Delft University of Technology
Faculty of Electrical Engineering, Mathematics and Computer Science
Delft Institute of Applied Mathematics

Constrained Codes for DNA-Based Storage Systems

(Dutch title: Beperkte codes voor DNA-gebaseerde opslagsystemen)

Thesis submitted to the
Delft Institute of Applied Mathematics
in partial fulfillment of the requirements

for the degree

BACHELOR OF SCIENCE
in
APPLIED MATHEMATICS

by

C.J. van Leeuwen

Delft, Nederland
May 2020

BSc thesis APPLIED MATHEMATICS

"Constrained Codes for DNA-Based Storage Systems"
Dutch title: "Beperkte codes voor DNA-gebaseerde opslagsystemen"

C.J. van Leeuwen

Delft University of Technology

Thesis committee

Dr.ir. J.H. Weber (supervisor)

Dr. J.A.M. de Groot (supervisor)

Dr.ir. W.G.M Groenevelt

May, 2020 Delft

Preface

This thesis is the final part of the Bachelor program of Applied Mathematics at the Delft University of Technology. In high school my interest in mathematics started and the choice to study Applied Mathematics was easily made. In my first year Dr. J.A.M. de Groot gave a lecture in coding theory and in this lecture he explained how coding theory is used to enable the step from Long Playing (LP's) records to compact discs (CD's). This is where I first got interested in the subject and why I chose the elective course "Applied Algebra; Codes and Cryptography Systems", lectured by this same Dr. J.A.M. de Groot. Because I find the field of coding theory very interesting I decided do my Bachelor thesis in this field.
Dr.ir. J.H. Weber came to me with the idea to explore data storage using DNA. Before I started I never heard of the concept but after reading some literature I understood why it could be used to save the enormous amounts of data that is produced. A big advantage of using DNA over other classical methods of saving data is that DNA can store a lot of information on very little space. But there are some constraints that we need to take into account and in this thesis I will elaborate more on this. A lot of research has been done on the subject of saving data using DNA and the paper of Limbichaya et. al. [1] focuses especially on the constraints. For this thesis I mostly focused on this paper of Limbichaya et. al. with the title: "Family of Constrained Codes for Archival DNA Data Storage" published in October 2018.
Lastly I would like to thank my supervisors Dr.ir. J.H. Weber and Dr. J.A.M. de Groot for their guidance and support. They gave me a lot of feedback during the process of writing this thesis and were always happy to answer my questions. Finally I also want to thank my Bachelor committee for reviewing my thesis and presentation.

Lot van Leeuwen
Delft, May 2020

Abstract

Every day we produce an extremely high amount of data and a significant portion of this data is archival data. It is an important challenge to save this data in a cheap and environmentally friendly way. The current methods to save archival data are sufficient, but improvements can be made. Synthetic DNA based data storage is a great choice to do so. DNA is (roughly) made out of four nucleotides, Adine (A), Cytosine (C), Guanine (G) and Thymine (T). To save data using DNA the binary data is encoded to quaternary data and later DNA strands are created using this quaternary data. During the process of saving the data errors can occur. Previous research [3] has found that some errors can be prevented by taking two constraints into account. The no run-length constraint, which states that no DNA word can have two repeated symbols and GC-weight constraint, which states that every DNA word must have a fixed number of G and C nucleotides. These constraints reduce the number of quaternary data sequences that can be used to save data.

The aim of this thesis is to improve the lower bound on the maximum number of quaternary data sequences that satisfy the no run-length constraint, the fixed GC-weight and a minimum distance. Limbachiya et. al. [1] gave an algorithm to compute a set with a given minimum distance of quaternary data sequences that satisfies the constraints. The size of this set is the lower bound that we aim to improve. We do so by introducing two other algorithms that also compute a quaternary data sequences that satisfy the constraints and a minimum distance. The size of each computed set gives a lower bound on the maximum size. The lower bound computed with these two algorithms is always better or equal to the lower bound computed by Limbachiya et. al for certain parameters. When the minimum distance of a code is 2 we know this maximum size. The formula for this maximum size is given in this thesis together with a proof for this formula.

Contents

1 Introduction

This first chapter explains why DNA-based data storage could be a good method to store archival data, which part of this subject will be researched in this thesis and the organisation of this thesis.

1.1 Motivation

Nowadays an extremely high amount of data is produced every day. A significant portion of this data is archival data. Archival data is meant to be stored away because the data does not need to be accessible at every moment. Right now most of this data is stored on magnetic and optical media [2]. Storing the data in this way is sufficient but there are improvements possible. Storing data on magnetic and optical media has a density of about $100GB/mm^3$. This means that storing zettabytes of data takes up a lot of physical space [2]. Durability is also important for storing data. With the current techniques we can reach a durability of 30 years [2]. To store the world's archival data in a cheaper and environmentally friendly way it is important to improve storage density and durability.

Synthetic DNA based information storage is a solution for this problem, as explained in [3] DNA can store data up to 2 PB/gram, so 2 million GB/gram. The durability of DNA is also extremely good, S. Yazdi states about the subject: *"one can still recover the DNA of species extinct for more than 10,000 years"* [3]. Despite writing and reading DNA sequences is expensive, DNA data storage has the potential to be a good option for archival data storage [1]. In this thesis we focus on the sequences of nucleotides that eventually can be used to save archival data.

1.2 Thesis statement

The research of Limbachiya et. al. named "Family of Constrained Codes for Archival DNA Data storage" [1] draws our attention. This paper introduces two constraints on quaternary data sequences that are very important for archival data storage using DNA. Using these constraints a code is introduced. This code is called a DNA-d code and it consists of DNA codewords. The constraints are: a DNA codeword cannot have a run of the same nucleotides, it has a fixed number of G and C nucleotides and a code has a specific minimum distance. A DNA-d code has a size and therefore their exists a DNA-d code of maximum size. This maximum size is unknown in the paper of Limbachiya. To get a lower bound on this maximum size an algorithm is given that computes a DNA-d code. The size of this computed DNA-d code is the existing lower bound.

This thesis aims to answer the question: '*'Can the existing lower bound for the maximum size of a DNA-d be improved*". In this thesis two algorithms to compute a DNA-d code are given. These algorithms provide a better or equal lower bound on the maximum size of a DNA-d code than the lower bound computed by Limbachiya et. al. for certain parameters. Further a formula for the maximum size of a DNA-d code is given when the minimum distance of the DNA-d code is 2. This thesis states and proves this formula.

1.3 Organisation of the thesis

This section gives a description of how the remainder of this thesis is organised.

Chapter 2: Prerequisites. This chapter provides some basic knowledge about DNA and how it is used to store data. The concepts discussed in this chapter will be used throughout the thesis. The last section of this thesis introduces some basic concepts of coding theory, these concepts are only a small part of the coding theory but very important to understand the information given in this thesis.

Chapter 3: Constraints on DNA-based data storage. This chapter will begin by explaining the two constraints that are important in DNA-based data storage. After that the DNA set and the DNA-d code are defined.

Chapter 4: Three algorithms to compute a DNA-*d* code. Firstly, this chapter gives the algorithm from the paper of Limbachiya et. al. [1]. Secondly, two different algorithms that compute a DNA-d code are given. The last section evaluates the given algorithms.

Chapter 5: A DNA-*2* code of maximum size. This chapter states the most important theorem of this thesis. This theorem gives a formula for the maximum size of a DNA-d code with minimum distance 2. Some more knowledge is needed to prove this theorem. This knowledge is given in Section 5.1 up to and including Section 5.4. In Section 5.1 we discuss the size of the DNA set, Section 5.2 discusses the isolated points, Section 5.3 discusses how the DNA words from the DNA set relate to each other given an minimum distance. In Section 5.4 we discuss the binary even weight codes. Using the information given in all these sections, Section 5.5 gives the proof.

Chapter 6: Conclusions and recommendations. The final chapter concludes the results of all the chapters of this thesis. It also gives recommendations for further research in this subject.

2 Prerequisites

To understand the basics about data storage using DNA, some basic knowledge about DNA and data storage is given in this chapter. Section 2.1 explains how we can use the structure of DNA for data storage. Next, Section 2.2 elaborates a little bit on how to encode data. After which the process of creating and reading DNA is described in Section 2.3. Lastly some basic concepts of coding theory are explained in Section 2.4.

2.1 What is DNA

In this section we explain some basic knowledge of DNA, the information in this section is retrieved from [2] and [5].

DNA is short for deoxyribonucleic acid and is found in every known organism. It is a macro molecule and it stores the biological information of an organism. DNA is composed of two strands, these strands are built by four basic building blocks. These units are called nucleotides. The four nucleotides are Adine (A), Thymine (T), Guanine (G) and Cytosine (C). In this thesis the letters A, T, G and C will be used. The building units also consist of a sugar and a phosphate group. The two strands are linked in a helix shape, this link happens in the following way: the nucleotide A in one strand aligns with the nucleotide T in the other strand and the nucleotide C in one strand aligns with the nucleotide G in the other strand. This is called complementary base pairing. We say that the two strands are reverse complement of each other.

The 4 nucleotides are important for storing data because the order in which they appear holds information. This is the phenomenon that is used to store archival data using DNA. In the next section we elaborate more on this process.

2.2 From data to DNA

As explained in the motivation of this thesis, DNA can be a good alternative for archival data storage. We assume that the archival data is binary. There are three steps in archival data storage using DNA.

1. Binary data is encoded to quaternary data. We use quaternary data because a DNA strand is made up of four nucleotides, Adine (A), Cytosine (C), Guanine (G) and Thymine (T).

2. DNA strands are created (synthesized) of the encoded quaternary data using a DNA synthesizer after which the DNA strands are stored until use.

3. We read the DNA strands using DNA sequencing to get the quaternary data. When the DNA strands are known, we decode the quaternary data to get the original binary data [1].

There are multiple ways to carry out the first step, for example using a Hamming code or a Reed-Solomon code. This thesis will not be focused on this subject. For more information about this subject we recommend reading the thesis of Eva Slingerland [6]. The second and the third step are discussed in the next section. In this thesis we will focus on which DNA sequences can be used to save archival data using DNA.

2.3 Creating and reading DNA

The creation of DNA strands is done by a method called DNA synthesizing. Reading DNA strands is done by DNA sequencing. In this section the methods to do this are explained, the information is retrieved from [2] and [3].

Both DNA synthesizing and DNA sequencing use a technique called polymerase chain reaction (PCR). PCR is used to make a large amount of copies of a DNA strand. There are four main components that PCR requires: a template, sequencing primers, a thermostable polymerase and individual nucleotides. The template is either single stranded or double stranded in the double helix shape (explained in Section 2.1). The template is copied a lot of times using the other three components with complementary base paring. These copies are needed to analyse a DNA strand.

To store information, it is needed to create a strand of DNA containing specific nucleotides in a specific order. This technique is called DNA synthesizing [3]. When the DNA strand is created using DNA synthesizing the PCR technique is used to create many copies, these copies are then stored for later use.

DNA sequencing is used to obtain the data from the copies of DNA strands. DNA sequencing obtains the order of the nucleotides in a strand, we also say that this method reads the DNA strand. DNA polymerase enzymes, also known as “sequencing by synthesis” [3], is the most popular technique to do this. This is a technique that uses PCR with the strands that need to be read as the template but fluorescent nucleotides are used. The complement sequence can be seen optically because every type of fluorescent nucleotide has a different color of light. The original DNA strand can be determined using complementary base pairing.

These techniques to create and read DNA strands can lead to errors. We say that an error has occurred when the quaternary sequence after encoding the binary data is not equal to the quaternary sequence obtained after reading the DNA strand. These errors can lead to mistakes in the decoded binary archival data after saving and this is something we want to avoid as much as possible.

