathcal{C}''$ with $|\mathcal{C}''_A| \geq |\mathcal{C}''_C| \geq |\mathcal{C}''_G|$ and $|\mathcal{C}''_A| \geq |\mathcal{C}''_T|$. $\square$

Chapter 4

Properties of DNA Codes

In this chapter, we explore the properties of DNA codes. We focus on analyzing DNA codes for some special parameter values.

Define $E_4(n, d)$ to be the maximum size of quaternary codes with length n and minimum edit distance d. An $(n, M, d)_4$ edit code is said to be *optimal* if the number of codewords M is equal to $E_4(n, d)$. First define $E_4^{GC}(n, d, w)$ to be the maximum size of DNA codes with length n, minimum edit distance d and fixed GC weight w. Define $E_4^{RC,GC}(n, d, w)$ to be the maximum size of DNA codes with length n, minimum edit distance d and fixed GC weight w, that satisfy the reverse–complement constraint.

4.1 Theorems of Optimal DNA codes

Several observations are investigated and we discuss the details in the following paragraphs.

First, we examine the relationship between $E_4^{RC,GC}(n, d, w)$ and $E_4^{GC}(n, d, w)$. A similar proof of the relationship of $A_4^{RC,GC}(n, d, w)$ with $A_4^{GC}(n, d, w)$ for DNA codes over Hamming distance can be found in [20].

Theorem 3 *For* $1 \leq d \leq n$ *and* $0 \leq w \leq n$, $E_4^{RC,GC}(n,d,w) \leq \frac{1}{2} \cdot E_4^{GC}(n,d,w)$.

Proof: Let C be an optimal code of length n, minimum edit distance d and fixed GC weight w, that satisfies the reverse–complement constraint. Assume that the size of C is $|C|$. Let $C^{RC} = \{x^{RC} | x \in C\}$. It is obvious that C^{RC} is also an optimal code of size $|C|$, since the edit distance between every two words in C^{RC} remains the same when reverse–complement is applied, and the GC weight remains the same as well. Let $C' = C \bigcup C^{RC}$, then we have C' is a $(n,d,w)_4$ code of length n, minimum edit distance d and fixed GC weight w. Clearly, $C \bigcap C^{RC} = \Phi$. Thus, we have $|C'| = 2|C|$. In addition, we always have $|C'| \leq E_4^{GC}(n,d,w)$. It follows $E_4^{RC,GC}(n,d,w) = |C| = \frac{1}{2}|C'| \leq \frac{1}{2} \cdot E_4^{GC}(n,d,w)$. $\square$

Theorem 4 (Codes with $n < d$)
If $d > n$ *and* $0 \leq w \leq n$, *then* $E_4^{RC,GC}(n,d,w) = 1$.

Proof: Codes with $n < d$ are trivial in that such a code always contains exactly one word. By the definition of the edit metric, the maximum distance between any two words of length n is at most n. Thus for any w, there exists no more than one word in a $(n,d,w)_4$ DNA code if $d > n$. $\square$

A similar proof for DNA codes over Hamming distance with $n < d$ can be found in [17].

Theorem 5 (Codes with $n = d$)

$$E_4^{GC,RC}(n,n,w) = \begin{cases} 2 & \text{if } w = n/2 \text{ and } n \text{ even} \\ 1 & \text{if } w = \lfloor n/2 \rfloor \text{and } n \text{ odd} \end{cases}$$

Proof: Let $E_4^{GC}(n,n,w)$ be the maximum size of $(n,n,w)_4$ codes that meet the fixed GC–content constraint. Since the distance between every two codewords must be at least n, it implies that no two codewords agree in any coordinate. Therefore, there can be at most four codewords for $(n,n,w)_4$ codes by the pigeonhole principle. Hence, we have $E_4^{GC}(n,n,w) \leq 4$ for any n and $0 \leq w \leq n$. Combined with Theorem 3, we obtain $E_4^{GC,RC}(n,n,w) \leq 2$.

First consider the case when n is even and $w = \frac{n}{2}$. We can always construct a code $\{A^wC^w, C^wA^w, G^wT^w, T^wG^w\}$ to be an $(n,n,w)_4$ code that satisfies the GC–content constraint. It is clear that A^wC^w and G^wT^w, and C^wA^w and T^wG^w, are reverse–complements of each other. Furthermore, it is clear that the distance between any two codewords is $n = 2w$. Thus, we can have $\{A^wC^w,\ C^wA^w\}$, $\{G^wT^w,\ T^wG^w\}$, $\{A^wC^w,\ T^wG^w\}$ and $\{C^wA^w\ \ G^wT^w\}$ as $(n,n,w)_4$ codes that satisfy both the fixed GC–content constraint and the reverse–complement constraint. Therefore, for n even, $E_4^{GC}(n,n,w) = 2$.

Now consider the case when n is odd and $w = \lfloor n/2 \rfloor$. When $n = 2w+1$, we construct a GC–content restricted $(n,n,w)_4$ code as $\{X, Y, X^{RC}, Y^{RC}\}$, where $X = \{x_1x_2...x_w...x_n\}$ and $Y = \{y_1y_2...y_w...y_n\}$. We assume that x_w is either A or T, so that $\overline{x_w}$ is the remaining T or A, and y_w can be either C or G, so that $\overline{y_w}$ is the remaining G or C. By the pigeonhole principle, the total number of Gs and Cs in those 4 words is equal to $\frac{1}{2} \times (4n) = 2n$. This contradicts the given $w = \lfloor n/2 \rfloor$. Then, combining $E_4^{GC}(n,n,w) < 4$ with Theorem 3, we have $E_4^{GC,RC}(n,n,w) < 2$. However, we can always have codes that contain a single codeword which has $w = \lfloor n/2 \rfloor$. Thus, $E_4^{GC,RC}(n,n,w) = 1$. □

A similar proof for DNA codes over Hamming distance with $n = d$ can be found in [20].

Theorem 6 (Codes with $d = 1$)

$$E_4^{GC,RC}(n,1,w) = \begin{cases} \frac{1}{2}(\binom{n}{w}2^n - \binom{n/2}{w/2}2^{n/2}) & \text{if } n \text{ is even and } w \text{ is even} \\ \frac{1}{2}\binom{n}{w}2^n & \text{if } n \text{ is odd or } w \text{ is odd} \end{cases}$$

Proof: For $n > 0$, with $0 \le d \le n$ and $0 \le w \le n$, we partition $\{A, C, G, T\}$ into $P_0 = \{G, C\}$ and $P_1 = \{A, T\}$. Denote a codeword x as $\{x_1, x_2, ..., x_n\}$ and its reverse–complement word x^{RC} as $\{\overline{x_n}, \overline{x_{n-1}}, ..., \overline{x_1}\}$.

First of all, we apply the fixed GC weight w on all codewords of length n. There are in total $\binom{n}{w}2^n$ codewords that have GC weight w because w of n positions are chosen for P_0 and for each position, there are two symbols to be chosen from.

Then, we apply the reverse–complement constraint on those codewords. It is noted that applying reverse–complement on a word x does not change the GC weight of x, since we just flip x end to end, and replace all the Gs with Cs and As with Ts in x. There are three cases to be considered:

1. For n even and w even, there are $\binom{n/2}{w/2}2^{n/2}$ codewords that are unchanged, called self–complementary words. If $x = x^{RC}$, then $\overline{x_i} = x_{n-i+1}$. Thus, if the first half of x is determined, then the last half of x is also determined. It is inferred that the first half and the last half of x must both contain $\frac{w}{2}$ Gs and Cs. So, there are in total $\binom{n/2}{w/2}2^{n/2}$ possibilities for the first half with length $\frac{n}{2}$. Then, we remove

these $\begin{pmatrix} n/2 \\ w/2 \end{pmatrix} 2^{n/2}$ self–complementary codewords from the set of $\begin{pmatrix} n \\ w \end{pmatrix} 2^n$ total codewords, thus obtaining a set that contains no self–complementary codewords. Since no codeword can be in a code with its reverse–complement, we can have only half of the remaining $\begin{pmatrix} n \\ w \end{pmatrix} 2^n - \begin{pmatrix} n/2 \\ w/2 \end{pmatrix} 2^{n/2}$ codewords in a $(n, 1, w)$ DNA code.

2. For n even and w odd, there does not exist any codeword that is unchanged by reverse–complement. Because if $x = x^{RC}$, then $\overline{x_i} = x_{n-i+1}$ for $0 \leq i \leq n$. This contradicts the condition that w is odd. Again, no codeword can be in a code with its reverse–complement, so we have only half of $\begin{pmatrix} n \\ w \end{pmatrix} 2^n$ words in an $(n, 1, w)$ DNA code.

