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

Constrained Single-Error-Detecting codes for DNA-based Storage Systems

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

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

for the degree

BACHELOR OF SCIENCE
in
APPLIED MATHEMATICS

by

H. Vermeer

Delft, Nederland
February 2021

BSc thesis APPLIED MATHEMATICS

"Constrained Single-Error-Detecting Codes for DNA-Based Storage Systems"

(Dutch title: "Beperkte Enkel-Fout-Detecterende Codes voor DNA-Gebaseerde Opslagsystemen"

H.Vermeer

Delft University of Technology

Thesis committee

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

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

Drs E.M. van Elderen

February, 2021 Delft

Abstract

The amount of data being produced is growing exponentially [1]. An important challenge is to find methods to store this data efficiently and in an environmentally friendly way. One idea that is a growing research topic involves using synthetic DNA. DNA has the potential to be more efficient and environmentally friendly than current methods. DNA is made of a sequence of four nucleotides, Adenine (A), Cytosince (C), Guanine (G), and Thymine (T). To store data, DNA strands can be created with specific nucleotide sequences. In the process of reading and storing data substitution errors can occur. Two constraints are introduced to minimise the number of errors. The GC-weight constraint which states that every DNA sequence must have a fixed number of G and C nucleotides, and the runlength constraint, which states the maximum number of repeating nucleotides possible in every DNA sequence.

In this thesis the maximum number of DNA sequences $B_r(n, w, d)$ of length n that satisfy the runlength constraint r, GC-weight constraint w and minimum (Hamming) distance d is investigated for $r > 1$ and $d = 2$. The research continues the work of Van Leeuwen [2] and Weber et al. [3] where the case was settled for $r = 1$ and $d = 2$. Six algorithms are used to obtain lower bounds for this maximum number of DNA sequences. Further lower bounds and an upper bound for $B_r(n, w, 2)$ are obtained or proven after considering a construction of DNA sequences containing words with specific properties. Two of the bounds result in a narrow range of possible values for $B_r(n, w, 2)$.

Contents

1 Introduction

1.1 Motivation

In the last hundred years society has adapted to the ability to store large amounts of information as data. Currently data is stored on magnetic and optical storage devices that use on and off switches to represent bits. Storing data in this way has a density of about 100 GB/mm^3 and a durability of 30 years [4]. While this is sufficient at the moment, the amount of data is growing fast [1] and research is being done into more efficient and environmentally friendly methods. One of those methods uses synthetic DNA which is a potential solution to both issues. With the development of DNA technologies, the use of DNA as information storage has become a popular research topic.

Among other potential solutions DNA is unique in its high storage capacity and structural integrity. A human cell can hold 6.4 GB of information and DNA has been recovered from species extinct for more than 10,000 years [5]. Using DNA for data storage has the potential to solve many of the issues currently surrounding electronic data storage. This thesis focuses on the sequences of nucleotides that can be used to store information.

1.2 Research question

In [6] and [7] two constraints are explained that are important to consider when storing data using DNA. A code consisting of DNA codewords is introduced which applies these constraints. The constraints involve the maximum number of repeated nucleotides, and the fixed number of G and C nucleotides. If the code also has minimum Hamming distance of 2 it can detect single substitution errors. Every code has a size and therefore there exists a DNA code of maximum size which has minimum distance 2. This maximum size has been settled for the case where no repeated nucleotides are allowed [3]. However the maximum size is still unknown if repetitions are permitted. This thesis addresses the question "What upper bounds and lower bounds exist for DNA codes of maximum size with minimum distance 2". Several algorithms are given which compute a DNA code with minimum distance 2 providing lower bounds for the maximum size. A more structured method to construct such a DNA code is also given which results into other lower bounds as well as an upper bound for the maximum size of a DNA code with minimum distance 2.

1.3 Organisation of the thesis

This section gives a description of the remaining Chapters in this thesis.

Chapter 2: Prerequisites. The first half of this chapter provides some basic knowledge about DNA and coding theory that will be used throughout this thesis. The second half explains two constraints that are important in DNA-based storage and defines DNA sets and DNA codes.

Chapter 3: Six Algorithms to construct a DNA code with minimum distance 2. In Section 3.1 three adjusted algorithms from [2] are given. In Section 3.2 three additional algorithms are conceived that compute a DNA code. The last section evaluates the algorithms.

Chapter 4: Parity Symbol Construction. In this Chapter the main research of this thesis is presented. A construction from binary codes is applied to the DNA case. This construction results in a conjecture about the maximum number of words in a DNA code with minimum distance 2. In Section 4.1 the construction is introduced with two examples. In Sections 4.2 and 4.3 the construction is formally defined. In Sections 4.4 and 4.5 the results from the construction are evaluated.

Chapter 5: Upper and Lower bounds for $B_r(n, w, 2)$ This chapter summarises the upper bounds and lower bounds obtained for $B_r(n, w, 2)$. In Section 5.1 an additional lower bound is proven. In Section 5.2 lower bounds and an upper bound obtained as a result of Chapter 4 are discussed. In Section 5.3 the bounds are compared resulting in a narrow range of possible values for $B_r(n, w, 2)$.