Using PCR we copy a lot of DNA strands and by synthesizing and sequencing a lot of strands instead of just one the chance of an error to occur is smaller. But this will not prevent all errors. For example, an error might occur in the following way when DNA strands are read using DNA sequencing. The DNA strand is analysed by the color of its nucleotides but when two the same nucleotides are next to each other, the light intensity and therefore the color decreases [3]. This can result in errors.

2.4 Basic concepts of coding theory

This section states some fundamental definitions of coding theory in particular applied to the quaternary alphabet. The book “Coding Theory And Cryptography, the essentials” [7] is used as an information source for this section.

Suppose some binary information needs to be stored using DNA. The first step is encoding this information to data that consists of the letters A, T, C and G. These encoded symbols form a word that contains all the information. Later, this word is saved using DNA synthesizing. The DNA strand is read using DNA sequencing as explained in Section 2.3. For our convenience we use the following bijection between the nucleotides and the quaternary alphabet.

$$A \leftrightarrow 0, \quad T \leftrightarrow 1, \quad G \leftrightarrow 2, \quad C \leftrightarrow 3$$

Definition 1. We define a **word** of length n as $\underline{c} = [c_1 c_2 \dots c_n]$ where $c_i \in \{0, 1, 2, 3\}$ for $1 \leq i \leq n$. We say c_1 is the first **symbol** of $\underline{c}$, c_2 the second symbol etc. Take $a, b \in \mathbb{N}$ and $1 \leq a, b \leq n$. We say that the symbols at c_a and c_b are **next to each other** if $\mid a - b \mid = 1$.

Definition 2. A **quaternary code** is a set Q of words over the quaternary alphabet.

A block code is a code where all its words have the same length. Only block codes are considered in this thesis, so from now on a quaternary code (Q) will always be a quaternary block code. The words that belong to a given code are called codewords. The size of a code is defined as the number of codewords in a code and is denoted by $|Q|$. For example the set $\{[00], [12], [20], [23], [33]\}$ is a quaternary code where the codewords have length 2 and the size of this code is 5.

Definition 3. We define the **quaternary set** as the quaternary code of maximum size for codewords of length n. The quaternary set for $n = 2$ is

$$\{[00], [01], [02], [03], [10], [11], [12], [13], [20], [21], [22], [23], [30], [31], [32], [33]\}$$

note that a quaternary code is always a subset of this quaternary set.

As explained before, it is possible that one or multiple errors occur. The minimum distance of a code is a powerful tool to detect errors or even correct these errors. First the definition of the distance between two words is given, after the definition of a minimum distance of a code is given.

Definition 4. Let $\underline{c}_1$ and $\underline{c}_2$ be words of length n. The (Hamming) **distance** between $\underline{c}_1$ and $\underline{c}_2$ is the number of symbols in which they differ. The notation for the distance between these two words is $d(\underline{c}_1,\underline{c}_2)$.

All the codewords in a code have a distance to each other. For a code Q of at least two codewords the minimum distance of the code is the smallest distance between any two different codewords from the code.

Definition 5. The **minimum distance** of a code Q is $d = \min\{d(\underline{c}_1, \underline{c}_2), \forall \underline{c}_1, \underline{c}_2 \in Q : \underline{c}_1 \neq \underline{c}_2\}$.

The minimum distance of a code is a powerful property because not all the words from the quaternary set can be in the code (except when $d = 1$). All the codewords from a code have at least distance d to each other, using this an error can be detected. When a code has minimum distance d that code is $d - 1$ error detecting and $\lfloor \frac{d-1}{2} \rfloor$ error correcting. We refer to [7] for more information about these properties of codes.

3 Constraints on DNA-based data storage

In this thesis we are interested in which words we can use to store data. As explained in Section 2.3, it is possible that errors occur during the synthesizing and sequencing of DNA strands. Previous studies stated that there are two major reasons that cause these errors, namely homopolymer runs and GC content [1] [4] [3] [8]. In this chapter two constraints are explained, these constraints reduce the number of synthesis and sequencing errors caused by homopolymer runs and GC-content. Using these constraints we define the DNA set in Section 3.3. The DNA set consists of DNA words and these DNA words satisfy the constraints. In the last section, a third constraint is added: the minimum distance. Using this minimum distance a DNA-d code is defined in Section 3.4.

3.1 The no run-length constraint

A homopolymer run is a consecutive repetition of the same nucleotides. The first constraint that is implemented is the no run-length constraint. Each DNA nucleotide is read as a (color)signal during the DNA synthesis and sequencing. These nucleotides might be read as a single signal if two or more repeated nucleotides occur in a DNA strand [1] [3]. An might error occurs if this happens. For example in the word [20133320], the 3 is repeated three times. During the sequencing or synthesizing, the homopolymer run might be read longer or shorter than it actually is. To minimise the number of errors we do not allow a repetition of the same nucleotides in a DNA word.

Definition 6. A word has **no run-length** when the word does not contain two repeated nucleotides. The no run-length constraint says that no DNA word from the DNA set should contain two or more repeated nucleotides.

3.2 The GC-weight constraint

The other important source for errors is the number of G and C nucleotides in a DNA strand. We say that the GC-content of a DNA strand is the percentage of G and C nucleotides. Erlich et. al. states that DNA strands with GC-content higher then 60% have a high chance that an error occurs [1]. So the other requirement for the DNA set is the fixed GC-weight because a DNA set with fixed GC-weight is more stable and this avoids errors [1].

Definition 7. The number of G and C nucleotides in a word is the **GC-weight**. The notation for the GC-weight of a word is w. Note that with the notation introduced in Definition 1, the GC-weight is the number of 2 and 3 symbols in a word.

For example [023102] is a word with GC-weight $w = 3$ because the 2 is mapped to the G nucleotide and the 3 is mapped to the C nucleotide. As a result of this constraint the number of

0 and 1 symbols in a DNA word and the number of 2 and 3 symbols in a DNA word are linked to each other. To use this in the rest of the thesis the following definition is introduced.

Definition 8. As described we use the quaternary alphabet, {0,1,2,3}. We say that 0 and 1 are **opposites** of each other and that 2 and 3 are **opposites** of each other.

3.3 The DNA set

With this knowledge we define the DNA set containing the words that satisfy the constraints. The set will take 2 parameters. The first is the length (n) of the DNA words. The second is the GC-weight (w) as defined in Definition 7.

Definition 9. We define the **DNA set**, $DNA(n, w)$, as

$$DNA(n, w) = \{\underline{c} : \underline{c} \text{ has length } n, \text{ GC-weight } w, \text{ no run-length}\},$$

We say that the words in $DNA(n, w)$ are **DNA words** and the size of $DNA(n, w)$ is $\boldsymbol{B(n, w)}$ (i.e. $\mid DNA(n, w) \mid = B(n, w)$). Note that the DNA set is a subset of the quaternary set of length n.

The DNA set is important because this set contains the words that satisfy the constraints. These DNA words can be used to save archival data as they lower the chance that an error occurs. Look at the word $\underline{c} = [21320]$. $\underline{c}$ has length 5 and GC-weight 3 so $\underline{c} = [21320] \in DNA(5, 3)$.

In this thesis the DNA set is always created in the same way. This is relevant because in the algorithms explained in Chapter 4, it is important in what order the DNA words appear in the DNA set. To create the DNA set first generate all the words of length n over the quaternary alphabet lexicographical [9]. It is immediately checked if the words satisfy the no run-length constraint and the GC-weight constraint during the generation of the words. If a word satisfies the constraints it is a DNA word and is added to the set $DNA(n, w)$, if not then the word is not a DNA word and thus not added. In the next example it is shown how the set $DNA(3, 1)$ is created. The Python code for computing the DNA set is given in Appendix A.1.

Example 1. We generate the DNA set $DNA(3, 1)$. The first word generated lexicographical of length 3 over the quaternary alphabet is [000]. This word does not satisfy the no run-length constraint so we do not add it to $DNA(3, 1)$. The same applies to the next 3 words, [001], [002] and [003] so they are not added to $DNA(3, 1)$. The next word, [010], does satisfy the no run-length constraint but not the GC-weight of 1 so [010] is also not added to $DNA(3, 1)$. The first word that does satisfy the constraints is the word [012]. This word is added to the set $DNA(3, 1)$. Continue doing this and eventually the DNA set $DNA(3, 1)$ is:

$$\begin{aligned} DNA(3,1) = \{&[012], [013], [020], [021], [030], [031], [102], [103], \\ &[120], [121], [130], [131], [201], [210], [301], [310]\}. \end{aligned}$$

We now know that the size of $DNA(3, 1)$, also defined as $B(3, 1)$, is equal to 16. Other ways to get $B(n, w)$, without computing the set $DNA(n, w)$, are explained in Section 5.1.

3.4 A DNA-d code

We have lowered the chance of errors to occur with the two constraints but it is still possible for errors to occur. As explained in Section 2.4 the minimum distance of a code is a powerful tool to detect or even correct errors. The minimum distance of a set is defined in Definition 5. To add this constraint to our DNA set we define the following code.

Definition 10. A **DNA-*d* code** is a code containing words of length n, GC-weight w, no run-length and the minimum distance is d. We define a word from this code as a DNA codeword.

A DNA-d code is a subset of of the DNA set, so every DNA codeword is also a DNA word. Now the DNA codewords can be used to save archival data. The aim of this thesis is to improve the lower bound on the maximum number of DNA codewords.

Definition 11. Let the maximum number of DNA codewords of length n, GC-weight w and minimum distance d be $\boldsymbol{B_d(n,w)}$.

Note that the DNA set is a DNA-1 code and thus that $B_1(n,w) = B(n,w)$. It is important to realise that there can be multiple different DNA-d codes with size $B_d(n,w)$. To get a DNA-d code we start with all the DNA words from the DNA set and delete DNA words to create a set that satisfies the minimum distance.

Example 2. Take $n = 3$, $w = 1$ and $d = 2$ and we are looking for a DNA-2 code that satisfies these parameters. From Example 1 we know what $DNA(3,1)$ looks like. By observation we delete some DNA words to form the following code.

$$\{[012], [310], [103], [201]\}$$

This is a DNA-2 code. This DNA-2 code can be used to save data but this is not a DNA-2 code of maximum size. In Section 5.5 we learn that $B_2(3,1) = 8$.

Assume we have the DNA set and the desired minimum distance d is given. The DNA words from the DNA set have a distance to each other (Definition 4). That distance is either smaller than or equal to $d-1$ or greater than $d-1$. To use this throughout the thesis the following definition is introduced.

Definition 12. Take $\underline{c}_1, \underline{c}_2 \in DNA(n,w)$ and $d \in \mathbb{Z}^+$. $\underline{c}_1$ and $\underline{c}_2$ are **(d-1)-neighbours** when $\mathrm{d}(\underline{c}_1, \underline{c}_2) \leq d-1$.

When a DNA word of $DNA(n,w)$ has no $(d-1)$-neighbours that DNA word is special and is defined as an isolated point.

Definition 13. Let $\underline{c} \in DNA(n,w)$ and $d \in \mathbb{Z}^+$. $\underline{c}$ is called an **isolated point** when $\underline{c}$ has no $(d-1)$-neighbours. Let $\boldsymbol{IP_d(n,w)}$ be the set of isolated points matching the given parameters and $\boldsymbol{I_d(n,w)}$ the number of isolated points, thus: $|IP_d(n,w)| = I_d(n,w)$.

To get a better understanding of these definitions and their use, an example is given. In this example it is shown that a DNA word is an isolated point by checking that the DNA word has no $(d-1)$-neighbours.

Example 3. Let $n = 4$, $w = 2$ and $d = 2$. In this example it is shown that $\underline{c} = [0123]$ is an isolated point. In other words, we will try to find a (1)-neighbour of [0123] and show that no such DNA word exists. In this case $d = 2$ so two DNA words are (1)-neighbours when their distance is 1. A DNA word is a (1)-neighbour of [0123] when one symbol is changed. Note that a symbol can only change into its opposite due to the GC-weight. Remember that 0 and 1 are opposites and that 2 and 3 are opposites.