3. For n odd, there does not exist any codeword that is unchanged by reverse–complement. Since if $n = 2k+1$, $x_{k+1} \neq \overline{x_{k+1}}$. So, there are $\frac{1}{2} \begin{pmatrix} n \\ w \end{pmatrix} 2^n$ codewords in a $(n, 1, w)$ DNA code. □

Theorem 7 (codes with $d = n = 1$)

For all codes with $n = 1$, $E_4^{RC,GC}(1, 1) = 1$.

Proof: If $n = 1$, then $\lfloor \frac{n}{2} \rfloor = 0$. From Figure 3.1, it is shown that the vertex $\{A/T\}$ has weight $w = 0$ for code length 1, and that these are valid words, while G and C are both invalid codewords. So, a single codeword either A or T can be contained in a $(1, 1, 0)_4$ DNA code. □

4.1.1 Binary Reverse–Complement Codes

In this section, we introduce a new code, the binary reverse–complement code. The study of this type of code is motivated by constructing DNA codes from certain specific binary codes. The study of constructing DNA codes over Hamming distance from Hamming binary codes has been discussed in [23]. We apply a similar approach to generate DNA codes over edit distance with distance of 2.

A binary reverse–complement code $C_2^{RC}(n,d)$ is a binary edit code over alphabet set $\{0,1\}$, that satisfies the reverse–complement constraint. If we have $x = \{x_1, x_2, ..., x_n\}$, the reverse–complement of x, denoted by x^{RC}, is defined to be $x^{RC} = \{\overline{x_n}, \overline{x_{n-1}}, ..., \overline{x_1}\}$ where $0^{RC} = 1$ and $1^{RC} = 0$. A word x for which $x = x^{RC}$ is called *self–complementary*. The maximum size of $C_2^{RC}(n,d)$ is denoted by $E_2^{RC}(n,2)$.

Proposition 1 *For n even, all 2^n binary words of length n can be partitioned into 2 subsets such that each contains 2^{n-1} words, with the following properties:*

1. *Any two words x,y in the same subset satisfy $d_{edit}(x,y) \geq 2$.*
2. *Any word x must be in the same subset as its reverse–complement x^{RC}.*
3. *All words for which $x = x^{RC}$ are in the same subset.*

Proof: The proof is by induction. When the length is 2, the words are partitioned into $S_1^2 = \{01, 10\}$ and $S_2^2 = \{00, 11\}$, with S_1^2 containing all self complementary words.

When the length n is even, assume that the words are partitioned into S_1^n and S_2^n, each with 2^{n-1} words, satisfying the above three properties and with S_1^n containing all self complementary words.

For length $n+2$, we define the construction of subsets as

$S_1^{n+2} = S_1^n \cdot S_1^2 \cup S_2^n \cdot S_2^2$ and

$S_2^{n+2} = S_1^n \cdot S_2^2 \cup S_2^n \cdot S_1^2$

where $A \cdot B = \{pwq | w \in A, pq \in B, |p| = |q| = 1\}$. We can verify that the partition S_1^{n+2} and S_2^{n+2} also has all of the three properties, with S_1^{n+2} and S_2^{n+2} each containing 2^{n+1} words and S_1^{n+2} containing all self complementary words. □

Observation 1 *For length n, the words in S_2^n are either all odd weight or all even weight.*

Proof: This is proved by induction. When the length is 2, we have $S_1^2 = \{01, 10\}$ and $S_2^2 = \{00, 11\}$, such that S_2^1 contains all odd weight words and S_2^2 contains all even weight words.

When the length n is even, suppose that S_n^1 contains all odd weight words and S_n^2 contains all even weight words.

As defined above, the construction of subsets is

$S_1^{n+2} = S_1^n \cdot S_1^2 \cup S_2^n \cdot S_2^2$ and

$S_2^{n+2} = S_1^n \cdot S_2^2 \cup S_2^n \cdot S_1^2$

Clearly, we have that S_1^{n+2} contains all even weight words and S_2^{n+2} contains all odd weight words. Because for S_2^{n+2}, words generated by $S_1^n \cdot S_2^2$ are of odd weight since words in S_1^n are all odd weight and words in S_2^2 are all even weight; words created by $S_2^n \cdot S_1^2$ are of odd weight since words in S_2^n are all even weight and words in S_1^2 are all odd weight. Similarly, we can verify that words in S_1^{n+2} are all even weight words.

Similarly, for length n, if S_n^1 contains all even weight words and S_n^2 contains all odd weight words, then S_2^{n+2} are all even weight words and S_1^{n+2} are all odd weight words. □

Theorem 8 *For $1 \leq d \leq n$ and $0 \leq w \leq n$, $E_2^{RC}(n, d) \leq \frac{1}{2} \cdot E_2(n, d)$.*

Proof: Let $\mathcal{C}$ be an optimal binary code of length n and minimum edit distance d that satisfies the reverse–complement constraint. Assume that the size of $\mathcal{C}$ is $|\mathcal{C}|$. Let $\mathcal{C}^{RC} = \{x^{RC} | x \in \mathcal{C}\}$. Clearly, $\mathcal{C}^{RC}$ is also an optimal code binary with size of $|\mathcal{C}|$, since the edit distance between every two words in $\mathcal{C}^{RC}$ remains the same when reverse–complement is applied. Let $\mathcal{C}' = \mathcal{C} \bigcup \mathcal{C}^{RC}$, so that $\mathcal{C}'$ is a binary code of length n and minimum edit distance d. Clearly, $\mathcal{C} \bigcap \mathcal{C}^{RC} = \Phi$. In addition, $|\mathcal{C}'| = |\mathcal{C} \bigcup \mathcal{C}^{RC}| = |2\mathcal{C}| \leq E_2(n,d)$. It follows that $E_2^{RC}(n,d) = |\mathcal{C}| \leq \frac{1}{2} \cdot E_2(n,d)$. □

Theorem 9 *For n even, $E_2^{RC}(n,2) = 2^{n-2}$.*

Proof: The proof is based on proposition 1.

For a partition of the binary words of length n, we choose the subset that does not contain any self complementary words, and drop either x or x^{RC} for each word x repeatedly, thus obtaining a set of words of size 2^{n-2} meeting the edit distance and reverse–complement constraint for $d = 2$. Thus, we have $E_2^{RC}(n,2) \geq 2^{n-2}$.

Theorem 3 in [17] states that $E_q(n,q) = q^{n-1}$ when $d = 2$ and combined with Theorem 8, we have $E_2^{RC}(n,2) \leq \frac{1}{2} \cdot 2^{n-1}$. It follows that $E_2^{RC}(n,2) = 2^{n-2}$ for n even. □

Theorem 10 *For $0 \leq w \leq n$ and n even, $E_4^{GC,RC}(n,2,w) = \begin{pmatrix} n \\ w \end{pmatrix} 2^{n-2}$.*

Proof: Theorem 17 in [20] shows that for DNA codes over Hamming distance and $0 \leq w \leq n$, we have $A_4^{GC,RC}(n,2,w) \leq \begin{pmatrix} n \\ w \end{pmatrix} 2^{n-2}$. Furthermore, it is proved in Theorem 1 in [17] that $E_q(n,d) \leq A_q(n,d)$. Thus we have $E_4^{GC,RC}(n,2,w) \leq A_4^{GC,RC}(n,2,w) \leq \begin{pmatrix} n \\ w \end{pmatrix} 2^{n-2}$. Therefore, $E_4^{GC,RC}(n,2,w) \leq \begin{pmatrix} n \\ w \end{pmatrix} 2^{n-2}$.

Now we shall show $E_4^{GC,RC}(n,2,w) \geq \begin{pmatrix} n \\ w \end{pmatrix} 2^{n-2}$.

By using the template–map strategy in [2], we can construct a DNA code from two binary codes. Two binary codes are assigned respectively for the constraints on the DNA code. One specifies the GC weight of the DNA code, and is called the template. The other one, called the map, specifies the distance between any two words in the DNA code.

Define a template c_t to be an $(n,2,w)_2$ code over the alphabet $\{A,C\}$ such that for all $x \in c_t$, we have $W_{GC}(x) = w$ for $0 \leq w \leq n$. Define a map c_m to be an $(n,2)_2$ edit metric code over the alphabet $\{0,1\}$ that meets the reverse–complement constraint, discussed in Theorem 8. The product of these two codes produces a quaternary code, denoted by $S = \{x \cdot y | x \in c_t, y \in c_m\}$. The mapping is defined as follows:

1. $x_i = A$ and $y_i = 1$, $x_i \cdot y_i = A$.

2. $x_i = A$ and $y_i = 0$, $x_i \cdot y_i = T$.

3. $x_i = C$ and $y_i = 1$, $x_i \cdot y_i = C$.

4. $x_i = C$ and $y_i = 0$, $x_i \cdot y_i = G$.

where x_i is the ith character in x and y_i is the ith character in y.