Chapter 6: Conclusions and Future Research. This chapter concludes the results of the preceding Chapters.

2 Prerequisites

To understand how DNA is used in data storage we first give a summary of how the structure of DNA can be useful. We then explain some basic concepts about coding which are used to define DNA sets and DNA codes.

2.1 What is DNA?

Deoxyribonucleic acid, commonly known as DNA, is a molecule which contains the biological information of an organism and is thus found in all aspects of life. The molecule is composed of two strands which carry the genetic information in the form of a linear sequence of four basic blocks called nucleotides [4]. There are four possible nucleotides, Adenine (A), Thymine (T), Guanine (G) and Cytosine (C). The two strands are connected in a helical structure with every nucleotide in one strand connected to a nucleotide in the other strand, Adenine with Thymine and Guanine with Cytosine. The order in which the nucleotides form the sequence determines what information is stored, much like the order of zeros and ones determines the information stored on a computer. This is the basis for using DNA as data storage.

To store information a specific sequence of nucleotides is created using DNA synthesising. This sequence can then be read and copied using a method called DNA sequencing. Using these two methods DNA can be used to store, read and copy information. However, the techniques used can lead to errors in the sequence of nucleotides, which means the stored sequence is not the same as the sequence after reading the DNA strand.

2.2 Basic coding concepts

In this section some of the fundamental concepts of coding theory are stated. In the context of DNA the theory can be applied to the quarternary alphabet as every nucleotide can be paired with a number. In this thesis the nucleotides are represented by numerical symbols. The notation used follows the notation Weber et al. [3]

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

Definition 1. A **word** of **length** n is defined as $\mathbf{x} = (x_1, x_2, \ldots, x_n)$ with $x_i \in \{0, 1, 2, 3\}$ for $1 \leq i \leq n$. x_1 is called the first symbol, x_2 the second, etc.

Definition 2. A **code** is a set $\mathcal{C}$ of words. The words in a code are called **codewords**. A block code is a code where all words have the same length. The size of the code is the number of words in the code denoted by $|\mathcal{C}|$.

In this thesis only block codes are considered.

Example 1. Let $\mathcal{C} = \{(0,0), (0,1), (1,2), (2,3), (3,3)\}$. Then the size of this code is $|C| = 5$ and the length is $n = 2$.

Definition 3. The **weight** of word $\mathbf{x}$ denoted $wt(\mathbf{x})$ is defined as the sum of the symbols of $\mathbf{x}$:

$$wt(\mathbf{x}) = \sum_{i=1}^{n} x_i \, .$$

Definition 4. A word is called a word with **even weight** if $wt(\mathbf{x})$ is an even integer and a word with **odd weight** if $wt(\mathbf{x})$ is an odd integer.

Example 2. $\mathbf{x} = (0, 1, 2, 3)$ has $wt(\mathbf{x}) = 6$ making it a word with even weight.

As explained in the previous section errors can occur in DNA sequences. To detect and correct potential errors an important concept called the minimum distance is used.

Definition 5. Let $\mathbf{x}$ and $\mathbf{y}$ be words of the same length n. The **(Hamming) distance** between $\mathbf{x}$ and $\mathbf{y}$ is the number of positions at which the corresponding symbols are different, denoted by $d(\mathbf{x}, \mathbf{y})$

Definition 6. The **minimum (Hamming) distance** of a code $\mathcal{C}$, denoted by d, is the smallest distance between any two different words $\mathbf{x}$ and $\mathbf{y}$ in $\mathcal{C}$:

$$d = min\{d(\mathbf{x}, \mathbf{y}) : \mathbf{x}, \mathbf{y} \in \mathcal{C}, \mathbf{x} \neq \mathbf{y}\}.$$

The minimum distance is an important property of a code because it limits the number of words possible such that errors can be detected and corrected. If one error occurs in a word from a code with minimum distance $d = 2$, then the word is not in the code and the error is detected. Similarly using a code with minimum distance d means up to $d - 1$ errors are always detected. Additionally, a code with minimum distance d can always correct up to $\lfloor \frac{d-1}{2} \rfloor$ errors [8].

Example 3. Let $\mathcal{C} = \{(0, 0, 0, 0), (1, 1, 1, 1), (2, 2, 2, 2), (3, 3, 3, 3)$. Then the minimum distance of $\mathcal{C}$ is $d = 4$. So this code can detect up to 3 errors and correct 1 error.

For more information on the properties of codes we refer to [8].

2.3 Constraints on DNA-based codes

In the context of DNA, previous studies [5][6] have shown that there are two main causes of errors during DNA synthesising and sequencing: homopolymer runs and the GC-weight. In this section two constraints are explained that are used to reduce the number of occuring errors.

A homopolymer run in a strand of nucleotides is a repetition of nucleotides within the strand. When reading a strand every nucleotide is read as a colour signal which means an error is more likely to occur when multiple neighbouring nucleotides are the same. The runlength constraint restricts the maximum number of repeating nucleotides allowed in a word.