Look at the first symbol, $c_1 = 0$. It cannot change into a 1 because of the no run-length constraint but that is the only option due to the fixed GC-weight. So the first symbol cannot change. A similar argument is used for c_2, c_3 and c_4. In conclusion there is no DNA word of length 4, GC-weight 2 and no run-length that is a (1)-neighbour of $\underline{c}$.

4 Three algorithms to compute a DNA-*d* code

In this thesis we are interested in the maximum size of a DNA-d code. Unfortunately there does not (yet) exists an explicit formula for $B_d(n,w)$ but there are algorithms to find a DNA-d code and therefore a lower bound on $B_d(n,w)$. As explained in Section 3.4, a DNA-d code is generated in the following way: the first step, start with the DNA set. The second step: either delete some DNA words from the DNA set to satisfy the distance constraint or add some DNA words to a new list without breaking the distance constraint. In this section three algorithms are explained, the algorithms all differ in the second step.

The most important factor that affects the generated DNA-d code and thus the lower bound for $B_d(n,w)$ is the way and in what order the DNA words are added or deleted. When a DNA word is added to a new list or deleted this can effect the next steps of creating the DNA-d code. The first algorithm is from the paper of Limbachiya et. al. [1] and the second and third algorithm are algorithms that we designed. These algorithms are variations on the algorithm of Limbachiya et. al. Algorithm 2 and 3 are conceived to improve the lower bound given by Algorithm 1.

4.1 Algorithm 1

The first algorithm is the algorithm from the paper of Limbachiya et. al. [1]. The algorithm takes as input a length n, GC-weight w and minimum distance d, the result is a DNA-d code that satisfies the input. The algorithm checks for all the DNA words of the DNA set what its $(d-1)$-neighbours are and how many $(d-1)$-neighbours each DNA word has. This information is put in a dictionary with the DNA words as keys and its number of $(d-1)$-neighbours as its value. An iteration starts with deleting the DNA word with the most $(d-1)$-neighbours. After the dictionary is adjusted for all those $(d-1)$-neighbours. When multiple DNA words have the maximum number of $(d-1)$-neighbours the first of those DNA words is chosen. Keep iterating until all the DNA words in the dictionary have zero $(d-1)$-neighbours, the DNA words that are still in the dictionary form a DNA-d code. The code for this algorithm is written in Python and can be found in Appendix A.2.

```
Algorithm 1:
  Data: Length $n$, GC-weight $w$ and minimum distance $d$
  Result: A DNA-$d$ code
  1. Generate a list called DNAset containing all the DNA words in lexicographical order
     from the set $DNA(n, w)$.
  2. Create a dictionary called words_in_sphere where the keys are the DNA words from
     DNAset and its value is the number of $(d-1)$-neighbours of the key.
  if There is a value in words_in_sphere that is not equal to 0 then
      delete the first DNA word with the maximum value from words_in_sphere. Reduce
        the values of the DNA words in words_in_sphere that are a $(d-1)$-neighbour of the
        deleted DNA word by one.
  else
      The keys of words_in_sphere form a DNA-$d$ code.
  end
```

This algorithm computes a DNA-d code. We count the number of DNA codewords in the DNA-d code to get a lower bound on $B_d(n, w)$. An example is given to get a better understanding of the algorithm. In this example we use $n = 2$, $w = 1$ and $d = 2$.

Example 4. The input of the algorithm is $n = 2$, $w = 1$ and $d = 2$. The algorithm computes a DNA-d code that satisfies the given parameters.
Step 1: We generate list "*DNAset*" that consists of all the DNA words of $DNA(2, 1)$. The list "*DNAset*" is:

$$DNAset = [[02], [03], [12], [13], [20], [21], [30], [31]]$$

Step 2: We create a dictionary "*words_in_sphere*" with the 8 words as keys. For the value we count the number of (1)-neighbours of the key. For example: The DNA word [02] has two DNA words at distance 1, [03] and [12], so the value of [02] is 2. The generated dictionary is:

$$words\_in\_sphere = \{[02] : 2, [03] : 2, [12] : 2, [13] : 2, [20] : 2, [21] : 2, [30] : 2, [31] : 2\}$$

There is a value that is not equal to 0 so we start the first iteration. We delete the DNA word with the maximum value. Because there are multiple keys that have the maximum value we choose the first one. In this case we delete the DNA word [02] from the dictionary. The DNA words that have distance 1 to [02] are [03] and [12] so we adjust the values of those words. Now the dictionary is:

$$words\_in\_sphere = \{[03] : 1, [12] : 1, [13] : 2, [20] : 2, [21] : 2, [30] : 2, [31] : 2\}$$

Because there is still a value that is not equal to 0 we get into the second iteration. The first key with the highest value is [13] and the two DNA words that have distance 1 to [13] are [03] and [12]. After this iteration the dictionary looks like this:

$$words\_in\_sphere = \{[03] : 0, [12] : 0, [20] : 2, [21] : 2, [30] : 2, [31] : 2\}$$

After 2 iterations where [20] and [31] are deleted respectively the dictionary is:

$$words\_in\_sphere = \{[03] : 0, [12] : 0, [21] : 0, [30] : 0\}$$

Every value is 0 thus we have reached the end of the algorithm. The found DNA-2 code is:

$$\{[03], [12], [21], [30]\}$$

The size of the generated DNA-2 code is 4 thus $B_2(3, 1) \geq 4$.

4.2 Algorithm 2

The second algorithm is a variation on the first algorithm. As explained in the introduction of this chapter the order in which DNA words are added or deleted is the most important factor in generating a DNA-d code. Our goal is to improve the lower bound of $B_d(n, w)$. We came up with a different algorithm that might give better results than Algorithm 1. The difference between these algorithms is that the minimum key is added to a list instead of deleting the maximum key.

Algorithm 2 takes the same input as Algorithm 1, a length n, GC-weight w and minimum distance d and the result is a DNA-d code that satisfies the input. This algorithm creates a dictionary just as in Algorithm 1 and it also creates an empty list. An iteration starts with adding the key with minimum value to the list. When multiple keys have the minimal value the first one is chosen. Next the minimal key and all the $(d-1)$-neighbours of this minimal key are deleted from the dictionary. Last the values of all the $(d-1)$-neighbours of the deleted keys are reduced to make the dictionary correct again. Keep iterating until the dictionary is empty. The DNA words that are in the list form a DNA-d code. The DNA words now are also DNA codewords. The python code of this algorithm can be found in Appendix A.3. The algorithm is given in Algorithm 2 and after an example is given.

```
Algorithm 2:
  Data: Length $n$, GC-weight $w$ and minimum distance $d$
  Result: A DNA-$d$ code
  1. Generate a list called $DNAset$ containing all the DNA words in lexicographical order
     from the set $DNA(n, w)$.
  2. Create a dictionary called words_in_sphere where the keys are the DNA words from
     $DNAset$ and the value is the number of $(d-1)$-neighbours of the key.
  3. Create an empty list named DNAdistance.
  if words_in_sphere is not empty then
      Add the first key with minimal value to the list DNAdistance, delete this minimal
      key from words_in_sphere and delete all the DNA words from words_in_sphere that
      are a $(d-1)$-neighbours of the minimum key. Reduce the values of the DNA words
      in words_in_sphere that are a $(d-1)$-neighbour of a deleted DNA word.
  else
      The words in the list DNAdistance form a DNA-$d$ code.
  end
```

Example 5. As input of Algorithm 2 we again choose $n = 2$, $w = 1$ and $d = 2$. Step 1 and 2 are the same as in Example 4 so the dictionary is:

$$words\_in\_sphere = \{[02] : 2, [03] : 2, [12] : 2, [13] : 2, [20] : 2, [21] : 2, [30] : 2, [31] : 2\}$$

In step 3 we also create an empty list named "*DNAdistance*".

$$DNAdistance = []$$

The dictionary is not empty so we start the first iteration. The first minimal key is [02] so we add this DNA word to "*DNAdistance*". Next we delete [02] and the DNA words that are (1)-neighbours of [02] from "*words_in_sphere*". Last adjust the value of the (1)-neigbours of the deleted DNA words. The list and dictionary now are:

$$DNAdistance = [[02]]$$

$$words\_in\_sphere = \{[13]:0, [20]:2, [21]:2, [30]:2, [31]:2\}$$

The dictionary is not empty so again we start an iteration. The first minimal key is [13] so we add it to "*DNAdistance*". There are no (1)-neighbours of [13] in the dictionary so all we do is delete [13] from the dictionary and we end this iteration with:

$$DNAdistance = [[02], [13]]$$

$$words\_in\_sphere = \{[20]:2, [21]:2, [30]:2, [31]:2\}$$

In the next two iterations the DNA words [20] and [31] are added to "*DNAdistance*" respectively, the list and dictionary are now:

$$DNAdistance = [[02], [13], [20], [31]]$$

$$words\_in\_sphere = \{\}$$

Because the dictionary is empty the end of the algorithm is reached. The DNA words in "*DNAdistance*" form a DNA-d code and the size of this code is 4 thus $B_2(3,1) \geq 4$. We have already established this in Example 4. In this case we have not improved the lower bound.

4.3 Algorithm 3

The last algorithm that will be explained in this thesis is a bit different than the other two algorithms because this algorithm does not generate the dictionary with $(d-1)$-neighbours. First a list with all the $B(n,w)$ DNA words is created. Remember from Section 3.3 that these DNA words are generated lexicographically. Also create an empty list, when the algorithm is finished the words in this list form a DNA-d code. In every iteration the first DNA word from the generated list is moved to the other list if this does not break the minimum distance. If moving the DNA word does break the minimum distance this word is deleted. This algorithm is a bit simpler than the other two algorithms but it might give a better lower bound. Below the algorithm and an example is given. The python code can be found in Appendix A.4.

```
Algorithm 3:
  Data: Length $n$, GC-weight $w$ and minimum distance $d$
  Result: A DNA-$d$ code
  1. Generate a list called DNAset containing all the DNA words in lexicographical order
     from the set $DNA(n,w)$ as explained in Section 3.3.
  2. Create an empty list named DNAdistance.
  3. Add the first DNA word of DNAset to DNAdistance and delete it from DNAset
  if DNAset is not empty then
      Define the first DNA word from DNAset as $\underline{c}$.
      if If $\underline{c}$ is a $(d-1)$-neighbour of a DNA word already in DNAdistance then
          Delete $\underline{c}$ from DNAset
      else
          Add $\underline{c}$ to DNAdistance and delete $\underline{c}$ from DNAset
      end
  else
      The DNA words in the list DNAdistance form a DNA-$d$ code.
  end
```

Example 6. For this example we again take the parameters: $n = 2$, $w = 1$ and $d = 2$. In step 1 we generate the list "*DNAset*", step 2 is creating an empty list named "*DNAdistance*".

$$DNAset = [[02], [03], [12], [13], [20], [21], [30], [31]]$$

$$DNAdistance = []$$

For the third step we delete the first DNA word, [02], from "*DNAset*" and add it to "*DNAdistance*".

$$DNAset = [[03], [12], [13], [20], [21], [30], [31]]$$

$$DNAdistance = [[02]]$$

In the first iteration we define $\underline{c} = [03]$. Because [02] and [03] are (1)-neighbours and [02] is in "*DNAdistance*" we delete [03]. The same argument is used for the next DNA word, [12]. After these iterations the lists are:

$$DNAset = [[13], [20], [21], [30], [31]]$$

$$DNAdistance = [[02]]$$

In the next iteration, [13] is added to "*DNAdistance*" because [02] and [13] are not (1)-neighbours. If we continue this process eventually the lists are:

$$DNAset = []$$

$$DNAdistance = [[02], [13], [20], [31]]$$

The algorithm is done and the computed DNA-d code is {[02],[13],[20],[31]}. This again does not improve our lower bound in this case but it might in other cases.