For example, in Figure 4.1, we present the construction of 24 words of length of 4 with minimum distance of 2 and GC weight of 2, satisfying the reverse–complement constraint.

The upper bound of c_t is $A_2(n,2,w) = \begin{pmatrix} n \\ w \end{pmatrix}$ since each word chooses w characters to be A among n characters. The upper bound of c_m is $E_2^{RC}(n,2) = 2^{n-2}$ from Theorem 8. Thus, we obtain a quaternary code S satisfying $W_{GC}(s_i) = w$, $d_{edit}(s_i, s_j) \geq 2$ and $d_{edit}(s_i, s_j^{RC}) \geq 2$ where $s_i, s_j \in S$ and $0 \leq i,j \leq |S|$. The size of the code S, $|S|$, is equal

Template	**Map**
AACC	0001
CAAC	0010
CCAA	0100
ACAC	1000
ACCA	
CACA	

$$\longrightarrow \quad \begin{array}{r} \text{AACC} \\ \times \ 0\,0\,0\,1 \\ \hline \text{TTGC} \end{array}$$

Figure 4.1: Template–Map construction for a $(4, 2, 2)_4$ DNA code

to $\left(\begin{array}{c} n \\ \\ w \end{array}\right) 2^{n-2}$.

It is obvious that every codeword in S has GC weight w since there are w Cs in every x in c_t and the mapping does not change the total number of Gs and Cs in any codeword. Moreover, any pair of codewords s_i, s_j in S must satisfy $d_{edit}(s_i, s_j) \geq 2$ and $d_{edit}(s_i, s_j^{RC}) \geq 2$.

We now show that for any two codewords s_i, s_j in S, we have $d_{edit}(s_i, s_j) \geq 2$ and $d_{edit}(s_i, s_j^{RC}) \geq 2$.

Assume that the size of the template code c_t is P and the size of the map code c_m is Q. We denote an element of S as $s_{ij} = x_i \cdot y_j$, $x_i \in c_t$ and $y_j \in c_m$. For any two codewords x_i, $x_{i'}$ in c_t, we have $d_H(x_i, x_{i'}) \geq 2$. For any two codewords y_i, $y_{i'}$ in c_m, we have $d_{edit}(y_j, y_{j'}) \geq 2$.

First we shall show that $d_{edit}(s_{ij}, s_{i'j'}) \geq 2$ for any two distinct words s_{ij}, $s_{i'j'}$ in S.

1. If $x_i = x_{i'}$ and $y_j \neq y_{j'}$, which implies that the same template is applied to different maps, then the distance between resulting words s_{ij} and $s_{ij'}$ is always greater than or equal to 2 since the coordinates that differ in y_j and $y_{j'}$ are still different in s_{ij} and

$s_{ij'}$ and $d_{edit}(y_j, y_{j'}) \geq 2$.

2. If $x_i \neq x_{i'}$ and $y_j = y_{j'}$, which implies that the same map is applied to different templates, then $d_{edit}(s_{ij}, s_{i'j}) \geq 2$ since there are at least two coordinates that differ between x_i and $x_{i'}$ because all words in c_t have the same weight.

3. If $x_i \neq x_{i'}$ and $y_j \neq y_{j'}$, which implies that different maps x_i and $x_{i'}$ are applied to different templates y_j and $y_{j'}$ respectively, then we shall show that we also have $d_{edit}(s_{ij}, s_{i'j'}) \geq 2$.

   Assume that for some s_{ij} and $s_{i'j'}$ such that $d_{edit}(s_{ij}, s_{i'j'}) < 2$ where $i \neq i'$ and $j \neq j'$. It is always true that the distance between any two words can not be negative. Thus, there are 2 cases to consider.

   (a) $d_{edit}(s_{ij}, s_{i'j'}) = 0$, which implies that $s_{ij} = s_{i'j'}$. So, we must have $x_i = x_{i'}$ and $y_j = y_{j'}$, which contradicts with $i \neq i'$ and $j \neq j'$.

   (b) $d_{edit}(s_{ij}, s_{i'j'}) = 1$, which implies that there exists a position k in s_{ij} and $s_{i'j'}$ that is different where $0 \leq k < n$. So, we must have either x_i and $x_{i'}$ that only differ in position k or y_j and $y_{j'}$ that only differ in position k, or both. This contradicts with $d_{edit}(x_i, x_{i'}) \geq 2$ and $d_H(y_j, y_{j'}) \geq 2$.

   From the above, we conclude that $d_{edit}(s_{ij}, s_{i'j'}) \geq 2$ for $x_i \neq x_{i'}$ and $y_j \neq y_{j'}$.

We now shall show that $d_{edit}(s_i, s_j^{RC}) \geq 2$ for any two words s_i, s_j in S.

When choosing the words from S_2^n, we can choose one of y, y^{RC} to be an element of c_m. Without loss of generality, we can choose the one with lower weight to be y.

We construct $c_m' = \{z : z \in c_m \bigcup c_m^{RC}\}$ where $c_m^{RC} = \{y^{RC} : y \in c_m\}$. Since c_m is a binary reverse–complement code, the distance between any two distinct words in

c_m' is at least 2. Thus, the distance of any two distinct words in S' is at least 2, where $S' = \{x \cdot z : x \in c_t, z \in c_m'\}$. The proof is similar to that above.

1. Let $s_{ij} = x_i \cdot z_j$, $x_i \in c_t$ and $z_j \in c_{m'}$ we can construct s_{ij}^{RC} by product x_i^R with z_j^{RC} such that x_i^R is also in c_t and $d_{edit}(z_j, z_j^{RC}) \geq 2$.

    (a) If $x_{i'} = x_i^{RC}$ where $i' = i$ and $z_{j'} = z_j^{RC}$ where $j' = |Q+j|$, which implies that the same template is applied to different maps, then we have $d_{edit}(s_{ij}, s_{ij}^{RC}) \geq 2$ since $d_{edit}(z_j, z_j^{RC}) \geq 2$.

    (b) If $x_{i'} \neq x_i^{RC}$ where $i' = i$ and $z_{j'} = z_j^{RC}$ where $j' = |Q+j|$, which implies that different templates are applied to different maps, then we have $d_{edit}(s_{ij}, s_{ij}^{RC}) \geq 2$. This proof is similar to the above case 3.

2. There are 2 cases to consider when we compare the distance between a word and the reverse–complement of a distinct word.

    (a) If $x_{i'} = x_i^{RC}$ where $i' = i$ and $z_{j'} \neq z_j^{RC}$ where $j' \neq |Q|+j$, which implies that the same template is applied to different maps, then we have $d_{edit}(s_{ij}, s_{ij'}^{RC}) \geq 2$ since $d_{edit}(z_j, z_{j'}^{RC}) \geq 2$.

    (b) If $x_{i'} \neq x_i^{RC}$ where $i' = i$ and $z_{j'} \neq z_j^{RC}$ where $j' \neq |Q+j|$, which implies that different templates are applied to different maps, then we have $d_{edit}(s_{ij}, s_{i'j'}^{RC}) \geq 2$. This proof is similar to the above case 3.

From the above, we conclude that $d_{edit}(s_{ij}, s_{i'j'}^{RC}) \geq 2$ for any two words in S.

From the above construction, we have $E_4^{GC,RC}(n, 2, w) \geq A_2(n, 2, w) \cdot E_2^{RC}(n, 2) = \begin{pmatrix} n \\ w \end{pmatrix} 2^{n-2}$. Therefore, $E_4^{GC,RC}(n, 2, w) = \begin{pmatrix} n \\ w \end{pmatrix} 2^{n-2}$. □

4.2 Lower and Upper Bounds

The lower bound for $E_4^{GC,RC}(n, d, w)$ may be established by constructing $(n, d, w)_4$ DNA codes and keeping track of those of largest size. As discussed in Chapter 2, mathematical constructions, as well as various heuristics have been developed to obtain lower bound for conventional error–correcting codes and edit metric codes. In the following chapters, algorithms are designed for generating DNA codes that are as large as possible, to obtain good lower bounds for $E_4^{GC,RC}(n, d, w)$.

We consider establishing upper bounds for $(n, d, w)_4$ DNA codes. One way to establish an upper bound of $E_4^{GC,RC}(n, d, w)$ is by using Theorem 3. However, the construction of $(n, d, w)_4$ codes with fixed *GC*–content itself is very difficult.