Definition 7. The **maximum runlength** of a word $\mathbf{x}$, denoted by $r(\mathbf{x})$, is the maximum number of repeated symbols in $\mathbf{x}$:

$$r(\mathbf{x}) = max\{r : \exists i \text{ such that } x_i = x_{i+1} \cdots = x_{i+r-1}\}.$$

The number of G and C nucleotides in a word also called the GC-weight is the other important source of errors. The chance of errors occurring is higher in words with a GC-weight higher than 60% or lower than 40%. For this reason the GC-weight constraint is introduced.

Definition 8. The **GC-weight** of a word $\mathbf{x}$, denoted by $w(\mathbf{x})$, is the number of symbols in $\mathbf{x}$ that are equal to 2 or 3:

$$w(\mathbf{x}) = |\{i : x_i \in \{2, 3\}\}|.$$

Example 4. Let $\mathbf{x} = (0, 1, 1, 3, 2, 2, 2, 1)$. Then $r(\mathbf{x}) = 3$ since the largest number of repeated symbols is the three 2's, and $w(\mathbf{x}) = 4$.

Definition 9. The symbols 0 and 1 are called **opposite symbols** of each other and 2 and 3 are opposite symbols of each other.

2.4 The DNA set

With the knowledge from the previous sections we can now define DNA sets that satisfy given parameters and constraints. Considering the set of all words with length n, a subset of this set contains all the words that also have GC-weight w.

Definition 10. $\mathcal{B}(n, w)$ is the **DNA set** of all words with length n and GC-weight w.

A subset of $\mathcal{B}(n, w)$ contains all such words that have no more than r repeated symbols.

Definition 11. We define $\mathcal{B}_r(n, w)$ as the **DNA set** of all words that have length n, GC-weight w and maximum runlength r:

$$\mathcal{B}_r(n, w) = \{\mathbf{x} : \mathbf{x} \text{ has length } n \wedge w(\mathbf{x}) = w \wedge r(\mathbf{x}) \leq r\}.$$

Its cardinality is denoted by $|\mathcal{B}_r(n, w)| = B_r(n, w)$.

Remark: $\mathcal{B}_1(n, w) \subseteq \mathcal{B}_2(n, w) \subseteq \cdots \subseteq \mathcal{B}_n(n, w) = \mathcal{B}_{n+1}(n, w) = \cdots = \mathcal{B}(n, w)$

This set of words can always be generated by taking the entire set of words with length n and removing the words that do not meet the requirements. The Python code for generating the set can be found in Appendix A.1

Example 5. Let us generate the set $\mathcal{B}_1(3, 1)$. This is the set of all words with length $n = 3$, GC-weight $w = 1$ and maximum runlength $r = 1$. This means that every word has one 2 or one 3 and has no repeating symbols.

$$\begin{aligned}\mathcal{B}_1(3, 1) = \{&(0, 1, 2), (0, 1, 3), (0, 2, 0), (0, 2, 1), (0, 3, 0), (0, 3, 1), (1, 0, 2), (1, 0, 3),\\ &(1, 2, 0), (1, 2, 1), (1, 3, 0), (1, 3, 1), (2, 0, 1), (2, 1, 0), (3, 0, 1), (3, 1, 0)\}.\end{aligned}$$

The number of words in this set is $B_2(3,1) = 16$. Any other combination of symbols either does not have length 3, or does not satisfy the GC-weight or runlength constraint so it is not in this set.

The number of words in the set $\mathcal{B}_r(n,w)$ are counted in Theorem 1 from [3]. The result is a recursive formula involving the subset of all words that do not end with a 0:

$$\mathcal{N}_r(n,w) = \{\mathbf{x} \in \mathcal{B}_r(n,w) : x_n \neq 0\}.$$

The number of words for which this is the case is denoted by $N_r(n,w)$. The following property is also stated in [3] which will be used in Section 4.3:

$$N_r(n,w) = |\{\mathbf{x} \in \mathcal{B}_r(n,w) : x_n \neq 0\}| = |\{\mathbf{x} \in \mathcal{B}_r(n,w) : x_n \neq 1\}|.$$

Theorem 1. *For $0 \leq w \leq n$ and $r \geq 1$, it holds that*
$N_r(0,0) = 1$,

$$N_r(n,w) = 2^{n-1}\binom{n-1}{w} + 2^n\binom{n-1}{w-1} \tag{1}$$

if $1 \leq n \leq r$,

$$N_r(n,w) = \sum_{j=1}^{\min\{r,n-w\}} N_r(n-j,w) + 2\sum_{j=1}^{\min\{r,w\}} N_r(n-j,n-w) \tag{2}$$

if $n > r$, and

$$B_r(n,w) = \sum_{j=0}^{\min\{r,n-w\}} N_r(n-j,w). \tag{3}$$

Table 1: $B_r(n,w)$ from Theorem 1 with $w = \lfloor \frac{n}{2} \rfloor$.