4.4 Evaluation of the algorithms

Remember that our goal was to improve the lower bound computed with Algorithm 1. To evaluate the three algorithms that are explained in Section 4.1, 4.2 and 4.3 a table of the size of the computed DNA-d codes is given in Table 4.1. The 1, 2 and 3 in the table correspond to the similar named algorithms. In this table we use that the GC-weight $w = \lfloor \frac{n}{2} \rfloor$. The value for $B(n, w)$ is also given. The values that are underlined are the highest lower bounds computed with these algorithms with an exception when the three algorithms compute the same lower bound. No value is underlined in that case. Table 4.1 gives the values for $2 \leq n \leq 11$ and $2 \leq d \leq 4$. In Appendix B a table with more values is given.

In the table we can see that for most of the cases the algorithms compute different lower bounds for $B_d(n, \lfloor \frac{n}{2} \rfloor)$. For the values given in the table we can see that most of the time the lower bound computed with Algorithm 2 is the best. But we do not know if this is close to the value for $B_d(n, w)$. We do know that Algorithm 2 does not always compute the best lower bound because for $n = 6$, $w = 3$ and $d = 4$ the lower bound computed by Algorithm 3 is better than the lower bound computed by Algorithm 2. It might also be the case that for bigger n and d an other algorithm gives a better lower bound. Thus we do not know what algorithm is the best, or even if one of the algorithms is always better than another. The aim of this thesis was to improve the lower bound that was given in the paper of Limbachiya et. al. and we have done that for $3 \leq n \leq 11$, $w = \lfloor \frac{n}{2} \rfloor$ and $3 \leq d \leq 5$.

When analyzing Table 4.1 a thing that stands out is the fact that the three algorithms give the same lower bound for $d = 2$, $2 \leq n \leq 11$ and $w = \lfloor \frac{n}{2} \rfloor$. This makes us wonder if the value that is given in the table for $d = 2$ is the value for $B_2(n, \lfloor \frac{n}{2} \rfloor)$. This is the case and in the next chapter the formula together with its proof for $B_2(n, w)$ is given.

		$d = 2$			$d = 3$			$d = 4$		
n	$B(n, w)$	1	2	3	1	2	3	1	2	3
2	8	4	4	4	-	-	-	-	-	-
3	16	8	8	8	2	$\underline{3}$	2	-	-	-
4	56	32	32	32	11	$\underline{12}$	7	4	4	4
5	128	68	68	68	17	$\underline{18}$	$\underline{18}$	7	7	7
6	424	216	216	216	44	$\underline{53}$	45	16	20	$\underline{21}$
7	1040	528	528	528	110	$\underline{119}$	101	36	$\underline{44}$	37
8	3352	1704	1704	1704	289	$\underline{326}$	286	86	$\underline{127}$	97
9	8576	4336	4336	4336	662	$\underline{762}$	687	199	$\underline{227}$	216
10	27208	13688	13688	13688	1810	$\underline{2126}$	1939	525	$\underline{618}$	557
11	71568	35936	35936	35936	4320	$\underline{5145}$	4666	1235	$\underline{1426}$	1323

Table 4.1: Lower bounds for $B_d(n, w)$ computed with the 3 algorithms, $w = \lfloor \frac{n}{2} \rfloor$. The underlined values denote the highest lower bounds except when all three algorithms compute the same lower bound.

5 A DNA-2 code of maximum size

The algorithms in Chapter 4 do not improve the current lower bound for $B_2(n, w)$. It is not possible to improve this lower bound when the computed DNA-d codes are of maximum size. That is why we want to know the value for $B_2(n, w)$. In this chapter the formula for $B_2(n, w)$ is given together with a proof.

The parameter d is equal to 2 in this chapter and this will not be explicitly mentioned every time. We also shorten the term (1)-neighbour to: "neighbour". The isolated points of $DNA(n, w)$ play an important role in the formula for $B_2(n, w)$. Note that in this situation an isolated point is a DNA word with no neighbours.

Theorem 1. A DNA-2 code of size $B_2(n, w)$ contains all the isolated points of $DNA(n, w)$, that is $IP_2(n, w)$.

Proof. Let $DNA_2(n, w) \subseteq DNA(n, w)$ be a DNA-2 code of maximum size $B_2(n, w)$. Let $\underline{i} \in IP_2(n, w)$ be an isolated point of $DNA(n, w)$. Assume $\underline{i} \notin DNA_2(n, w)$. We derive a contradiction. Take $\underline{c} \in DNA_2(n, w)$ arbitrary, then $\underline{c} \in DNA(n, w)$ and $d(\underline{i}, \underline{c}) \geq 2$ because $\underline{i}$ is an isolated point. So $\underline{i}$ can be added to $DNA_2(n, w)$ and $DNA_2(n, w)$ would still match its given parameters. This proves that $DNA_2(n, w)$ was not a DNA-2 code of maximum size before $\underline{i}$ was added and this is a contradiction. □

Theorem 1 proves that $IP_2(n, w)$ is always a subset of a DNA-2 code of maximum size. The size of $IP_2(n, w)$ is defined as $I_2(n, w)$. Now consider the set $DNA(n, w) \backslash IP_2(n, w)$, this is the biggest set of DNA words that does not contain an isolated point. In this chapter we prove that there exists a subset of $DNA(n, w) \backslash IP_2(n, w)$ of size $\frac{B(n,w)-I_2(n,w)}{2}$ that forms, together with the isolated points, a DNA-2 code of maximum size. The following theorem gives the formula for $B_2(n, w)$.

Theorem 2.

$$B_2(n, w) = \frac{B(n, w) - I_2(n, w)}{2} + I_2(n, w) = \frac{B(n, w) + I_2(n, w)}{2}$$

To prove that a subset of $DNA(n, w) \backslash IP_2(n, w)$ of size $\frac{B(n,w)-I_2(n,w)}{2}$ exists and that it forms, together with the isolated points, a DNA-2 code of maximum size we need some more knowledge. Eventually in Section 5.5 the proof of Theorem 2 is given. We see that to calculate $B_2(n, w)$ we need a formula for $B(n, w)$ and a formula for $I_2(n, w)$. These are given in Sections 5.1 and 5.2 respectively.

5.1 The size of the DNA set

To calculate the value of $B_2(n, w)$ we need a formula for $B(n, w)$. Limbachiya et. al. [1] stated and proved the following theorem.

Theorem 3. The number of DNA words in the DNA set is given by

$$B(n,w) = \sum_{y=0}^{\nu-1} 2^{2\nu+1-2y} \binom{\nu-1}{y} \binom{n-\nu}{\nu-y} + \sum_{y=0}^{\nu-2} 2^{2\nu-1-2y} \binom{\nu-1}{y} \binom{n-\nu-1}{\nu-y-2} \tag{5.1}$$

for $\nu > 0$, where $\nu = min(w, n-w)$. Further $B(n,w) = 2$ for $min(w, n-w) = 0$.

Proof. The proof of this theorem is given in [1] in section III. □

Theorem 3 gives a formula for $B(n,w)$ but this formula contains multiple binomial coefficients that can take a long time to compute. That is why we also give a recursive function for $B(n,w)$. We say that the 0- and 1-symbol are low category symbols and the 2- and 3-symbol are high category symbols.

Definition 14. Define $DNA(n,w,i)$ with $i \in \{L, H\}$ as follows

$$DNA(n,w,L) = \{\underline{c} \in DNA(n,w) : c_n = 0 \vee c_n = 1\}$$

$$DNA(n,w,H) = \{\underline{c} \in DNA(n,w) : c_n = 2 \vee c_n = 3\}$$

Define $B(n,w,i)$ with $i \in \{L, H\}$ as follows

$$B(n,w,L) = |DNA(n,w,L)|$$

$$B(n,w,H) = |DNA(n,w,H)|$$

The sets $DNA(n,w,L)$ and $DNA(n,w,H)$ are disjoint because a DNA word can only have one symbol as its last symbol.

Theorem 4. Let $n, w \in \mathbb{N}$, $n \geq 1$ and $0 \leq w \leq n$ then

$$B(n,w) = B(n,w,L) + B(n,w,H) \tag{5.2}$$

where $B(n,w,L)$ and $B(n,w,H)$ are specified as follows:

- $B(n,0,L) = 2$
- $B(n,0,H) = 0$
- $B(n,n,L) = 0$
- $B(n,n,H) = 2$

For $n \geq 2$ and $0 < w < n$:

$$B(n,w,L) = B(n-1,w,L) + 2B(n-1,w,H) \tag{5.3}$$

$$B(n,w,H) = 2B(n-1,w-1,L) + B(n-1,w-1,H) \tag{5.4}$$

Proof. $DNA(n,0)$ is the set containing the two DNA words of length n that alternate the 0-symbol and the 1-symbol because the GC-weight is zero. The first symbols of the DNA words are 0 and 1 respectively. Using Definition 14, $B(n,0,L) = 2$ and $B(n,0,H) = 0$. A similar argument for $B(n,n,L) = 0$ and $B(n,n,H) = 2$ is used.

Let $\underline{c} \in DNA(n, w, L)$ with $n \geq 2$ and $0 < w < n$. We prove that $\underline{c}$ can always be created by the following method: adding the last symbol of $\underline{c}$ to a DNA word of $DNA(n-1, w, L)$ or $DNA(n-1, w, H)$ and that there is only one way of creating $\underline{c}$ with this method. Because $\underline{c} \in DNA(n, w, L)$ we have $c_n = 0$ or $c_n = 1$. So $[c_1 c_2 \dots c_{n-1}] \in DNA(n-1, w, H)$ or $[c_1 c_2 \dots c_{n-1}] \in DNA(n-1, w, L)$.

First assume that $[c_1 c_2 \dots c_{n-1}] \in DNA(n-1, w, H)$ then there are two options to create $\underline{c}$, $c_n = 0$ or $c_n = 1$. Both of these options will not break the no run-length constraint because $c_{n-1} = 2$ or $c_{n-1} = 3$. Next assume that $[c_1 c_2 \dots c_{n-1}] \in DNA(n-1, w, L)$ then there is only one option to create $\underline{c}$, if $c_{n-1} = 0$ then $c_n = 1$ and if $c_{n-1} = 1$ then $c_n = 0$.

Because $[c_1 c_2 \dots c_{n-1}]$ is always in $DNA(n-1, w, L)$ or in $DNA(n-1, w, H)$ we can always create $\underline{c}$ out of these two sets with the method explained above. Because the sets $DNA(n-1, w, L)$ and $DNA(n-1, w, H)$ are disjoint $\underline{c}$ can only be created by adding the last symbol in one way, thus $\underline{c}$ will never be counted twice. In conclusion: to count $B(n, w, L)$ add $B(n-1, w, L)$ and $2 \cdot B(n-1, w, H)$.

Next let $\underline{c} \in DNA(n, w, H)$ with $n \geq 2$ and $0 < w < n$. The proof for $B(n, w, H)$ is similar to the proof of $B(n, w, L)$ but note that because $\underline{c} \in DNA(n, w, H)$ we have $c_n = 2$ or $c_n = 3$. So $[c_1 c_2 \dots c_{n-1}] \in DNA(n-1, w-1, L)$ or $[c_1 c_2 \dots c_{n-1}] \in DNA(n-1, w-1, H)$.