For both (n, d) Hamming distance codes and (n, d) edit codes with no restriction, we have $E_4(n, d) \leq 4 \cdot E_4(n-1, d)$ [17], since for any of those codes, we can obtain an equivalent code $\mathcal{C}'$ with $|\mathcal{C}'_A| \geq |\mathcal{C}'_C| \geq |\mathcal{C}'_G| \geq |\mathcal{C}'_T|$. However, this method of obtaining upper bounds can not be used for DNA codes with the reverse–complement constraint and the fixed *GC*–content constraint. For every DNA code $\mathcal{C}$ of given n, d and w in general, it is equivalent to some code $\mathcal{C}'$ such that $|\mathcal{C}'_A| \geq |\mathcal{C}'_T|$ and $|\mathcal{C}'_C| \geq |\mathcal{C}'_G|$. Thereby, whether $|\mathcal{C}'_A| \geq |\mathcal{C}'_C|$ is not known. For every DNA code $\mathcal{C}$ with even n and $w = \frac{n}{2}$, we have some code $\mathcal{C}"$ with $|\mathcal{C}"_A| \geq |\mathcal{C}"_C| \geq |\mathcal{C}"_G|$ and $|\mathcal{C}"_A| \geq |\mathcal{C}"_T|$, as its equivalent code. Then, we consider those words that start with A to be the largest subcode of the $(n, d, w)_4$ code. However, it is not guaranteed that we can obtain an $(n-1, d, w)_4$ code by removing the first column from this subcode, since the reverse–complement constraint might be violated.

Chapter 5

Algorithms

In this chapter, we introduce two deterministic algorithms that are designed for solving the problem of finding a large set of DNA codewords. We include Conway's Lexicode algorithm, as well as exhaustive search and some optimizations. We also present tables of bounds on DNA codes.

5.1 Conway's Lexicode Algorithm

Conway's Lexicode algorithm, proposed by Conway and Sloane in 1986 [12], used the greedy construction to generate a class of error–correcting codes, called lexicodes. Authors in [3] used this algorithm to construct edit codes with given minimum distance d and length n. We implemented this algorithm in the thesis, and report the lower bounds on the value of $E_4^{RC,GC}(n, d, w)$ it provided.

5.1.1 Outline of Conway's Lexicode

We used the Conway's Lexicode algorithm mentioned above to generate DNA codes with given parameter values, that satisfy both the reverse–complement constraint and the fixed GC–content constraint.

Conway's Lexicode Algorithm:

1. Given n, d, w, find all words of length n that satisfy $d_{edit}(x, x^{RC}) \geq d$ and $W_{GC}(x) = w$ and place them into a candidate list $L(\mathcal{C})$ in lexicographical order.

2. Initialize an empty set $\mathcal{C}$ of words for storing the target code.

3. Scan $L(\mathcal{C})$ in lexicographic order, selecting a word $z \in L(\mathcal{C})$ and placing it in $\mathcal{C}$ if and only if $d_{edit}(x, z) \geq d$ and $d_{edit}(x, z^{RC}) \geq d$ for all words $x \in \mathcal{C}$.

4. Apply Step 3 repeatedly until the end of $L(\mathcal{C})$ is reached.

5.1.2 Table of Bounds on DNA Codes by Conway's Lexicode

Obviously, the order in which the words are chosen is determined ahead of time in this algorithm. Each time a word is selected, the words that are 'incompatible' with it, violating either the edit distance constraint or the reverse–complement constraint, have to be excluded from the candidate words set. Consequently, the words chosen later are strongly dependent on the earlier chosen words in this code. Thus, in most cases, we can not expect the codes to be optimal. But it allows the construction of DNA codes in an immediate way and obtains reasonable lower bounds.

Table 5.1.2 represents the table of lower bounds on DNA codes obtained by Conway's lexicode algorithm, with $2 \leqslant n \leqslant 9$, and $1 \leqslant d \leqslant 9$.

Table 5.1: Bounds on DNA Codes by Conway's Lexicode Algorithm

n \ d	1	2	3	4	5	6	7	8	9
2	4	2	-	-	-	-	-	-	-
3	12	5	1	-	-	-	-	-	-
4	44	20	4	2	-	-	-	-	-
5	160	68	9	3	1	-	-	-	-
6	640	320	31	7	2	2	-	-	-
7	2240	954	91	18	5	2	1	-	-
8	8912	4432	294	54	12	5	2	2	-
9	32256	13672	912	150	29	9	3	2	1

5.2 Exhaustive Search Algorithm

We wish to obtain values of $E_4^{RC,GC}(n, d, w)$ for small length n, minimum distance d. In addition, we consider fixed GC weight $w = \lfloor \frac{n}{2} \rfloor$ so that the DNA strings can be maintained stable in a proper temperature range in most practical applications, as mentioned in Chapter 1.

From Chapter 4, we already know the exact values for $E_4(n, d)$ when $d = n$ or $d = 1$, as well as for the trivial cases when $n = d = 1$ and $n < d$ from Chapter 3. We also derived the exact bounds of codes when $d = 2$ and length is even. The values of $E_4(n, d)$ for the above cases will be included in the following tables.

5.2.1 Outline of Exhaustive Search

In [17], the exhaustive search method has been utilized to create optimal edit codes with small parameter values. A similar approach is applied to solve our problem in which we consider the additional biological constraints, the reverse–complement constraint and the fixed GC–content constraint. The exact values for $E_4^{RC,GC}(n,d,w)$ with small values of parameters can be found. The exhaustive search algorithm uses backtracking and is outlined below.

Exhaustive Search Algorithm:

1. Select all words of length n that satisfy both $d_{edit}(x, x^{RC}) \geq d$ and $W_{GC}(x) = w$ where $w = \lfloor \frac{n}{2} \rfloor$ and put them into a candidate list $L(\mathcal{C})$ in lexicographic order.

2. For each x in $L(\mathcal{C})$, create a base code $\mathcal{C}$ containing x and a lexicographically ordered candidate list $L'(\mathcal{C})$ that consists of all remaining words $y \in L(\mathcal{C})$ that are compatible with x, i.e. $d_{edit}(x, y) \geq d$ and $d_{edit}(x, y^{RC}) \geq d$.

3. For each word $z \in L'(\mathcal{C})$, if z is compatible with all codewords in $\mathcal{C}$, we add z to $\mathcal{C}$, otherwise trim z from $L'(\mathcal{C})$. In other words, we can create an $(n, |\mathcal{C}| + 1, d)$ DNA code by appending one compatible word to base code $\mathcal{C}$, thereby producing $L'(\mathcal{C})$ base codes for each level of backtracking.

4. If no word remains in $L'(\mathcal{C})$ of a code, we return to the previous level, removing the most recently–chosen word $c_{current}$ and adding those words that are incompatible with $c_{current}$ but compatible with the remaining words in $\mathcal{C}$ back to $L'(\mathcal{C})$. Then, select the word in $L'(\mathcal{C})$ that appears later than the most recently–chosen word lexicographically.

5. We apply Steps 3 and 4 recursively until we return to level 0.

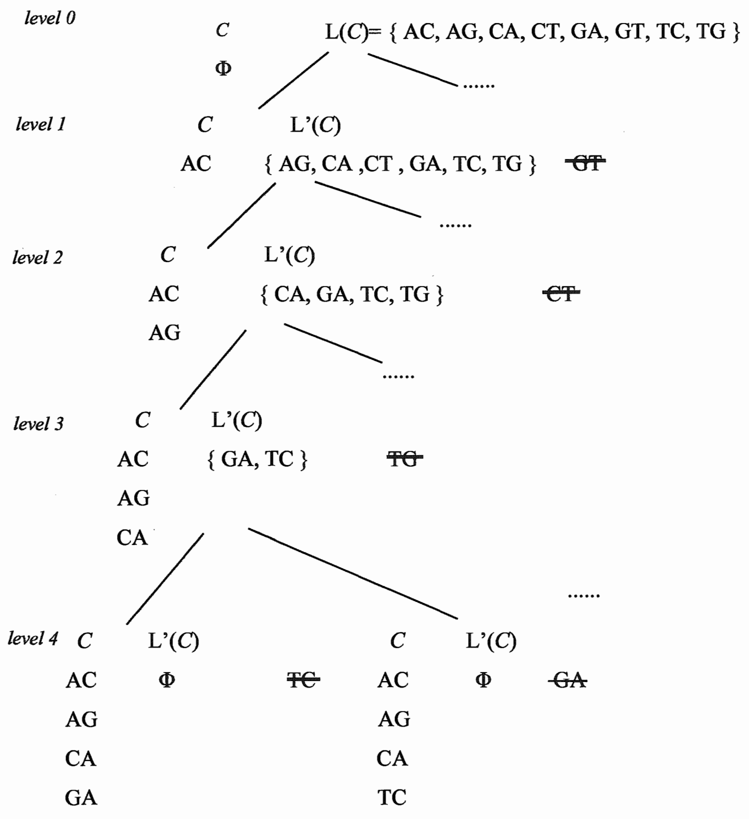

Figure 5.1: Exhaustive Search procedure for $(2, 1, 1)_4$ codes

Figure 5.1 shows how the exhaustive search performs to obtain $(2, 1, 1)_4$ codes. Firstly, we obtain the candidate list of all valid words of length 2 lexicographically (see Figure 3.1), as $L(\mathcal{C}) = \{AC, AG, CA, CT, GA, GT, TC, TG\}$. If we select word AC as the starting point of a base code, we have the candidate list at level 1 as $L'(\mathcal{C}) = \{AG, CA, CT, GA, TC, TG\}$, dropping GT which is incompatible with AC. For current code $\mathcal{C} = \{AC\}$, we select word AG in $L'(\mathcal{C})$, then obtain $\mathcal{C} = \{AC, AG\}$ and $L'(\mathcal{C}) = \{CA, GA, TC, TG\}$ at level 2. Then, we repeated this process until $L'(\mathcal{C}) = \phi$, producing a new $(2, 1, 1)_4$ code

$\mathcal{C} = \{AC, AG, CA, GA\}$. Then, it returns to the previous level and tries the next word in $L'(\mathcal{C})$, obtaining $\mathcal{C} = \{AC, AG, CA, TC\}$. At each level, all words are eventually considered, thus it generates all the $(2, 1, 1)_4$ codes.