	$B_r(n,w)$			
n	$r=1$	$r=2$	$r=3$	$r=4$
2	8	8	8	8
3	16	24	24	24
4	56	96	96	96
5	128	296	320	320
6	424	1160	1280	1280
7	1040	3784	4416	4480
8	3352	14696	17608	17920
9	8576	49392	62408	64352
10	27208	190848	248360	257288

To reduce the number of errors that occur the DNA set satisfies the GC-weight and runlength constraints. To increase the robustness of the set a minimum distance property is included. A general research challenge is determining a largest possible subset of $\mathcal{B}_r(n,w)$ that has minimum distance d.

Definition 12. A set $\mathcal{C}$ is called a **DNA code** if $\mathcal{C} \subseteq \mathcal{B}_r(n, w)$ and $d(\mathcal{C}) \geq 2$.

Definition 13. We define $B_r(n, w, d)$ as the number of words in a largest possible DNA code with length n, GC-weight w and maximum runlength r with minimum distance d,

$$B_r(n, w, d) = \max\{|\mathcal{C}| : \mathcal{C} \subseteq \mathcal{B}_r(n, w) \wedge d(\mathcal{C}) = d\}.$$

Remark: $B_r(n, w, 1) = B_r(n, w)$

Example 6. We know the set $\mathcal{B}_1(3, 1)$ from Example 5. By deleting words we can obtain minimum distance 2. For example the distance between the first two words $d((0, 1, 2), (0, 1, 3)) = 1$. This means we can never have both words in the set if we want minimum distance 2, so we delete the word $(0, 1, 3)$. Let us assume after deleting 12 words that the following code is obtained:

$$\{(0, 1, 2), (0, 2, 0), (1, 3, 0), (3, 1, 1)\}.$$

This code is a DNA code with minimum distance 2 with size 4, and so $B_1(3, 1, 2) \geq 4$. It is not of maximum size. In the next Chapter we show a larger code can be constructed with the same parameters.

We note that there are many ways of constructing DNA codes with minimum distance d and we will outline some in the next Chapter. Also there may be multiple DNA codes with size $B_r(n, w, d)$.

In [2] the case for $r = 1$ has been investigated and in [2] and [3] a formula for $B_1(n, w, 2)$ is stated and proven. Here we investigate $B_r(n, w, 2)$ with a relaxed runlength constraint i.e. $r > 1$.

3 Six algorithms to construct a DNA code

To find a lower bound for $B_r(n, w, 2)$ it is sufficient to construct a DNA code that fits the parameters. Every time a code is constructed. we know $B_r(n, w, 2)$ is at least as large as the size of the code. In this section six algorithms are explained that each construct a DNA code that satisfies the constraints and has minimum distance 2. Each algorithm starts by generating the set $\mathcal{B}_r(n, w)$, which already satisfies the GC-weight and runlength constraints, and deletes words in order to obtain minimum distance 2. The order in which the words appear in this set is important, here the set is generated in lexicographical order [2].

The first algorithm is from Limbachiya et al. [6] and the second and third algorithm are from Van Leeuwen [2]. Here the same algorithms are adjusted to account for an arbitrary maximum runlength r. Algorithms 4,5 and 6 were conceived in this research to improve the lower bound obtained in the first three algorithms. Every algorithm takes as input a length n, a GC-weight w, a maximum runlength r and a minimum distance d and results in a code that satisfies the parameters. The algorithms were implemented in Python code, which can be found in Appendices A.2 and A.3.

3.1 Algorithms 1,2 and 3

Algorithm 1:

1. The set $\mathcal{B}_r(n, w)$ is generated lexicographically which contains all the words that have length n and satisfy the GC-weight w and maximum runlength r.
2. The algorithm checks for every word in this set how many other words are within $d-1$ distance of this word and places this information in a dictionary with the words as keys and the number of words within the $d-1$ radius as values.
3. It iterates through the dictionary with each iteration removing a key with the maximum value. Every time a key is removed the values are adjusted since this word should no longer be counted.
4. After the iterations there are no words within $d-1$ distance of each other so the words in the dictionary have minimum distance d. The number of words in the dictionary can be counted to obtain a lower bound for $\mathcal{B}_r(n, w, d)$.

Algorithm 2 which is a variation of the first algorithm:

1. A dictionary is created in the same way as Algorithm 1 and an empty list is created.
2. Instead of removing keys with the maximum value from the dictionary, the iteration adds the key with the lowest value to the empty list.
3. This key is removed from the dictionary along with all the words that have distance up to $d-1$ to this key.

4. The values of all the keys that have distance up to $d-1$ of the removed words are reduced.

5. The iteration repeats until the dictionary is empty. The words in the list have minimum distance d resulting in a lower bound for $\mathcal{B}_r(n, w, d)$.

Algorithm 3 which is simpler than the previous two algorithms as it does not create a dictionary:

1. A list with all the words from $\mathcal{B}_r(n, w)$ is generated lexicographically, and an empty list is created.

2. In the first iteration the first word is moved to the empty list.

3. With every iteration the algorithm checks if the next word has at least a distance of d with every word in the other list. If it does, the word is moved to this list. If it does not the word is deleted.