Because $DNA(n, w, L)$ and $DNA(n, w, H)$ are disjoint and the union of these sets is $DNA(n, w)$, i.e. $DNA(n, w, L) \cup DNA(n, w, H) = DNA(n, w)$. We conclude that $B(n, w) = B(n, w, L) + B(n, w, H)$. □

Table 5.1 is created with the equations in Theorem 4, "L" denotes the output of $B(n, w, L)$ and "H" denotes the output of $B(n, w, H)$. "T" is the value of $B(n, w)$. The Python code that calculates these values can be found in Appendix A.6. A bigger table with more values of $B(n, w)$ is given in Appendix B.

	w=0			w=1			w=2			w=3		
n	L	H	T	L	H	T	L	H	T	L	H	T
1	2	0	2	0	2	2	-	-	-	-	-	-
2	2	0	2	4	4	8	0	2	2	-	-	-
3	2	0	2	12	4	16	4	12	16	0	2	2
4	2	0	2	20	4	24	28	28	56	4	20	24
5	2	0	2	28	4	32	84	44	128	44	84	128
6	2	0	2	36	4	40	172	60	232	212	212	424
7	2	0	2	44	4	48	292	76	368	636	404	1040
8	2	0	2	52	4	56	444	92	536	1444	660	2104
9	2	0	2	60	4	64	628	108	736	2764	980	3744

Table 5.1: The values of $B(n, w, L)$ indicated by "L", $B(n, w, H)$ indicated by "H" and $B(n, w)$ indicated by "T" for DNA words of length n and GC-weight w.

5.2 The number of isolated points

As stated in Theorem 1, we know that the set $IP_2(n,w)$ is always included in a DNA-2 code of size $B_2(n,w)$. We want to count the number of isolated points. To do so we need to know when a DNA word is an isolated point and when it is not an isolated point. The following theorem gives us this.

Theorem 5. Let $\underline{c} \in DNA(n,w)$. Then $\underline{c}$ is an isolated point $\iff$ every symbol of $\underline{c}$ is next to its opposite.

Proof. $\implies$
Take $\underline{c}_1 \in IP_2(n,2)$. Assume that there exists an a to which applies that $\underline{c}_{1,a}$ is not next to its opposite. We derive a contradiction.

Case 1: If $\underline{c}_{1,a} = 0$ then $\begin{cases} \underline{c}_{1,a+1} = 2 \text{ or } 3 & \text{if } a = 1 \\ \underline{c}_{1,a-1} = 2 \text{ or } 3 & \text{if } a = n \\ \underline{c}_{1,a-1} = 2 \text{ or } 3 \text{ and } \underline{c}_{1,a+1} = 2 \text{ or } 3 & \text{otherwise} \end{cases}$.

Look at the DNA word, $\underline{c}_2 \in DNA(n,w)$, that is exactly the same as $\underline{c}_1$ but $\underline{c}_{1,a}$ is changed into its opposite. In this case $\underline{c}_{2,a} = 1$. Now $\underline{c}_2$ still has length n, GC-weight w and $\underline{c}_2$ has no run-length so $\underline{c}_2$ is a DNA word. $\underline{c}_1$ has a neighbour because $d(\underline{c}_1, \underline{c}_2) = 1$. $\underline{c}_1$ not an isolated point. The proof of case 2 ($\underline{c}_{1,a} = 1$), case 3 ($\underline{c}_{1,a} = 2$) and case 4 ($\underline{c}_{1,a} = 3$) is similar to the proof of case 1. This completes the contradiction.

$\impliedby$
Let $\underline{c} \in DNA(n,w)$, every symbol of $\underline{c}$ is (on the left side and/or on the right side) next to its opposite. We prove that $\underline{c}$ is an isolated point. Take $1 \le a \le n$ arbitrary so c_a is a symbol of $\underline{c}$. We check that it is not possible to change c_a and still match the given n, w and no run-length.

- Because of the fixed GC-weight w, c_a can only change into its opposite.
- Because c_a is next to its opposite, due to the no run-length constraint c_a cannot change into its opposite.

This proves that no symbol can change so the DNA word $\underline{c}$ has no neighbours. $\underline{c}$ is an isolated point. □

To calculate $B_2(n,w)$ we need a formula for $I_2(n,w)$. This formula is given in the next theorem. We first define two subsets of $IP_2(n,w)$.

Definition 15. Define $IP_2(n,w,i)$ with $i \in \{L,H\}$ as follows

$$IP_2(n,w,L) = \{\underline{c} \in IP_2(n,w) : c_n = 0 \lor c_n = 1\}$$

$$IP_2(n,w,H) = \{\underline{c} \in IP_2(n,w) : c_n = 2 \lor c_n = 3\}$$

Define $I_2(n,w,i)$ with $i \in \{L,H\}$ as follows

$$I_2(n,w,L) = |IP_2(n,w,L)|$$

$$I_2(n,w,H) = |IP_2(n,w,H)|$$

Note that $IP_2(n,w) \subseteq DNA(n,w)$, $IP_2(n,w,L) \subseteq DNA(n,w,L)$ and that $IP_2(n,w,H) \subseteq DNA(n,w,H)$. It follows directly that $I_2(n,w) \le B(n,w)$, $I_2(n,w,L) \le B(n,w,L)$ and that $I_2(n,w,H) \le B(n,w,H)$.

Theorem 6. Let $n, w \in \mathbb{N}$, $n \geq 1$ and $0 \leq w \leq n$ then

$$I_2(n,w) = I_2(n,w,L) + I_2(n,w,H) \tag{5.5}$$

Where $I_2(n,w,L)$ and $I_2(n,w,H)$ are specified as follows:

- $I_2(1,0,L) = 0$
- $I_2(1,1,L) = 0$
- $I_2(1,0,H) = 0$
- $I_2(1,1,H) = 0$

for $n \geq 2$:

- $I_2(n,0,L) = 2$
- $I_2(n,1,L) = 0$
- $I_2(n,0,H) = 0$
- $I_2(n,1,H) = 0$
- $I_2(n,n,L) = 0$
- $I_2(n,n-1,L) = 0$
- $I_2(n,n,H) = 2$
- $I_2(n,n-1,H) = 0$

And for $n \geq 3$ and $1 < w < n-1$:

$$I_2(n,w,L) = I_2(n-1,w,L) + 2I_2(n-2,w,H) \tag{5.6}$$

$$I_2(n,w,H) = 2I_2(n-2,w-2,L) + I_2(n-1,w-1,H) \tag{5.7}$$

Proof. The two DNA words from $DNA(1,w)$ always have distance 1 to each other for $0 \leq w \leq 1$ so $I_2(1,w,L) = 0$ and $I_2(1,w,H) = 0$.

$DNA(n,0)$ is the set containing two DNA words and the symbols of both these DNA words alternate between "0" and "1" where one DNA word starts with 0 and the other starts with 1. The distance between these DNA words is n so both DNA words are isolated points. Thus $I_2(n,0,L) = 2$ and $I_2(n,0,H) = 0$. The proof for $I_2(n,n,L)$ and $I_2(n,n,H)$ is similar.

The 0- and 1-symbols are difined as low category symbols. The 2- and 3-symbols are defined as high category symbols. Let $\underline{c} \in DNA(n,1)$ then there is only one high category symbol in $\underline{c}$ so that symbol cannot be next to its opposite. Using Theorem 5, $\underline{c}$ is never an isolated point. Thus $I_2(n,1,L) = 0$ and $I_2(n,1,H) = 0$. The proof for $I_2(n,n-1,L)$ and $I_2(n,n-1,H)$ is similar.

Next we will look at the part of the formula where $I_2(n,w,L)$ is described. Let $\underline{c} \in IP_2(n,w,L)$ with $n \geq 3$ and $1 < w < n-1$. Because $\underline{c} \in IP_2(n,w,L)$ $\underline{c}$ always ends with 2 or more low category symbols (Theorem 5). We prove that $\underline{c}$ can always be created by the following method: adding the last symbol to a DNA word of $IP_2(n-1,w,L)$ or adding the last two symbols to a DNA word of $IP_2(n-2,w,H)$ and that there is only one way to create $\underline{c}$ with this method.

First we assume that $\underline{c}$ ends with exactly 2 low category symbols then $[c_1 \dots c_{n-2}] \in DNA(n-2,w,H)$. Note that $[c_1 \dots c_{n-2}] \in IP_2(n-2,w,H)$ because of Theorem 5. There are two options to create $\underline{c}$: $c_{n-1} = 0$ and $c_n = 1$ or $c_{n-1} = 1$ and $c_n = 0$. This will not break the no run-length constraint because $c_{n-2} = 2$ or $c_{n-2} = 3$. Next we assume that $\underline{c}$ ends with 3 or more low category symbols. Then $[c_1 \dots c_{n-1}] \in DNA(n-1,w,L)$ and because of Theorem 5, $[c_1 \dots c_{n-1}] \in IP_2(n-1,w,L)$. To create $\underline{c}$ there is only one option for c_n, if $c_{n-1} = 0 \Rightarrow c_n = 1$ and if $c_{n-1} = 1 \Rightarrow c_n = 0$.

We see that we can always create $\underline{c}$ out of the set $IP_2(n-2, w, H)$ or out of the set $IP_2(n-1, w, L)$. Last we prove that there is only one way of creating $\underline{c}$ out of these two sets, or in other words, an isolated point is never counted double. If $\underline{c}$ ends with exactly 2 low category symbols, $\underline{c}$ can never be created from an isolated point from $IP_2(n-1, w, L)$ because the DNA word $[c_1 c_2 \ldots c_{n-1}] \notin IP_2(n-1, w, L)$. If $\underline{c}$ ends with 3 or more low category symbols, $\underline{c}$ can never be created from the DNA words of $IP_2(n-2, w, H)$ because the DNA words in $IP_2(n-2, w, H)$ end with a high category symbol. In conclusion: to count $I_2(n, w, L)$ add two times $I_2(n-2, w, H)$ and one time $I_2(n-1, w, L)$.

Next Let $\underline{c} \in IP_2(n, w, H)$ with $n \geq 3$ and $1 < w < n-1$. The proof for $I_2(n, w, H)$ is similar to the proof of $I_2(n, w, L)$. Note that $\underline{c}$ can end with exactly 2 high category symbols or $\underline{c}$ can end with 3 or more high category symbols.

Because $IP_2(n, w, L)$ and $IP_2(n, w, H)$ are disjoint and $IP_2(n, w, L) \cup IP_2(n, w, H) = IP_2(n, w)$. We conclude that $I_2(n, w) = I_2(n, w, L) + I_2(n, w, H)$. □

Table 5.2 gives the output of the recursive formula for $I_2(n, w)$, where L denotes $I_2(n, w, L)$, H denotes $I_2(n, w, H)$ and T gives the value for $I_2(n, w)$. The Python code of this recursive formula is given in Appendix A.7. A bigger table with more values can be found in Appendix B. We now know when a DNA word is an isolated point and therefore also when a DNA word is not an isolated point. All the DNA words of $DNA(n, w)$ can be clustered in a certain way and this is needed to prove Theorem 2. It is explained in the next section.

	w=0			w=1			w=2			w=3			w=4			w=5		
n	L	H	T	L	H	T	L	H	T	L	H	T	L	H	T	L	H	T
1	0	0	0	0	0	0	-	-	-	-	-	-	-	-	-	-	-	-
2	2	0	2	0	0	0	0	2	2	-	-	-	-	-	-	-	-	-
3	2	0	2	0	0	0	0	0	0	0	2	2	-	-	-	-	-	-
4	2	0	2	0	0	0	4	4	8	0	0	0	0	2	2	-	-	-
5	2	0	2	0	0	0	4	4	8	4	4	8	0	0	0	0	2	2
6	2	0	2	0	0	0	12	4	16	4	4	8	4	12	16	0	0	0
7	2	0	2	0	0	0	20	4	24	12	4	16	4	12	16	4	20	24
8	2	0	2	0	0	0	28	4	32	20	4	24	28	28	56	4	20	24
9	2	0	2	0	0	0	36	4	40	28	4	32	52	44	96	44	52	96

Table 5.2: The values of $I_2(n, w, L)$ indicated by "L", $I_2(n, w, H)$ indicated by "H" and $I_2(n, w)$ indicated by "T" for DNA words of length n and GC-weight w.