5.2.2 Optimizations of Exhaustive Search

We notice that in the process of the backtrack search, we often need to calculate the distance between pairs of words, which costs a large amount of running time. It is beneficial that we precompute the information about distances between all possible pairs of words to save computation time. Hence we generate all possible words of length n, and store the information in a compatibility matrix, in which entry $(i, j) = 1$ if word i and word j are compatible with each other, and entry $(i, j) = 0$ otherwise. However, using a compatibility matrix of all pairs of possible words demands a large amount of memory, so that it can be only applied for codes with relatively small parameter values.

For step 4, we only consider words in $L'(\mathcal{C})$ that appear later than the most recently–chosen word in lexicographical order. Otherwise, we might create duplicate codes which choose the same set of words but in a different order.

During the search, we save l, the current size of the largest code found so far. Suppose that $\mathcal{C}$ is our current code at level k of the search. If $|\mathcal{C}|+$ the remainder of $L'(\mathcal{C})$ is less than l, then we can prune this branch. This will prevent the generation of codes smaller than the optimal code and reduce the running time. For example, we showed that we obtain a $(2, 1, 1)_4$ code of size of 4 in the early search in Figure 5.1. In a later search as shown in Figure 5.2, we select word GT as the starting point of a base code, but only two words remain in the candidate set $L'(\mathcal{C})$. Thus, this branch of backtracking can never generate a code that has a size of at least of 4.

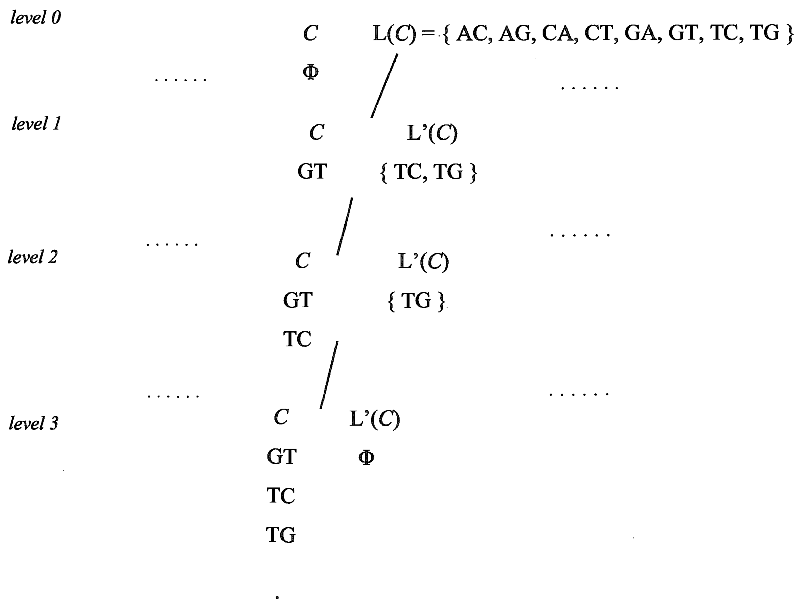

Figure 5.2: Pruning method of Exhaustive Search for $(2, 1, 1)_4$ codes

From Theorem 1 in Chapter 3, for any DNA code $\mathcal{C}$, there always exists an equivalent code $\mathcal{C}''$ with $|\mathcal{C}''_A| \geq |\mathcal{C}''_T|$ and $|\mathcal{C}''_C| \geq |\mathcal{C}''_G|$. Once $\mathcal{C}$ is found, we want to prune those branches of the search tree that lead to a code that is equivalent to $\mathcal{C}$. This will prevent the generation of equivalent codes and speed up the search. Clearly, we have $|\mathcal{C}''_A| + |\mathcal{C}''_C| \geq |\mathcal{C}''_G| + |\mathcal{C}''_T|$ for $\mathcal{C}''$. For a level k, if $|\mathcal{C}''_A| + |\mathcal{C}''_C| < |\mathcal{C}''_G| + |\mathcal{C}''_T|$, then we can prune this branch.

Combined with the discussion of above paragraphs, it is indicated that the total number of $|\mathcal{C}''_A|$ and $|\mathcal{C}''_C|$ must be greater than half of the size of the largest code found so far, as l, so it follows that $|\mathcal{C}''_A| + |\mathcal{C}''_C| \geq \frac{l}{2}$. For a level k, if $|\mathcal{C}''_A| + |\mathcal{C}''_C| < \frac{l}{2}$, then prune this branch.

The above pruning methods are used to optimize the search in our program, avoiding unnecessary searches for those codes that will be less than optimal and for equivalent codes, which will indeed save a lot of computational time.

5.2.3 Table of Sizes of Optimal DNA Codes by Exhaustive Search

Here we present the table of bounds on optimal DNA codes obtained by the exhaustive search methods from above sections and Chapter 4, for length $2 \leqslant n \leqslant 9$, and minimum distance $1 \leqslant d \leqslant 9$.

Table 5.2: Bounds on DNA Codes by Exhaustive Search Algorithm

n \ d	1	2	3	4	5	6	7	8	9
2	4	2	-	-	-	-	-	-	-
3	12	5	1	-	-	-	-	-	-
4	44	24	6	2	-	-	-	-	-
5	160	70	13	3	1	-	-	-	-
6	640	320	34*	10	3	2	-	-	-
7	2240	954*	94*	22*	7	2	1	-	-
8	8912	4480	297*	57*	16*	7	2	2	-
9	32256	13672*	914*	150*	34*	11	5	2	1

Table 5.2 shows the results for DNA codes for $n \leqslant 9$ and $d \leqslant 9$. Entries denoted by an asterisk are cases for which the exhaustive search did not finish within two weeks. Thus these values provide lower bounds for $E_4^{RC,GC}(n,d,w)$ with corresponding length n, minimum distance d and GC weight w. The remaining values in this table are exact values

for $E_4^{RC,GC}(n,d,w)$, respectively for length n, minimum distance d and GC weight w.

Chapter 6

Genetic Algorithm

In this chapter, we first introduce the genetic algorithms in general. Then, a genetic algorithm, named Randomized GA, is designed to solve our problem and tables of lower bounds on DNA codes found by this algorithm are given.

6.1 Background of Genetic Algorithm

Genetic Algorithms (GAs) were invented by John Holland in the 1960s and developed by Holland and his students. GAs are developed so that the mechanisms of natural adaption are imported to computer systems. Based on simulating natural systems necessary for evolution, GAs use a kind of 'natural selection' together with genetic–inspired operators of crossover, mutation and reproduction to solve optimization problems.

Genetic Algorithms are said to be an adaptive heuristic, since they employ an 'intelligent' exploration within the search space, by using historical information and genetic mechanisms to search for better solutions. A more precise definition of *Genetic Algorithm* is given by John R. Koza [21],

" GA is a probabilistic search algorithm that iteratively transforms a set (called a population) of mathematical objects (typically fixed–length binary character strings), each with an associated fitness value, into a new population of offspring objects using the Darwinian principle of natural selection and using operations that are patterned after naturally occurring genetic operations, such as crossover and mutation."

6.2 The Operators of Genetic Algorithm

In this section, we introduce the basic concepts of GA operators.

Representation: A *chromosome*, is encoded to represent a solution to the problem. The chromosome uniquely identifies an *individual*, which is a point in the search space.

Fitness Function: The fitness function is used to evaluate how 'good' a solution is. The fitness values are then used in a process of natural selection to choose the potential solutions to be continued to the next generation.

Selection Operator: Selection determines which individuals survive and possibly mate and mutate in the next generation. It is noted that selection does not eliminate all 'unfit' chromosomes, since they might mutate to something useful after recessing for several generations. There are various types of selection mechanisms, such as fitness–proportionate selection, rank selection, tournament selection and roulette wheel selection.