4. The result is a list with words that satisfies minimum distance d so the size of this list is a lower bound for $\mathcal{B}_r(n, w, d)$.

3.2 Algorithms 4,5 and 6

Van Leeuwen [2] proves that the first three algorithms generate codes of maximum possible size for $\mathcal{B}_1(n, \lfloor \frac{n}{2} \rfloor, 2)$. This means the lower bound produced by those algorithms was of maximum size. The next three algorithms were conceived based on this result in an attempt to improve on the lower bounds obtained for $\mathcal{B}_r(n, w, 2)$ in the first three algorithms.

Algorithm 4:

1. This algorithm starts by generating a code with maximum runlength $r = 1$ and minimum distance $d = 2$ using Algorithm 1. We know this code is of maximum size for $w = \lfloor \frac{n}{2} \rfloor$. The list of words in $\mathcal{B}_r(n, w)$ is also generated.

2. With every iteration the algorithm checks if a word from $\mathcal{B}_r(n, w)$ has a distance of at least 2 with every word in the code. If it does the word is added to the code. If not the word is deleted from the list.

3. The resulting code has words from $\mathcal{B}_r(n, w)$ and has minimum distance 2.

Algorithm 5:

1. Step 1 from Algorithm 4.

2. Generate the set of all words in $\mathcal{B}_r(n, w)$ without the words that satisfy $r = 1$, $\mathcal{B}_r(n, w) \setminus \mathcal{B}_1(n, w)$. Note that no words from this set are in the generated code since only words from $\mathcal{B}_1(n, w)$ can be in the code.

3. The algorithm proceeds to add the words from $\mathcal{B}_r(n, w) \setminus \mathcal{B}_1(n, w)$ to the code, and then applies Algorithm 1 on the code again to obtain minimum distance 2.

4. The resulting code has words from $\mathcal{B}_r(n, w)$ and has minimum distance 2.

Algorithm 6 follows the same principle as Algorithm 5 but uses Algorithm 3 to generate the code and obtain minimum distance 2 at the end instead of Algorithm 1.

3.3 Evaluation of the Algorithms

The results from every algorithm for $r = 2$ and $r = 3$ are given in Tables 2 and 3 respectively for $2 \leq n \leq 9$ and GC-weight $\lfloor \frac{n}{2} \rfloor$. For the values of n where different lower bounds are obtained the largest lower bound is underlined. In Appendix B.1 tables can be found for other values of w.

For the case $r = 1$, Algorithms 1,2 and 3 all obtained the maximum possible value for $B_1(n, \lfloor \frac{n}{2} \rfloor, 2)$ [2]. This maximum value is given in Table 2.

From Tables 2 and 3 we see that different algorithms obtain the largest lower bound for $r \geq 2$. For $n = 7$ and $r = 2$, algorithms 1 and 2 obtain larger lower bounds than Algorithm 3, while for $n = 8$ and $r = 2$ Algorithm 2 obtains a larger lower bound than algorithms 1 and 3. For $n = 8$ and $r = 3$, Algorithm 3 produces the largest lower bound.

We can see that Algorithms 4,5 and 6 did not improve on the lower bound for $B_r(n, w, 2)$ compared to Algorithms 1,2 and 3.

Table 2: Lower bounds for $B_2(n, w, 2)$ from each algorithm with $w = \lfloor \frac{n}{2} \rfloor$.

	$r = 1$	$r = 2$					
n		Alg 1	Alg 2	Alg 3	Alg 4	Alg 5	Alg 6
2	4	4	4	4	4	4	4
3	8	12	12	12	12	12	12
4	32	48	48	48	48	48	48
5	68	<u>148</u>	<u>148</u>	<u>148</u>	125	120	138
6	216	<u>580</u>	<u>580</u>	<u>580</u>	472	504	544
7	528	<u>1892</u>	<u>1892</u>	1864	1419	1672	1694
8	1704	7344	<u>7352</u>	7244	5414	6512	6736
9	4336	24632	<u>24696</u>	23996	17057	22624	22464

Table 3: Lower bounds for $B_3(n, w, 2)$ from each algorithm with $w = \lfloor \frac{n}{2} \rfloor$.

	$r = 3$					
n	Alg 1	Alg 2	Alg 3	Alg 4	Alg 5	Alg 6
2	4	4	4	4	4	4
3	12	12	12	12	12	12
4	48	48	48	48	48	48
5	160	160	160	131	138	154
6	640	640	640	496	564	628
7	2208	2208	2208	1618	2036	2100
8	8800	8800	8808	6182	8056	8512
9	31092	31204	31088	21230	29376	22464

4 Parity Symbol Construction

One of the most common examples of error-detecting codes with minimum distance $d = 2$ in the binary case involves the use of a parity bit also called a check bit [9]. Starting with binary words with length $n - 1$ a bit can be added to each word such that the total sum of the bits is even. The resulting code with length n will have the property that the minimum distance is 2 since changing any bit would result in a word with an odd weight indicating the word is not in the code. A similar construction can be built for DNA codes that satisfy the constraints.