5.3 Clusters

Definition 16. We define the graph $G_2(n, w)$ on the set $DNA(n, w)$ with $d = 2$ as follows: The DNA words of $DNA(n, w)$ are the nodes of the graph and two nodes are connected by an edge when the DNA words corresponding to those nodes are neighbours, i.e. two DNA words $\underline{c}_1, \underline{c}_2$ are connected by an edge if $d(\underline{c}_1, \underline{c}_2) = 1$.

The DNA set can be visualised by a graph (Definition 16). Figure 5.1 and Figure 5.2 show $G_2(2, 1)$ and $G_2(3, 1)$. As seen in the figures $G_2(n, w)$ consists of separate smaller graphs, these smaller graphs are called clusters.

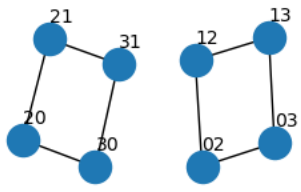

Figure 5.1: $G_2(2,1)$

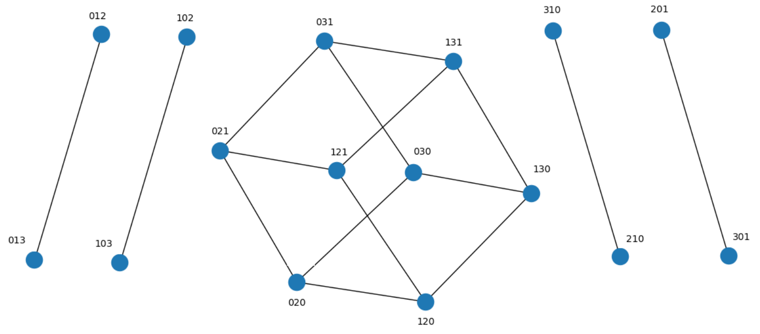

Figure 5.2: $G_2(3,1)$

Definition 17. A **path** in a graph is a consecutive sequence of edges. These edges join a sequence of nodes.

Definition 18. Let $\underline{c}_1$ and $\underline{c}_2 \in DNA(n,w)$ and $d = 2$. $G_2(n,w)$ is the graph created with $DNA(n,w)$. A **cluster** is a subgraph of $G_2(n,w)$ where $\underline{c}_1$ and $\underline{c}_2$ are in the same cluster when there is a path between the nodes of $\underline{c}_1$ and $\underline{c}_2$. The size of a cluster is defined as the number of nodes in that cluster. An isolated point is in a cluster of size 1.

It is important to notice that all the DNA words that are in the same cluster have similarities.

Definition 19. We define the **structure** of a DNA word $\underline{c} \in DNA(n,w)$ as a sequence $S(\underline{c}) = S(\underline{c})_1 S(\underline{c})_2 ... S(\underline{c})_n$ where

$$S(\underline{c})_a = \begin{cases} c_a & \textit{if } c_a \textit{ is next to its opposite} \\ L & \textit{if } c_a = 0 \vee 1 \textit{ and not next to its opposite} \\ H & \textit{if } c_a = 2 \vee 3 \textit{ and not next to its opposite} \end{cases}$$

with $1 \leq a \leq n$ and $a \in \mathbb{N}$.

Let $\underline{c} \in DNA(n,w)$ and $0 \leq x \leq n$ with $x \in \mathbb{N}$ the number of symbols of $\underline{c}$ that are not next to its opposite. Define the place of the first symbol of $\underline{c}$ that is not next to its opposite as $O_1^{\underline{c}}$, the place of the second symbol of $\underline{c}$ that is not next to its opposite as $O_2^{\underline{c}}$ etc. Thus the places of the symbols of $\underline{c}$ that are not next to its opposite are $O_1^{\underline{c}}, O_2^{\underline{c}}, ..., O_x^{\underline{c}}$.

Example 7. Let $\underline{c} = [310231] \in DNA(6,3)$. The structure of $\underline{c}$ is $H1023L$, where $x = 2$ and $O_1^{\underline{c}} = 1$ and $O_2^{\underline{c}} = 6$.

Theorem 7. Let $\underline{c}_1$,$\underline{c}_2 \in DNA(n,w)$. We have the following equivalence: $\underline{c}_1$ and $\underline{c}_2$ are in the same cluster $\iff S(\underline{c}_1) = S(\underline{c}_2)$.

Proof. $\Rightarrow$ Create the graph $G_2(n,w)$ and assume that the nodes of $\underline{c}_1$ and $\underline{c}_2$ are found in the same cluster. We prove that $S(\underline{c}_1) = S(\underline{c}_2)$.

$\underline{c}_1$ and $\underline{c}_2$ are found in the same cluster thus there is a path between $\underline{c}_1$ and $\underline{c}_2$. So $\underline{c}_1$ and $\underline{c}_2$ are connected by a consecutive sequence of edges, two nodes are connected by an edge if their distance is 1. So $\underline{c}_1$ and $\underline{c}_2$ are connected by DNA words from $DNA(n,w)$ that have distance 1 to each other.

$S(\underline{c}_1)$ is the structure of $\underline{c}_1$ and consist of symbols(0,1,2 and/or 3) and/or "H" and/or "L".

The symbols (0,1,2 and/or 3) can only change into its opposite because of the GC-weight but this is not possible because of the no run-length constraint. So all the DNA words in the same cluster as $\underline{c}_1$ have the same symbol (0,1,2 and/or 3) at the same place in their structure. Thus $S(\underline{c}_2)$ also has the same symbol (0,1,2 and/or 3) at the same place as $S(\underline{c}_1)$. The symbol that is denoted by an "L" or an "H" can only change into its opposite because of the GC-weight. Note that the "L" is used for the $0-1$ pair and "H" is used for the $2-3$ pair. So all the structures of the DNA words that are in the same cluster as $\underline{c}_1$ have an "L" at the same place and have an "H" at the same place. In conclusion because $\underline{c}_1$ and $\underline{c}_2$ are in the same cluster: $S(\underline{c}_1) = S(\underline{c}_2)$.

$\Leftarrow$ Assume that $S(\underline{c}_1) = S(\underline{c}_2)$, we prove that $\underline{c}_1$ and $\underline{c}_2$ are in the same cluster, i.e. $\underline{c}_1$ and $\underline{c}_2$ are connected by DNA words from $DNA(n,w)$ that have distance 1 to each other.

Without loss of generality we assume that the symbols at the places $O_1^{\underline{c}_1}, ..., O_y^{\underline{c}_1}$ differ from the symbols at the places $O_1^{\underline{c}_2}, ..., O_y^{\underline{c}_2}$ where $y = d(\underline{c}_1, \underline{c}_2)$. Define $\underline{c}_3 \in DNA(n,w)$ the same as $\underline{c}_1$ but with the the symbol at $O_1^{\underline{c}_1}$ changed into its opposite. Now $d(\underline{c}_1, \underline{c}_3) = 1$ so they are in the same cluster and because $S(\underline{c}_1) = S(\underline{c}_2) = S(\underline{c}_3)$: $d(\underline{c}_2, \underline{c}_3) = y-1$. Define the DNA word $\underline{c}_4 \in DNA(n,w)$ as $\underline{c}_3$ but with the symbol at $O_2^{\underline{c}_3}$ changed into its opposite. Now $d(\underline{c}_3, \underline{c}_4) = 1$ so $\underline{c}_1, \underline{c}_3$ and $\underline{c}_4$ are in the same cluster and because all the structures are the same $d(\underline{c}_2, \underline{c}_4) = y-2$. Continue this until we define a DNA word that has distance 0 to $\underline{c}_2$. Now we can conclude that $\underline{c}_1$ and $\underline{c}_2$ are in the same cluster of $G_2(n,w)$. $\square$

From Theorem 7 we know that all the DNA words in a cluster have the same structure. Let $str := S_1S_2 \dots S_n$ with $S_i \in \{0,1,2,3,L,H\}, 1 \leq i \leq n$ be a structure. Let x be the number of times $S_i \in \{L,H\}$. We characterize a cluster as:

$$Cl(str) := \{\underline{c} \in DNA(n,w) : S(\underline{c}) = str\}$$

Corollary 8. Let $\underline{c} \in DNA(n,w)$. Let x be the number of symbols of $\underline{c}$ that are not next to its opposite. Then $\underline{c}$ is found in a cluster of size 2^x.

Proof. We know that all the DNA words from the cluster of $\underline{c}$ have the same structure, we say $S(\underline{c}) = str^{\underline{c}}$. We count the number of DNA words that have the same structure as $\underline{c}$. $str^{\underline{c}}$ has x times an "L" or an "H" at the places $O_1^{\underline{c}}, O_2^{\underline{c}}, ..., O_x^{\underline{c}}$ and $n-x$ times a symbol $0,1,2$ or 3. Because there are two options for the "L", 0 or 1 and two options for the "H", 2 or 3 we can create 2^x different DNA words that have the same structure as $\underline{c}$ and thus $|Cl(str^{\underline{c}})| = 2^x$. $\square$

The DNA words in a cluster are a subset of the DNA set and every DNA word in that cluster has the same number of symbols that are not next to its opposite, x. We want to prove that for every cluster there exists a subset of size $\frac{2^x}{2}$ of that cluster, where the subset is a DNA-2 code. We also want to prove that it is not possible to have a bigger subset of the cluster that is a DNA-2 code. To prove this we use a well known bound from coding theory, the Singleton bound.

5.4 Even weight codes

We now know what the graph of the $DNA(n,w) \setminus IP(n,w,2)$ DNA words looks like and the size of its clusters. Before we can prove Theorem 2 we need one important theorem, the Singleton bound. This theorem is different than the other theorems and definitions in this thesis because we will use the binary code, in other words: binary words of length n. The size of the set containing all the binary words is 2^n [7]. Note that in this theorem there is no GC-weight constraint and no run-length constraint. The weight of a binary codeword $\underline{c}$ is defined as the number of ones in $\underline{c}$.

Definition 20. The expression $A_2(n, d)$ represents the maximum number of possible codewords in a binary code of length n and minimum distance d.

Theorem 9. The Singleton bound states that $A_2(n, d) \leq 2^{n-d+1}$.

A famous result of the singleton bound is that the maximum number of binary codewords of length n and with minimum distance 2 is equal to 2^{n-1}. Thus $A_2(n, 2) = 2^{n-1}$. For example the following two sets are binary codes with minimum distance 2 and have size 2^{n-1}.

$$C_{even} := \{\text{binary codeword } \underline{c} : \text{ weight of } \underline{c} \text{ is even}\}$$

$$C_{odd} := \{\text{binary codeword } \underline{c} : \text{ weight of } \underline{c} \text{ is odd}\}$$

5.5 Proof of Theorem 2

Our main goal is to prove Theorem 2 and we will use the result of Theorem 9 and a special mapping to do so. To explain the mapping we will look at a cluster of DNA words in a graph $G_2(n, w)$. This cluster has size 2^x (Corollary 8). The number of symbols that are not next to its opposite is x. We can map the DNA words of such a cluster to all the binary words of length x. This is done in the following way. As seen in Theorem 7 all the DNA words from a cluster have the same structure. The symbols that are next to its opposite cannot change and will be the same for every DNA word in that cluster, for this mapping we will forget about these symbols. The symbols that can change are more important. Remember that they can only change into its opposite so they are either $0 - 1$ or $2 - 3$. We map the symbols to binary symbols where a $0, 2 \mapsto 0$ and a $1, 3 \mapsto 1$. By mapping in this way we only need the structure and the mapped binary words to get the DNA words back. In Example 8 we give the mapping for the cluster with structure $01HLH$ of the graph $G_2(5, 2)$.