Crossover Operator: Crossover combines two parent chromosomes from a generation to produce new chromosomes for the next generation by swapping certain genetic material between two parent chromosomes.

Mutation Operator: Mutation introduces a certain mount of randomness to the process, which allows offspring chromosomes to evolve in new directions.

Replacement Operator: There are two replacement schemes: simple replacement and steady–state replacement. Simple replacement replaces the entire population with the new population at each generation, while the steady–state replacement preserves a fixed number of 'good' individuals from the old population to the new population at each generation.

6.3 Parameters of Genetic Algorithm

In this section, we introduce several GA parameters such as crossover rate, mutation rate and population size. We also analyze how those parameters affect the search.

Crossover Rate: The crossover probability determines how often crossover performs. Crossover is designed for the purpose that the new chromosomes will inherit good parts of old chromosomes and thus possibly create more 'fit' offspring. If there is no crossover, the offspring are exact copies of their parents. If crossover probability is high, then most of the offspring in new generation will be made by crossover. If it is low, then most of the offspring in new generation will be the exact copies of chromosomes from the old population.

Mutation Rate: The mutation probability determines how often a chromosome will be mutated. Mutation is designed to prevent GA from falling into local extrema. Mutation should not occur very often, since otherwise the GA will change to random search. If mutation probability is 100%, then a chromosome is always changed. If it is 0%, then nothing is changed.

Population size: The population size indicates how many chromosomes are in a population. If there are too few chromosomes, the GA has a few possibilities to perform crossover and only a small part of the search space is explored. On the other hand, if there are too many chromosomes, the computation time increases.

6.4 Outline of General Genetic Algorithm

We present the scheme of a standard genetic algorithm as follows:

```
i = 0
Initialize population $P_0$
Evaluate the fitness of $P_0$
while (Not done) Test for termination criterion (time, fitness, etc)
Begin
    Selection($P_i$)          select a population for reproduction
    Crossover($P_i$)          recombine the selected parents
    Mutation($P_i$)           perturb the mated population stochastically
    Replacement($P_i$)        replace new population with old population
    i = i + 1
End
```

The GA finishes when a solution is found, or it meets certain criteria such as reaching a set time limit, or reaching a predetermined number of generations, etc.

6.5 Genetic Algorithm for DNA Codes

In this section, we present a genetic algorithm to solve the problem of finding good DNA codes, called Randomized Genetic Algorithm, RGA for short. The algorithm incorporates a heuristic method named the repair operator with the standard genetic algorithm. It is accomplished by experimenting with different GA parameters (crossover rate, mutation rate and population size), as well as some specified GA operators (crossover, mutation, selection, etc.). We analyze the impact of these parameters on the GA behavior while running the GA system and try to find suitable parameters for our GA.

6.5.1 Parameter Sets

The Randomized GA is performed using the following parameter sets:

1. Number of generation: $(100, 200, 300, ...)$
2. Population size: $(50, 100, 200)$
3. Crossover rate: $0\% - 100\%(0\%, 10\%, 50\%, 80\%, 100\%)$
4. Mutation rate: $0\% - 100\%(100\%, 80\%, 50\%, 10\%, 0\%)$
5. Number of elitism(if elitism is applied): $(0, 5, 10)$

It is noted that the GA always converges after a number of generations, thus it should be set to an appropriate value for runs. Meanwhile, a proper value of population size should also be found. If the population size is too small, it limits the diversity of individuals in a population, however it costs too much time for computation at each generation if it is too large.

6.5.2 Representation

Given length n, distance d and GC weight w, find all words of length n that satisfy $d_{edit}(x, x^{RC}) \geq d$ and $W_{GC}(x) = w$ and place them into a candidate set C_{can} in lexicographical order. The representation of a chromosome, a solution, is a string of integers such that each integer is the index of a codeword in the set C_{can}. In other words, a chromosome consists of a set of codewords represented by their indices into C_{can}. In addition, all codewords in this set are compatible with each other. Recall that two words x, y are compatible with each other if $W_{GC}(x) = W_{GC}(y) = w$ and $min(d_{edit}(x, y), d_{edit}(x, y^{RC})) \geq d$. Thus the chromosome is a valid code, as it consists of a set of compatible codewords.

6.5.3 Fitness Function

Since we wish to find codes that are as large as possible, the fitness of a chromosome calculates the number of codewords in a chromosome, which is also called the size of the code.

6.5.4 Initialization

Given the fixed size of the population S_{pop}, the initialization process creates S_{pop} chromosomes. Each code (chromosome) c_i where $0 \leq i < S_{pop}$, is initialized as an empty set. For a given code c_i, we select one word x from set C_{can} randomly, adding x to c_i if it is compatible with all the words in c_i, otherwise removing it from C_{can}. We perform this repeatedly until there are no words remaining in C_{can}. The indices of the corresponding codewords form one chromosome. Thus, we can obtain S_{pop} randomized chromosomes as the starting points.

6.5.5 Selection operator

In selection, we apply k–tournament selection since it is demonstrated as a good selection mechanism for most GA problems. The idea of tournament selection is that at any given generation, we choose k chromosomes from that generation, then select the one with the best fitness among those k chromosomes for reproduction. In our program, we employ 2–tournament selection as our selection operator.

6.5.6 Repair Operator

The repair operator simply adds all remaining words from the current candidate set C_{can}, which are compatible with words in current code, to the code in random order. This extends the code as to be as large as possible. Figure 6.1 shows a repair process for an incomplete $(4, 3)$ DNA code.

C		C_R
2233	Repair	***0322***
3132	$\rightarrow$	***1331***
		2233
		2310
		3132

Figure 6.1: Repair of a $(4, 3)$ DNA code

6.5.7 Crossover Operator

The crossover we used in Randomized GA is union crossover. Union crossover 'merges' two indices of chosen chromosomes, called parents, into one set of indices. This set of indices maps to a set of words in C_{can}, denoted by C_{merge}. When a merge occurs, we add the indices of one code into C_{merge}, then append any index from the other code if it is not replicated in C_{merge}. It is noted that the new set of indices could possibly map to an invalid code since we might use a code in which its words are incompatible with some of the words of the other code. Then, we consider C_{merge} as a candidate set and construct a new code C_{cross} by selecting words from C_{merge} in random order. It is not guaranteed that the crossover operator can generate a new code which has a larger size than its parent codes. Figure 6.2 shows how two random $(4,3,2)_4$ codes 'cross over' to build a new $(4,3,2)_4$ code, with a size of 2.

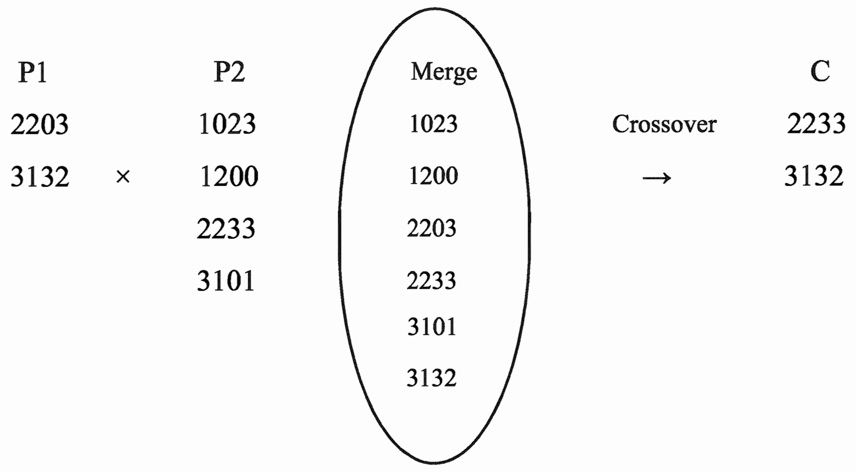

Figure 6.2: Crossover of two $(4,3)$ DNA codes

Afterwards, the repair operator is applied on C_{cross}, which extends C_{cross} by appending the remaining compatible words from the set $C_{can} - C_{cross}$ randomly. We 'repair' this code to extend it to as large a valid code as possible, finally obtaining a new $(4,3,2)$ DNA code that has a larger size than both its parent codes, as shown in Figure 6.1.

We should also note that we might obtain a new code smaller than the parent codes even after this repair operator. However, this 'downhill' move would potentially allow us to further explore the search space.

6.5.8 Mutation Operator

The mutation mechanism we apply is random mutation. We randomly select a word in the parent code, and remove it from the codeword set. Then we apply the repair operator to extend the partial code to be as large as possible.

6.5.9 Replacement Operator

There are two replacement operators we adopted for different tests. One is simply passing all the chromosomes as the new population to the next generation. Another one is called *elitism*, which preserves the best m chromosomes from the last generation, while replacing the remaining worst chromosomes.