4.1 Applying the constraints

It seems that one of the problems with applying this idea to DNA codes is that there always exists two symbols that when added result in a word with even weight: $\{0, 2\}$ or $\{1, 3\}$. However choosing different symbols affects the GC-weight of the word so the symbol can be specifically chosen to either increase the GC-weight, or keep it constant. Depending on which is chosen, the initial set of words with length $n - 1$ should have GC-weight w or $w - 1$. When constructing a subset of $\mathcal{B}_r(n, w)$ both cases need to be considered. This is illustrated in the following example.

Example 7. We construct a subset of $\mathcal{B}_2(3, 1)$ with minimum distance $d = 2$ using this method. The runlength constraint is redundant in this case since no word can have 3 of the same symbol, due to the GC-weight.

We start with two sets of words with length $n - 1 = 2$. The difference between the two sets is the GC-weight, one has GC-weight $w - 1$ and the other has GC-weight w. Those two sets are:

$$\mathcal{B}_2(2, 0) = \{00, 01, 10, 11\} \quad \text{and} \quad \mathcal{B}_2(2, 1) = \{02, 03, 12, 13, 20, 21, 30, 31\}.$$

We add a symbol at the end of each word to make the total sum of the symbols even. In the first set only a 2 or 3 can be added to obtain $w = 1$. In the second set only a 0 or 1 can be added. This results in the following subset of $\mathcal{B}_2(3, 1)$:

$$\mathcal{C} = \{002, 013, 103, 112\} \cup \{020, 031, 121, 130, 200, 211, 301, 310\}.$$

Notice that changing a 1 into a 3 in any word would keep the total sum even but change the GC-weight of the word so it is not in $\mathcal{B}_2(3, 1)$. In the same way a 0 cannot be interchanged with a 2. Additionally every word in $\mathcal{C}$ differs in at least two positions compared to any other word from $\mathcal{C}$, which means $d(\mathcal{C}) = 2$. Thus a subset of $\mathcal{B}_2(3, 1)$ has been constructed with minimum distance 2.

Combining the two sets we constructed a DNA code with size 12 that satisfies the given constraints and indicating

$$B_2(3, 1, 2) \geq B_2(2, 0) + B_2(2, 1) = 4 + 8 = 12.$$

The argument above can be used in the same way to construct any DNA code with the property $r \geq \max\{w, n - w\}$ as adding a symbol cannot increase the maximum

runlength in a word. However, when this is not the case there is a set of words that would violate the runlength constraint.

We have shown that the GC-weight constraint does not cause any issues with the construction of DNA codes using the method in Example 7. Taking the runlength constraint into account is more difficult. It is possible that the added symbol is equal to the last r symbols of a word which would lead to $r+1$ repeated symbols. This word cannot be in the code as it violates the runlength constraint.

Example 8. Consider the set $\mathcal{B}_2(5,2)$. To construct a code with words from this set using the method in Example 7, we start with all words from $\mathcal{B}_2(4,2)$ and add either a 0 or a 1 to make the total sum of the symbols even. The words 2200 and 3300 are in this set and would have a 0 added to it which violates the runlength constraint. Similarly the words 2311 and 3211 cannot have a 1 added to it. By inspection there are no other words that would violate the runlength constraint in this way so there are 4 words out of the $B_2(4,2) = 96$ that should not be counted.

Next we look at all the words from $\mathcal{B}_2(4,1)$ where we add either a 2 or a 3. Since $w = 1$, adding either would never violate the runlength constraint. So there are $B_2(4,1) = 56$ words that satisfy the conditions.

Combining the 2 results gives a DNA code that satisfies $n = 5$, $r = 2$, $w = 2$ and $d = 2$ showing

$$B_2(5,2,2) \geq B_2(4,2) + B_2(4,1) - 4 = 96 + 56 - 4 = 148.$$

Both examples ended up with lower bounds for $B_r(n,w,2)$ that are equal to the maximum values obtained using the algorithms in Chapter 3. The method used in examples 7 and 8 is generalised in the next section.

4.2 The Parity symbol

Definition 14. For given values of n, r and w, the **parity symbol** x_p of a word $\mathbf{x}$ is defined as

$$x_p = \begin{cases} 0 & \text{if } \mathbf{x} \in \mathcal{B}_r(n,w) \wedge wt(\mathbf{x}) \text{ is even,} \\ 1 & \text{if } \mathbf{x} \in \mathcal{B}_r(n,w) \wedge wt(\mathbf{x}) \text{ is odd,} \\ 2 & \text{if } \mathbf{x} \in \mathcal{B}_r(n,w-1) \wedge wt(\mathbf{x}) \text{ is even,} \\ 3 & \text{if } \mathbf{x} \in \mathcal{B}_r(n,w-1) \wedge wt(\mathbf{x}) \text{ is odd.} \end{cases}$$

Definition 14 indicates which symbol is added at the end of any word that results in a word with length $n+1$, even weight, GC-weight w and most importantly has minimum distance 2 with any other word that is generated using the parity symbol.