It is important that this mapping is bijective (surjective and injective), we first show that it is surjective. Assume we know the structure of the cluster defined as "*str*". To prove that this mapping is surjective we let $\underline{b}$ be an arbitrary binary word. From this binary word together with the structure we obtain a DNA word $\underline{c_1}$, we show that $\underline{c_1} \in Cl(str)$. The codeword $\underline{c_1}$ is in $Cl(str)$ if $S(\underline{c_1}) = str$ (Theorem 7). This is the case because we used that structure to obtain $\underline{c_1}$. Next we prove that the mapping is injective. Let $\underline{c_1}, \underline{c_2} \in Cl(str)$. Define $\underline{b_1}$ as the binary mapping of $\underline{c_1}$ and $\underline{b_2}$ as the binary mapping of $\underline{c_2}$. Assume $\underline{b_1} = \underline{b_2}$, we show that $\underline{c_1} = \underline{c_2}$. A "0" in the binary word shows that the symbol in the DNA word is a 0-symbol or a 2-symbol. A "1" in the binary word shows that the symbol in the DNA word is a 1-symbol or a 3-symbol. Because we know the structure there is always only one possibility, so $\underline{c_1} = \underline{c_2}$.

Example 8. Look at the DNA set with $n = 5$ and $w = 2$. In this example we show the way to map the DNA words of the cluster with structure $01HLH$ to a binary code of size 2^x.

The structure of this cluster is: 0 1 H L H

$$\begin{aligned}
[01313] &\mapsto [111]\\
[01303] &\mapsto [101]\\
[01213] &\mapsto [011]\\
[01312] &\mapsto [110]\\
[01302] &\mapsto [100]\\
[01212] &\mapsto [010]\\
[01203] &\mapsto [001]\\
[01202] &\mapsto [000]
\end{aligned}$$

We can use this mapping for every node in every cluster of $G_2(n, w)$. This mapping always gives us a binary code of length x (number of symbols that are not next to its opposite). From Theorem 9 we know that the even weight code or the odd weight code have maximum size. Therefore we also now that the quaternary words that map to these codes have maximum size.

Take the subset of a cluster that consists of all the DNA words that have an even (or odd) weight. The size of this subset is $\frac{2^x}{2}$. Because the mapping of these DNA-2 code have distance 2 or more to each other, this subset is a DNA-2 code. Such an subset can be created for every cluster. Combining all those subsets together with the isolated points and a DNA-2 code is created with size $\frac{B(n,w)+I_2(n,w)}{2}$. Because there is no subset of a cluster that can be of bigger size, the created DNA-2 code is of maximum size. This completes the proof of Theorem 2.

6 Conclusions and recommendations

In this chapter, a short summary is given of the research done in this thesis. We also discuss what conclusions can be deduced from the research done. Lastly some recommendations for future work are discussed.

6.1 Conclusion

A DNA-d code is defined in Chapter 3. This code takes three parameters, the length n of the DNA codewords, the GC-weight w of the DNA codewords and the minimum distance of the code. The maximum size of such a code is defined as $B_d(n, w)$. The aim of this thesis is to improve the current lower bound on $B_d(n, w)$. The current lower bound is described in the paper "Family of Constrained Codes for Archival DNA Data Storage" by Limbachiya et. al. [1]

In Chapter 4, three algorithms to get a lower bound on $B_d(n, w)$ were discussed. Algorithm 1 was the algorithm given in the paper of Limbachiya et. al. [1] and two additional algorithms were given with the aim to improve the lower bound given by Algorithm 1.

To analyse these algorithms we looked at the lower bound computed with the three algorithms for $2 \leq n \leq 11$, $w = \lfloor \frac{n}{2} \rfloor$ and $2 \leq d \leq 5$. The lower bounds are given in Table B.1. From these results we can draw the conclusion that Algorithm 2 improved the lower bound on $B_d(n, w)$ of Limbachiya et. al. [1] for mostly all of the parameters. The exceptions were:

- For minimum distance equal to 2 the lower bound computed by all three algorithms for all lengths was equal.
- For minimum distance 4 of both length 4 and 5 the computed bounds of all three algorithms were equal.
- For minimum distance 4 of length 6 the highest lower bound was computed by algorithm 3. The lower bound computed by Algorithm 2 still improved the lower bound given by Algorithm 1.

A DNA-d code with the best lower bound as its size can be used to store archival data. More data can be stored using such a DNA-d code when compared to a DNA-d code designed by Limbachiya et. al. This gets us a little step closer to implementing DNA-based data storage.

The goal to improve the current lower bound is reached for most cases. For $B_2(n, w)$ we have not improved it. This is because the computed lower bound is $B_2(n, w)$. The formula to get this value is given in Chapter 5. The maximum size for a DNA-2 code is given at the end of this section. $B(n, w)$ is the size of the DNA set and $I_2(n, w)$ is the number of isolated points of the DNA set.

In conclusion, the current lower bound is almost always improved for the parameters $2 \leq n \leq 11$, $w = \lfloor \frac{n}{2} \rfloor$ and $2 \leq d \leq 5$. We also conclude that the maximum size of a DNA-2 code is:

$$B_d(n, w) = \frac{B(n, w) + I_2(n, w)}{2}$$

6.2 Recommendations

The aim of this thesis was to improve the lower bound on $B_d(n, w)$ given by Limbachiya et. al. [1]. Multiple remarks can be made about this thesis and about future research.

In this thesis two constraints have been taken into account. Some adjustments can be made to these constraints. First the no run-length constraint will be discussed. In this thesis the decision was made to not allow two repeated symbols in a DNA word. This lowers the chance for an error to occur in a polymer run. When research has been done on these polymer runs, it might conclude that it is also sufficient to allow small polymer runs. When this constraint changes the DNA set will contain more DNA words. This results in a different DNA-d code with a different maximum size. More research has to be done on this subject to know what the effect of this would be on the DNA-d code.

As explained in [4], the chance of an error to occur decreases when the GC-rate is around 50%. When the GC-weight is half of the length the DNA set only takes parameter. This is an interesting subject for further research. Table B.3 shows the lower bounds computed with Algorithm 1, 2 and 3 for $w = \frac{n}{2}$ and $3 \leq d \leq 5$. Unfortunately, from this table we can not immediately propose a conjecture for $B_d(n, \lfloor \frac{n}{2} \rfloor)$. Finding a formula for $B_d(n, \lfloor \frac{n}{2} \rfloor)$ brings us a step closer to finding a formula for $B_d(n, w)$. When researching these specific parameters a conjecture might be purposed.

The aim of this thesis was to improve the lower bound on $B_d(n, w)$. Using algorithm 2 and 3 we have reached our goal but it is unknown how tight this lower bound is. It might be possible to give an indication of how tight this lower bound is. An upper bound for $B_d(n, w)$ is $B(n, w)$. Further research on a better upper bound might give more information about this subject.

It is possible that there is no formula for $B_d(n, w)$ in general. When the exact value for $B_d(n, w)$ is unknown it is of great value to know if an algorithm that computes a DNA-d code is always better than an other algorithm. We do not know which one of the three algorithms computes the biggest DNA-d code for the parameters that were not examined. We also do not know why Algorithm 2 performs better, most of the time, for the examined parameters. When the length of the DNA words get longer the time it takes for the algorithms to find a DNA-d code get very long. Because of time restrictions it might not be possible to try all three algorithms. Further research on these algorithms is recommended. Further research to find a different algorithm is also advised.

The exact value for $B_d(n, w)$ is still unknown but maybe there are formulas for specific parameters. In this thesis we have already researched this for $d = 2$. The case where $n = d$ is also recommended to research. For the case $n = d$, Table B.2 in Appendix B gives the lower bounds for the case $n = d$ computed with the algorithms from Chapter 4. We did not find a conjecture for the case $n = d$. Further research needs to be done and a conjecture might be purposed.

DNA-based data storage has the ability of becoming a state of the art method that could change the world of archival data storage. But before DNA-based data storage is implemented more research has to be done on this subject.

Bibliography

[1] D. Limbachiya, M.K. Gupta and V. Aggarwal, “Family of Constrained Codes for Archival DNA Data Storage”, *IEEE commun. lett.*, vol. 22, n. 10, pp 1972-1975, Oct 2018.

[2] J. Bornholt, R. Lopez, D. M. Carmean, L. Ceze, G. Seelig, and K. Strauss, “A DNA-Based Archival Storage System,” *IEEE Micro*, vol. 37, no. 3, pp. 98-104, 2016.

[3] S. M. H. T. Yazdi, H. M. Kiah, E. Garcia-Ruiz, J. Ma, H. Zhao, and O. Milenkovic, “DNA-based storage: Trends and methods”, *IEEE Trans. Mol. Biol. Multi-Scale Commun.*, vol. 1, no. 3, pp. 230–248, Sep. 2015.

[4] Y. Erlich and D. Zielinski, “DNA fountain enables a robust and efficient storage architecture,” *Science*, vol. 355, no. 6328, pp. 950–954, 2017.

[5] D. Limbachiya, B. Rao and M. K. Gupta, (2016).“The Art of DNA Strings: Sixteen Years of DNA”, [online]. Available: https://arxiv.org/abs/1607.00266

[6] Eva Slingerland, “DNA Data Storage using Hamming and Reed-Solomon Codes”, *University of Technology Delft*, 2019

[7] D.R. Hankerson, D.G. Hoffman, D. A. Leonard, C. C. Lindner, K. T. Phelps, C. A. Rodger, J. R. Wall, *Coding Theory And Cryptography, the essentials.* Auburn, Alabama, Auburn University.

[8] K. A. Schouhamer Immink and K. Cai, “Design of Capacity-Approaching Constrained Codes for DNA-Based Storage Systems”, *IEEE Communications Letters*, vol. 22, pp. 224-227, 2018.

[9] J. H. Conway and N. J. A. Sloane, “Lexicographic Codes: Error-Correcting Codes from Game Theory”, *IEEE Trans. on inf. Theory*, vol. 32, No 3, pp. 337, May 1986

Python code

```
import scipy as sp
from scipy import special
import math
import itertools
import numpy as np

def has_runlength(word):
    #function that checks if a word has run-length
    return any(word[i]==word[i+1] for i in range(len(word)-1))

def get_weight(word):
    #gets the GC-weight of a word
    return word.count(2) + word.count(3)

def get_distance(word1,word2):
    #determine the distance between two words
    return len([i for i in range(len(word1)) if word1[i]-word2[i] != 0])

def get_words_in_sphere(code,codeword, d):
    #get all the (d-1)-neighbours of a codeword
    return [w for w in code if 0<get_distance(codeword,w)<d]

def get_maximum(distances):
    #maximum is the key with the max value of the dictionary distances
    maximum = max(distances, key=distances.get)
    return maximum

def get_minimum(distances):
    #minimum is the key with the min value of the dictionary distances
    minimum = min(distances, key=distances.get)
    return minimum

def get_minimum_distance(code):
    #returns the minimum distance of a code
    min_distance = 10000
    for codeword1 in code:
        for codeword2 in code:
            if codeword1 != codeword2:
                distance = get_distance(codeword1,codeword2)
                if min_distance > distance:
                    min_distance = distance
    return min_distance
```

A.1 Generating DNA(n,w)

```
#This function returns the DNA list, the words are generated lexicographical.
#A word is added to the list if it satisfies the no run-length constraint
#and the fixed GC-weight.
def DNA(n,w):
    Qcode = list(itertools.product(range(4),repeat=n))
    DNAcode = [codeword for codeword in Qcode if
        not has_runlength(codeword) and get_weight(codeword)==w]
    return DNAcode
```