6.6 Examples of Experiments

Experiments are performed to search for good GA parameters of $(7,3,3)_{dna}$ codes, using the setup parameter sets.

Figure 6.3 shows the performance of the Randomized GA with and without elitism. As illustrated, both GAs start to converge after about 20 generations during the runs. Hence, we set up the number of generations to 100 for the remaining runs on $(7,3,3)_4$ DNA codes.

The following tables show the empirical results of the Randomized GA by examining GA parameter sets of crossover rate $(0.0, 0.1, 0.5, 0.8, 1.0)$, mutation rate $(0.0, 0.1, 0.5, 0.8, 1.0)$ and population size $(50, 100, 200)$.

Figure 6.4 shows how different population sizes impact on the Randomized GA. Our algorithm performs better for a population size of 200. It is noted that we can not guarantee

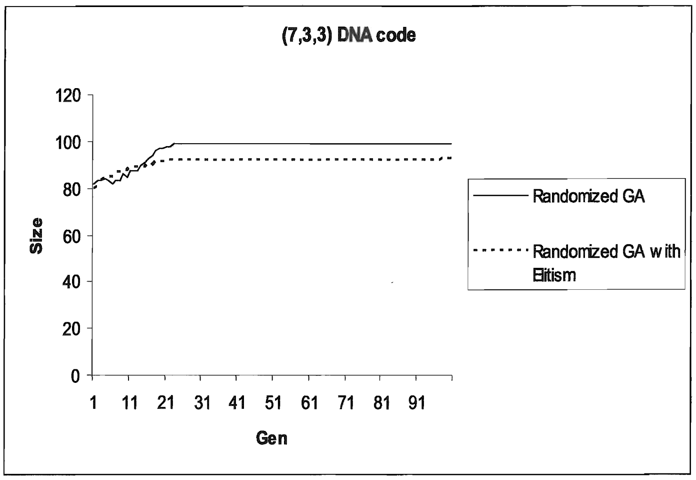

Figure 6.3: Randomized GA on $(7, 3, 3)_4$ DNA codes

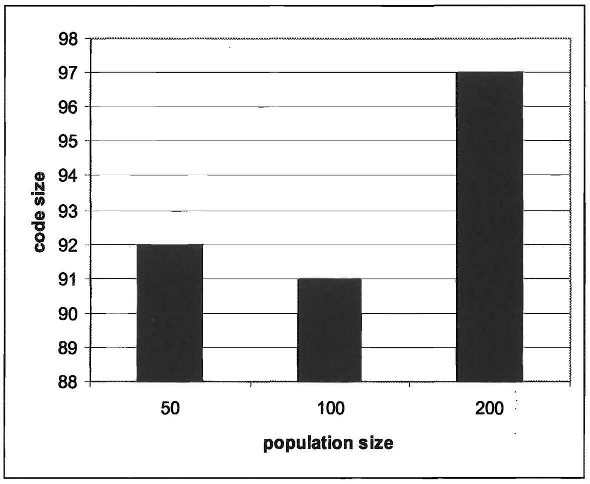

Figure 6.4: Different population sizes on $(7, 3, 3)_4$ DNA codes

Table 6.1: Sizes of DNA Codes by RGA of Population Size 50

	Crossover Rate	Mutation Rate	Best size
1	0.0	1.0	96
2	0.1	0.8	93
3	0.5	0.5	92
4	0.8	0.1	92
5	1.0	0.0	89

Table 6.2: Sizes of DNA Codes by RGA of Population Size 100

	Crossover Rate	Mutation Rate	Best size
1	0.0	1.0	91
2	0.1	0.8	90
3	0.5	0.5	95
4	0.8	0.1	91
5	1.0	0.0	100

that this population size is suitable for codes with different n and d. For a small code, runs with large population size slow down the GA, while the population lacks diversity if the population size is too small.

From the above tables, the parameter set of population size = 200, crossover rate = 1.0 and mutation rate = 0.0 performs best for $(7, 3, 3)_4$ DNA codes.

Table 6.3: Sizes of DNA Codes by RGA of Population Size 200

	Crossover Rate	Mutation Rate	Best size
1	0.0	1.0	97
2	0.1	0.8	95
3	0.5	0.5	92
4	0.8	0.1	97
5	1.0	0.0	101

6.7 Tables of Sizes of DNA Codes by Randomized Genetic Algorithm

For a better observation of the convergence of the best codes through generations, we set the maximum number of generations as 300 and population size as 200 in later experiments. We present tables of the size of the best codes found with different parameter sets of $(0.0, 1.0)$, $(1.0, 0.0)$ and $(0.9, 0.1)$, for pairs of crossover rate and mutation rate respectively. In following three tables, the sizes of codes with $d = 1$ are obtained by using Theorem 6 in Chapter 4.

Table 6.4 shows the bounds on DNA codes for $n \leqslant 9$ and $d \leqslant 9$ by Randomized GA under the crossover rate of 100% and mutation rate of 0%.

Table 6.5 shows the bounds on DNA codes for $n \leqslant 9$ and $d \leqslant 9$ by Randomized GA under the crossover rate of 0% and mutation rate of 100%.

Table 6.6 shows the bounds on DNA codes for $n \leqslant 9$ and $d \leqslant 9$ by Randomized GA under the crossover rate of 100% and mutation rate of 0%, while Elitism is also applied at

Table 6.4: Bounds on DNA Codes by RGA under $R_c = 1.0$, $R_m = 0.0$

n \ d	1	2	3	4	5	6	7	8	9
2	4	2	-	-	-	-	-	-	-
3	12	5	1	-	-	-	-	-	-
4	44	20	4	2	-	-	-	-	-
5	160	74	13	3	1	-	-	-	-
6	640	320	34	9	3	2	-	-	-
7	2240	1090	101	22	7	2	1	-	-
8	8912	4480	324	61	14	5	2	2	-
9	32256	15558	939	161	33	10	4	2	1

each generation.

The above three tables present the lower bounds on sizes of DNA codes for $n \leqslant 9$ and $d \leqslant 9$ under different crossover rates and mutation rates. For most cases, Randomized GA with crossover rate 1.00 and mutation rate 0.00 outperformed the others, except for the $(8,6,4)_4$ code. We also found that for codes for which d is close to n, e.g. $(6,5,3)_4$ and $(7,5,3)_4$, it is difficult for the GA to find the optimal values through many generations, while the exhaustive search can produce the exact values for the bounds within a few minutes. Even so, the GA still hits more than 95% of the sizes of codes obtained by exhaustive search in those cases.

The best known sizes of the codes from the above three tables are chosen to establish a table of lower bounds of DNA codes for the Randomized GA.

Table 6.5: Bounds on DNA Codes by RGA under $R_c = 0.0$, $R_m = 1.0$

n \ d	1	2	3	4	5	6	7	8	9
2	4	2	-	-	-	-	-	-	-
3	12	5	1	-	-	-	-	-	-
4	44	20	4	2	-	-	-	-	-
5	160	74	13	3	1	-	-	-	-
6	640	298	34	9	3	2	-	-	-
7	2240	985	99	21	6	2	1	-	-
8	8912	4419	306	58	15	6	2	2	-
9	32256	8378	869	154	32	10	5	2	1

Table 6.6: Bounds on DNA Codes by RGA with Elitism under $R_c = 0.9$, $R_m = 0.1$

n \ d	1	2	3	4	5	6	7	8	9
2	4	2	-	-	-	-	-	-	-
3	12	5	1	-	-	-	-	-	-
4	44	20	4	2	-	-	-	-	-
5	160	74	12	3	1	-	-	-	-
6	640	320	32	9	3	2	-	-	-
7	2240	1074	93	21	6	2	1	-	-
8	8912	4260	302	54	13	5	2	2	-
9	32256	14821	855	142	31	10	4	2	1

Chapter 7

Conclusion

In this thesis, we discussed several approaches to solving the problem of finding good DNA codes for given parameters n, d and $w = \lfloor \frac{n}{2} \rfloor$. We observed the DNA word space with certain biological constraints and explored the theory underlying DNA codes for some specific parameter values.

Three algorithms were used to obtain lower bounds for DNA codes. Exact values of bounds on codes for relatively small parameter values were obtained by using exhaustive search. However, exhaustive search, the slowest of the three algorithms, is not feasible for a large range of parameter values since it is computationally expensive. Conway's lexicode algorithm is the fastest algorithm, simply scanning codes in lexicographic order. However, the lower bound obtained by Conway's lexicode is weaker than the other two algorithms. The Randomized Genetic Algorithm proposed in Chapter 6, utilizing a random greedy algorithm as a local search during crossover and mutation, outperformed the other two within a reasonable computation time. For $(7, 3, 3)_4$ DNA codes, it took about half an hour to complete on a 2.2GHz CPU Linux XC 3.1 machine by using Randomized Genetic

Algorithm while the exhaustive search did not finish in more than two weeks. It is important to remember that we precomputed the compatibility of all possible candidate words to avoid repetitive comparison of distance between pairs of words, which requires a large amount of memory. This limits our search to a relatively small range of parameter values. Thus, a tradeoff must be made for both computation time and memory capacity.