Theorem 2. *Let n, w, and r be integers satisfying $0 \leq w \leq n$ and $r \geq 1$. Let $C_1 = \mathcal{B}_r(n, w)$ and $C_2 = \mathcal{B}_r(n, w-1)$. Let C be the DNA code generated by C_1 and C_2 by adding the parity symbol to every word in the two codes:*

$$C = \{\mathbf{x}x_p : \mathbf{x} \in C_1 \cup C_2\}. \tag{4}$$

Then C has words with length $n+1$, GC-weight w and C has minimum distance 2.

Proof. The words in C are words with length n with one symbol added. So the words clearly have length $n+1$

If $\mathbf{x} \in \mathcal{B}_r(n, w)$ then $x_p \in \{0, 1\}$ and so $\mathbf{x}x_p$ has GC-weight w. If on the other hand $\mathbf{x} \in \mathcal{B}_r(n, w-1)$ then $x_p \in \{2, 3\}$ so $\mathbf{x}x_p$ also has GC-weight w. Thus C only has words with GC-weight w.

To prove the minimum distance, let $\mathbf{x}$, $\mathbf{y} \in C_1 \cup C_2$ with $\mathbf{x} \neq \mathbf{y}$ and x_p and y_p their parity symbols. There are two cases to consider, $x_p = y_p$ and $x_p \neq y_p$.

If $x_p = y_p$, then $wt(\mathbf{x}) = wt(\mathbf{y})$ and either the GC-weight of both words is w or it is $w-1$. Suppose $d(\mathbf{x},\mathbf{y}) = 1$ and let i denote the position in which $\mathbf{x}$ and $\mathbf{y}$ differ. Since the GC-weights are equal, there are four possibilities for $(x_i, y_i) : (0, 1)$, $(1, 0)$, $(2, 3)$ and $(3, 2)$. The rest of the symbols of $\mathbf{x}$ and $\mathbf{y}$ are equal, so this indicates one word has even weight and one has odd weight which is a contradiction.

If $x_p \neq y_p$, then $\mathbf{x}x_p$ differs in at least 2 positions compared to $\mathbf{y}y_p$ since $\mathbf{x} \neq \mathbf{y}$. Together with the case $x_p = y_p$ this indicates the words in C have minimum distance ≥ 2.

To obtain minimum distance 2 we show two words exist that have distance 2. Let $\mathbf{x} \in C_1$ and $\mathbf{y} \in C_2$ with $x_i = y_i$ for $1 \leq i \leq n-1$, $x_n = 0$ and $y_n = 2$. Then $x_p \neq y_p$ so $d(\mathbf{x}x_p, \mathbf{y}y_p) = 2$. □

Theorem 2 shows the construction of DNA codes using the parity symbol always results in the code having minimum distance 2.

We can now formally define the set of words that violate the runlength constraint like the words in Example 8.

Definition 15. We define $\mathcal{R}_r(n, w)$ as the set of all words from $\mathcal{B}_r(n, w)$ and $\mathcal{B}_r(n, w-1)$ with the property that the last r symbols are equal to the parity symbol:

$$\mathcal{R}_r(n, w) = \{\mathbf{x} \in \mathcal{B}_r(n, w) \cup \mathcal{B}_r(n, w-1) : x_{n-r+1} = \cdots = x_n = x_p\}.$$

Its cardinality is denoted by $|\mathcal{R}_r(n, w)| = R_r(n, w)$.

The set $\mathcal{R}_r(n, w)$ can be viewed as the set of all words with the given parameters that would violate the runlength constraint if a parity symbol is added. If we omit those words from the construction, adding the parity symbol will not violate the

runlength constraint and every word in $\mathcal{B}_r(n, w)$ and $\mathcal{B}_r(n, w-1)$ corresponds to a unique word with length $n+1$ that differs in at least two positions from every other word. This results in the following lower bound for $B_r(n, w, 2)$:

$$B_r(n, w, 2) \geq B_r(n-1, w) + B_r(n-1, w-1) - R_r(n-1, w). \tag{5}$$

4.3 A formula for $\mathcal{R}_r(n, w)$

In this section we focus on the number of words in $\mathcal{R}_r(n, w)$. We start by defining several required sets with different properties which are then used in proving a recursive formula for $R_r(n, w)$. The proof of this formula follows a similar approach to the proof of Theorem 1 from [3].