A.2 Algorithm 1

```
#Using a list of DNA words a dictionary is created with the DNA words as keys
#and its (d-1)-neighbours as value.
def alg1_step2_list(code,d):
    words_in_sphere = {codeword: get_words_in_sphere(code,codeword,d)
                        for codeword in code}
    return words_in_sphere

#Create a dictionary with the DNA words as keys and the number of
#(d-1)-neighbours as value
def alg1_step2_dist(words_in_sphere):
    distances = {key:len(value) for key,value in words_in_sphere.items()}
    return distances


#The following function defines the maximum, deletes it from the dictionaries
#and reduces the value of its (d-1)-neighbours by one.
def alg1_step3(distances, words_in_sphere):
    maximum = get_maximum(distances)
    del distances[maximum]
    for value in words_in_sphere[maximum]:
        distances[value] -= 1
        words_in_sphere[value].remove(maximum)
    del words_in_sphere[maximum]

#While the dictionary "distances" is not empty we keep calling alg1_step3
def alg1_step4(distances, words_in_sphere):
    while max(distances.values())>0:
        alg1_step3(distances,words_in_sphere)
    return len(words_in_sphere)
```

```
#This function combines all the steps from above to create a DNA-d code.
#If the lower bound is needed change the last line in: "return len(step4)"
def alg1(n,w,d):
    DNAcode = DNA(n,w)
    step2list = alg1_step2_list(DNAcode,d)
    step2dist = alg1_step2_dist(step2list)
    step3 = alg1_step3(step2dist,step2list)
    step4 = alg1_step4(step2dist,step2list)
    return step4
```

A.3 Algorithm 2

```
#Using a list of DNA words a dictionary is created with the DNA words as keys
#and its (d-1)-neighbours as value.
def alg2_step2_list(code,d):
    words_in_sphere = {codeword: get_words_in_sphere(code,codeword,d)
                       for codeword in code}
    return words_in_sphere

#Create a dictionary with the DNA words as keys and the number of
#(d-1)-neighbours as value
def alg2_step2_dist(words_in_sphere):
    distances = {key:len(value) for key,value in words_in_sphere.items()}
    return distances

#The following function defines the minimum, adds this minimum to a list
#"code", deletes minimum and all its (d-1)-neighbours from the dictionaries
#and reduces the value of all (d-1)-neighbours of the deleted words by one.
def alg2_step3(distances,words_in_sphere,code):
    minimum = get_minimum(distances)
    code.append(minimum)
    del distances[minimum]
    for value in words_in_sphere[minimum]:
        del distances[value]
        for val in words_in_sphere[value]:
            if val in distances:
                words_in_sphere[val].remove(value)
                distances[val]-=1
        del words_in_sphere[value]
    del words_in_sphere[minimum]
    return code

#While the dictionary "words_in_sphere" is not empty we keep calling alg2_step3
def alg2_step4(distances, words_in_sphere,code):
    while words_in_sphere:
        alg2_step3(distances,words_in_sphere,code)
    return code
```

```
#This function combines all the steps from above to create a DNA-d code.
#If the lower bound is needed change the last line in: "return len(step4)"
def alg2(n,w,d):
    DNAcode = DNA(n,w)
    step2list = alg2_step2_list(DNAcode,d)
    step2dist = alg2_step2_dist(step2list)
    DNAdistance = []
    step3 = alg2_step3(step2dist,step2list,DNAdistance)
    step4 = alg2_step4(step2dist,step2list,step3)
    return step4
```

A.4 Algorithm 3

```
#define the first word from the DNA list and add it to
#the list "DNAcode" unless it breaks the minimal distance.
def alg3_step1(distancecode,DNAcode,d):
    word = DNAcode[0]
    if all(get_distance(word,codeword)>=d for codeword in distancecode):
        distancecode.append(word)
    DNAcode.remove(word)
    return distancecode, DNAcode

#keep calling alg3_step1 untill the list "DNAcode" is empty.
def alg3_step2(distancecode,DNAcode,d):
    while DNAcode:
        alg3_step1(distancecode,DNAcode,d)
    return distancecode

#This function combines the functions above to create a DNA-d code.
#If the lower bound is needed change the last line in: "return len(step2)"
def alg3(n,w,d):
    DNAcode = DNA(n,w)
    DNAdistance = []
    step1a,step1b = alg3_step1(DNAdistance,DNAcode,d)
    step2 = alg3_step2(step1a,step1b,d)
    return step2
```

A.5 The formula for *B(n,w)*

```
#This formula is from the paper of Limbachiya et. al. Part 1 calculates the
#first part of the formula and part 2 the second.
#To get B(n,w) these parts are added.
def B(n,w):
    v = min(w,n-w)
    part1 = 0
    part2 = 0
    for y in range(v):
        part1 = part1+2**(2*v+1-2*y)*sp.special.binom(v-1,y)
```

```
            *sp.special.binom(n-v,v-y)
    for y in range(v-1):
        part2 = part2 + 2**(2*v-1-2*y)*sp.special.binom(v-1,y)
            *sp.special.binom(n-v-1,v-y-2)
    return part1+part2
```

A.6 The recursive formula for the size of the DNA set

```
#The function below calculates the value for B(n,w,L).
def BL(n,w):
    if w==0:
        return 2
    if w==n:
        return 0
    else:
        return BL(n-1,w)+2*BH(n-1,w)

#The function below calculates the value for B(n,w,H).
def BH(n,w):
    if w==0:
        return 0
    if n==w:
        return 2
    else:
        return 2*BL(n-1,w-1)+BH(n-1,w-1)

#To get B(n,w) the values for B(n,w,L) and B(n,w,H) are added
def Brec(n,w):
    print("Low=", BL(n,w))
    print("High=", BH(n,w))
    return BL(n,w)+BH(n,w)
```

A.7 The recursive formula for the number of isolated points

```
#The function below calculates the value for I_2(n,w,L). First the
#starting conditions are given, after the recursive part of the function.
def IL(n,w):
    if n!=1 and w==0:
        return 2
    if n==1 or n==w or w==1 or w==n-1:
        return 0
    else:
        return IL(n-1,w)+2*IH(n-2,w)

#The function below calculates the value for I_2(n,w,H). First the
#starting conditions are given, after the recursive part of the function.
def IH(n,w):
    if n==1 or w==0 or w==1 or w==n-1:
        return 0
    if n!=1 and n==w:
        return 2
    else:
        return 2*IL(n-2,w-2)+IH(n-1,w-1)

#To get I_2(n,w) the values for I_2(n,w,L) and I_2(n,w,H) are added
def Irec(n,w):
    print("Low=",IL(n,w))
    print("High=",IH(n,w))
    return IL(n,w)+IH(n,w)
```

B Tables

B.1 Tables: Lower bounds for maximum size of DNA-d code

		$d = 2$			$d = 3$			$d = 4$			$d = 5$		
n	$B(n, w)$	1	2	3	1	2	3	1	2	3	1	2	3
2	8	4	4	4	-	-	-	-	-	-	-	-	-
3	16	8	8	8	2	3	2	-	-	-	-	-	-
4	56	32	32	32	11	12	7	4	4	4	-	-	-
5	128	68	68	68	17	18	18	7	7	7	2	3	2
6	424	216	216	216	44	53	45	16	20	21	6	7	6
7	1040	528	528	528	110	119	101	36	44	37	11	15	12
8	3352	1704	1704	1704	289	326	286	86	127	97	29	37	29
9	8576	4336	4336	4336	662	762	687	199	227	216	59	77	67
10	27208	13688	13688	13688	1810	2126	1939	525	618	557	141	180	164
11	71568	35936	35936	35936	4320	5145	4666	1235	1426	1323	284	389	349

Table B.1: Lower bounds for $B_d(n, w)$ computed with the 3 algorithms from Chapter 4. We use $w = \lfloor \frac{n}{2} \rfloor$

	$d = n = 3$			$d = n = 4$			$d = n = 5$			$d = n = 6$			$d = n = 7$		
w	1	2	3	1	2	3	1	2	3	1	2	3	1	2	3
1	2	3	2	2	2	2	2	2	2	2	2	2	2	2	2
2	2	3	2	4	4	4	2	3	2	2	3	2	2	2	2
3	2	2	2	2	2	2	2	3	2	4	4	4	2	3	2
4	-	-	-	2	2	2	2	2	2	2	3	2	2	3	2
5	-	-	-	-	-	-	2	2	2	2	2	2	2	2	2
6	-	-	-	-	-	-	-	-	-	2	2	2	2	2	2
7	-	-	-	-	-	-	-	-	-	-	-	-	2	2	2

Table B.2: Size of the generated DNA-distance codes by Algorithms 1,2 and 3 for $d = n$ and $1 \leq w \leq n$

B.2 Tables: the size of the DNA set and the number of isolated points

	w=0			w=1			w=2			w=3			w=4			w=5		
n	L	H	T	L	H	T	L	H	T	L	H	T	L	H	T	L	H	T
1	2	0	2	0	2	2	-	-	-	-	-	-	-	-	-	-	-	-
2	2	0	2	4	4	8	0	2	2	-	-	-	-	-	-	-	-	-
3	2	0	2	12	4	16	4	12	16	0	2	2	-	-	-	-	-	-
4	2	0	2	20	4	24	28	28	56	4	20	24	0	2	2	-	-	-
5	2	0	2	28	4	32	84	44	128	44	84	128	4	28	32	0	2	2
6	2	0	2	36	4	40	172	60	232	212	212	424	60	172	232	4	36	40
7	2	0	2	44	4	48	292	76	368	636	404	1040	404	636	1040	76	292	368
8	2	0	2	52	4	56	444	92	536	1444	660	2104	1676	1676	3352	660	1444	2104
9	2	0	2	60	4	64	628	108	736	2764	980	3744	5028	3548	8576	3548	5028	8576
10	2	0	2	68	4	72	844	124	968	4724	1364	6088	12124	6508	18632	13604	13604	27208
11	2	0	2	76	4	80	1092	140	1232	7452	1812	9264	25140	10812	35952	40812	30756	71568
12	2	0	2	84	4	88	1372	156	1528	11076	2324	13400	46764	16716	63480	102324	61092	163416
13	2	0	2	92	4	96	1684	172	1856	15724	2900	18624	80196	24476	104672	224508	110244	334752
14	2	0	2	100	4	104	2028	188	2216	21524	3540	25064	129148	34348	163496	444996	184868	629864

Table B.3: The values of $B(n,w,L)$ indicated by "L", $B(n,w,H)$ indicated by "H" and $B(n,w)$ indicated by "T" for words with length n and CG-weight w

	w=0			w=1			w=2			w=3			w=4			w=5			w=6		
n	L	H	T	L	H	T	L	H	T	L	H	T	L	H	T	L	H	T	L	H	T
1	0	0	0	0	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
2	2	0	2	0	0	0	0	2	2	-	-	-	-	-	-	-	-	-	-	-	-
3	2	0	2	0	0	0	0	0	0	0	2	2	-	-	-	-	-	-	-	-	-
4	2	0	2	0	0	0	4	4	8	0	0	0	0	2	2	-	-	-	-	-	-
5	2	0	2	0	0	0	4	4	8	4	4	8	0	0	0	0	2	2	-	-	-
6	2	0	2	0	0	0	12	4	16	4	4	8	4	12	16	0	0	0	0	2	2
7	2	0	2	0	0	0	20	4	24	12	4	16	4	12	16	4	20	24	0	0	0
8	2	0	2	0	0	0	28	4	32	20	4	24	28	28	56	4	20	24	4	28	32
9	2	0	2	0	0	0	36	4	40	28	4	32	52	44	96	44	52	96	4	28	32
10	2	0	2	0	0	0	44	4	48	36	4	40	108	60	186	84	84	168	60	108	168
11	2	0	2	0	0	0	52	4	56	44	4	48	196	76	272	188	116	304	116	188	304
12	2	0	2	0	0	0	60	4	64	52	4	56	316	92	408	356	148	504	332	332	664
13	2	0	2	0	0	0	68	4	72	60	4	64	468	108	576	588	180	768	708	540	1248
14	2	0	2	0	0	0	76	4	80	68	4	72	652	124	776	884	212	1096	1372	812	2184

Table B.4: The values of $I_2(n, w, L)$ indicated by "L", $I_2(n, w, H)$ indicated by "H" and $I_2(n, w)$ indicated by "T" for words with length n and CG-weight w