Table 7.1: Best–known Bounds on DNA Codes

n \ d	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
2	**4**	**2**	**1**	**1**	**1**	**1**	**1**	**1**	**1**
3	**12**	5	**1**	**1**	**1**	**1**	**1**	**1**	**1**
4	**44**	**24**	6	**2**	**1**	**1**	**1**	**1**	**1**
5	**160**	74^g	13	3	**1**	**1**	**1**	**1**	**1**
6	**640**	**320**	$34^{e,g}$	10	3	**2**	**1**	**1**	**1**
7	**2240**	1090^g	101^g	$22^{e,g}$	7	2	**1**	**1**	1
8	**8912**	**4480**	324^g	61^g	16^e	7	2	**2**	**1**
9	**32256**	15558^g	939^g	161^g	34^e	11^e	$5^{e,g}$	2	**1**

Table 7 presents the best–known values for DNA codes for $1 \leq n \leq 9$ and $1 \leq d \leq 9$. The entries marked in bond, are the exact values for the DNA codes of specified n, d and w by theorems in Chapter 4. For cases for which the exact value is unknown, we provide their lower bounds. Entries denoted by the superscript g are the cases for which the best known bound was found by Randomized GA. Entries denoted by the superscript e are the cases for which the best known bound was generated by exhaustive search, however there was not sufficient time to complete the runs. The remaining values are exact values of bounds on DNA codes by exhaustive search. It is noted that all the bounds found by Conway's lexicode algorithm are always smaller than or equal to those obtained by exhaustive search or Randomized GA.

It is of interest to compare our algorithms with those discussed in previous research of other people. The greedy closure genetic algorithm in [17] is demonstrated as an efficient algorithm for finding good edit codes. Our future work involves applying this strategy to search for good DNA codes.

As shown in Table 7, for most codes, we do not have exact values of the maximum size but rather lower bounds. So, an obvious future work is to improve the tables of bounds where possible. Furthermore, we will work on codes for a larger range of parameter values. One way is to try to compress the compatibility matrix so that larger parameter values maybe considered.

Another interesting task is to further examine the structure of the DNA word space and explore the theory underlying DNA codes for other specific parameter values. It is possible to consider constructing codes with even length and codes with odd length in different manners, since codes with even length contain self–complementary words while those with odd length do not.

In addition to the constraints considered in this thesis, we would concern about other issues which might be important in practical applications, such as forbidden subwords constraint, consecutive–bases constraint, frame–shift constraint and the secondary structure constraint, see Chapter 1. We might focus on a more accurate model of melting temperature, mentioned in [23]. In applications, it is also of interest to consider codes which have variable lengths.

Bibliography

[1] L. M. Adleman. Molecular computation of solutions to combinatorial problems. *Science*, 266(5187):1021–1024, November 1994.

[2] M. Arita. Writing information into dna. *Aspects of Molecular Computing*, 2950:211 – 222, 2004.

[3] D. Ashlock, L. Guo, and F. 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.

[4] S. Baker, R. Flack, and S. Houghten. Optimal variable–length insertion–deletion correcting codes and edit metric codes. *Congressus Numerantium*, 186:65–80, 2007.

[5] A. Ben-Dor, R. Karp, B. Schwikowski, and Z. Yakhini. Universal dna tag systems: A combinatorial design scheme. In *Journal of Computational Biology*, volume 7, pages 503–519. ACM Special Interest Group on Algorithms and Computation Theory, 2000.

[6] A. Brenneman and A. E. Condon. Strand design for biomolecular computation. *Theoretical Computer Science*, 287:39–58, 2002.

[7] K. J. Breslauer, R. Frank, H. Blocker, and L. A. Marky. Predicting dna duplex stability from the base sequence. In *Proc Natl Acad Sci*, volume 83, pages 3746–3750, USA, 1986.

[8] A. T. Brouwer. Small binary codes: Table of general binary codes. website at: http://www.win.tue.nl/ aeb/codes/binary-1.html.

[9] A. T. Brouwer. Small quaternary codes: Table of general quaternary codes. website at: http://www.win.tue.nl/ aeb/codes/quaternary-1.html.

[10] A. T. Brouwer. Small ternary codes: Table of general ternary codes. website at: http://www.win.tue.nl/ aeb/codes/ternary-1.html.

[11] Y. M. Chee and L. San. Improved lower bounds for constant gc–content dna codes. *IEEE Transactions on Infromation Theory*, 54(1):391–394, 2008.

[12] J. H. Conway and N. J. A. Sloane. Lexicographic codes: error–correcting codes from game theory. In *IEEE Trans. Inform. Theory*, volume 32, page 337C348, 1986.

[13] U. Feldkamp, H. Rauhe, and W. Banzhaf. Software tools for dna sequence design. *Genetic Programming and Evolvable Machines*, 4(2):153–171, June 2003.

[14] P. Gaborit and O. D. King. Linear constructions for dna codes. *Theoretical Computer Science*, 334(1-3):99–113, 2005.

[15] M. Garzon, R. Deaton, P. Neathery, D. R . Franceschetti, and R. C. Murphy. A new metric for dna computing. In *Proceedings of the Second Genetic Programming Conference*, 1997.

[16] W. Haas and S. Houghten. A comparison of evolutionary algorithms for finding optimal error-correcting codes. In *Proceedings of the 3rd IASTED Conference on Computational Intelligence*, pages 64–70, CI 2007.

[17] S. Houghten, D. Ashlock, and J. Lenarz. Construction of optimal edit metric codes. In *Proceedings of the 2006 IEEE Workshop on Information Theory*, pages 259–263, 2006.

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

[19] D. Kim, I. Lee S. Shin, and B. Zhang. Nacst/seq: A sequence design system with multiobjective optimization. In *DNA8 Lecture Notes in Computer Science*, pages 242–251. SpringerVerlag, 2003.

[20] O. D. King. Bounds for dna codes with constant gc-content. *Journal of Combinatorics*, 10:13, 2003.

[21] John R. Koza. Genetic algorithm and genetic programming. website at: http://www.smi.stanford.edu/people/koza/.

[22] F. J. MacWilliams and N. J. A. Sloane. *Theory of Error Correcting Codes*. Elsevier Science Ltd, 1977.

[23] A. Marathe, A. E. Condon, and R. M. Corn. On combinatorial dna word design. *Journal of Computational Biology: a journal of computational molecular cell biology*, 8(3):201–219, 2001.

[24] K. M. McGuire and R. E. Sabin. Using a genetic algorithm to find good linear error-correcting codes. In *Symposium on Applied Computing Proceedings of the 1998 ACM symposium on Applied Computing*, pages 332 – 337. ACM New York, NY, USA, 1998.

[25] Q. Qiu, D. Burns, Q. Wu, and P. Mukre. Hybrid architecture for accelerating dna codeword library searching. In *IEEE Symposium on Computational Intelligence and Bioinformatics and Computational Biology*, pages 323–330, 2007.

[26] J. SantaLucia, H. T. Allawi, and P. A. Seneviratne. Improved nearest–neighbor parameters for predicting dna duplex stability. *Biochemistry*, 35(11):3555–3562, March 1996.

[27] M. Schena, D. Shalon, R. W. Davis, and P. O. Brown. Quantitative monitoring of gene expression patterns with a complementary dna microarray. *Science*, 270(5235):467–470, October 1995.

[28] N. C. Seeman and N. R. Kallenbach. Design of immobile nucleic acid junctions. *Biophysical Journal*, 44:201–209, 1983.

[29] C. E. Shannon. *A Mathematical Theory of Communication*. CSLI Publications, 1948.

[30] S. Shen, K. Wang, G. Hu, and S. Xia. On the alignment space and its applications. *Information Theory Workshop*, (9485065):165 – 169, 2006.

[31] D. D. Shoemaker, D. A. Lashkari, D. Morris, M. Mittman, and R. W. Davis. Quantitative phenotypic analysis of yeast deletion mutants using a highly parallel molecular bar-coring strategy. *Nature Genetics*, 16:450–456, December 1996.

[32] D. C. Tulpan and H. H. Hoos. Hybrid randomised neighbourhoods improve stochastic local search for dna code design. *Adavances in Aritificial Intelligence: 16th Conf. of the Canadian Society for Comuptaional Studies of Intelligence*, 2671, 2003.

[33] Wikipedia. Double helix. website at: http://en.wikipedia.org/wiki/Double_helix.

[34] Wikipedia. Edit distance. website at: http://en.wikipedia.org/wiki/Levenshtein_distance.