We start by seperating the words in $\mathcal{R}_r(n, w)$. There are four disjoint sets of words that would violate the runlength constraint when a parity symbol is added:

$$\begin{aligned}
\mathcal{R}_0 &= \{\mathbf{x} \in \mathcal{B}_r(n, w) : wt(\mathbf{x}) \text{ is even} \wedge\ x_{n-r+1} = \cdots = x_n = 0\},\\
\mathcal{R}_1 &= \{\mathbf{x} \in \mathcal{B}_r(n, w) : wt(\mathbf{x}) \text{ is odd} \wedge\ x_{n-r+1} = \cdots = x_n = 1\},\\
\mathcal{R}_2 &= \{\mathbf{x} \in \mathcal{B}_r(n, w-1) : wt(\mathbf{x}) \text{ is even} \wedge\ x_{n-r+1} = \cdots = x_n = 2\},\\
\mathcal{R}_3 &= \{\mathbf{x} \in \mathcal{B}_r(n, w-1) : wt(\mathbf{x}) \text{ is odd} \wedge\ x_{n-r+1} = \cdots = x_n = 3\}.
\end{aligned} \tag{6}$$

Resulting in

$$\mathcal{R}_r(n, w) = \mathcal{R}_0 \cup \mathcal{R}_1 \cup \mathcal{R}_2 \cup \mathcal{R}_3, \tag{7}$$

and

$$R_r(n, w) = |R_0| + |R_1| + |R_2| + |R_3|.$$

Example 9. Following Example 8, the words in $\mathcal{R}_2(4, 2)$ can be written using $\mathcal{R}_0, \mathcal{R}_1, \mathcal{R}_2, \mathcal{R}_3$:

$$\begin{aligned}
\mathcal{R}_0 &= \{2200, 3300\},\\
\mathcal{R}_1 &= \{2311, 3211\},\\
\mathcal{R}_2 &= \emptyset,\\
\mathcal{R}_3 &= \emptyset.
\end{aligned}$$

This results in $\mathcal{R}_2(4, 2) = \{2200, 3300, 2311, 3211\}$ and $R_2(4, 2) = 4$.

To obtain a formula for the general value of $R_r(n, w)$ the number of words in $\mathcal{R}_i$ with $i \in \{0, 1, 2, 3\}$ can be counted. This can be done by looking more closely at the properties of $\mathcal{R}_i$. With the last r symbols of each word being equal, we can consider the symbols before the last r symbols to be from the set $\mathcal{B}_r(n-r, w)$ for $\mathcal{R}_0$ and $\mathcal{R}_1$ and from the set $\mathcal{B}_r(n-r, w-r-1)$ for $\mathcal{R}_2$ and $\mathcal{R}_3$. However, the words cannot end with the symbol in question and also have even/odd weight requirements.

Before we state and prove the recursive formula for $R_r(n, w)$ a number of sets are defined which apply these requirements:

Definition 16. $\mathcal{N}_r^{E_0}(n,w)$ is defined as the subset of all words in $\mathcal{B}_r(n,w)$ with even weight that do not end in a 0, and $\mathcal{N}_r^{E_1}(n,w)$ the subset of even words that do not end in a 1. $\mathcal{N}_r^{O_0}(n,w)$ and $\mathcal{N}_r^{O_1}(n,w)$ are the respective subsets with odd weight instead of even weight.

$$\begin{aligned}
\mathcal{N}_r^{E_0}(n,w) &= \{\mathbf{x} \in \mathcal{B}_r(n,w) : wt(\mathbf{x}) \text{ is even } \wedge x_n \neq 0\},\\
\mathcal{N}_r^{E_1}(n,w) &= \{\mathbf{x} \in \mathcal{B}_r(n,w) : wt(\mathbf{x}) \text{ is even } \wedge x_n \neq 1\},\\
\mathcal{N}_r^{O_0}(n,w) &= \{\mathbf{x} \in \mathcal{B}_r(n,w) : wt(\mathbf{x}) \text{ is odd } \wedge x_n \neq 0\},\\
\mathcal{N}_r^{O_1}(n,w) &= \{\mathbf{x} \in \mathcal{B}_r(n,w) : wt(\mathbf{x}) \text{ is odd } \wedge x_n \neq 1\}.
\end{aligned}$$

Their cardinalities are denoted by $N_r^{E_0}(n,w)$, $N_r^{E_1}(n,w)$, $N_r^{O_0}(n,w)$, and $N_r^{O_1}(n,w)$ respectively.

First Remark: The words with even weight and words with odd weight not ending with a 0 together form the set $\mathcal{N}_r(n,w)$ with the formula for $N_r(n,w)$ shown in Theorem 1. The number of words not ending with a 1 also equals $N_r(n,w)$. In equation:

$$N_r^{E_0}(n,w) + N_r^{O_0}(n,w) = N_r(n,w) = N_r^{E_1}(n,w) + N_r^{O_1}(n,w). \tag{8}$$

Second Remark: It may seem that $N_r^{E_0}(n,w)$ and $N_r^{O_1}(n,w)$ are equal in size for some parameters. The following is a counter-example:

Example 10. With the binary set $\mathcal{B}_2(4,0)$ ($w = 0$ therefore no 2's and 3's) the following sets are as defined in Definition 16.

$$\begin{aligned}
\mathcal{N}_2^{E_0}(4,0) &= \{(0,0,1,1),(0,1,0,1),(1,0,0,1)\},\\
\mathcal{N}_2^{E_1}(4,0) &= \{(0,1,1,0),(1,0,1,0),(1,1,0,0)\},\\
\mathcal{N}_2^{O_0}(4,0) &= \{(1,0,1,1),(1,1,0,1)\},\\
\mathcal{N}_2^{O_1}(4,0) &= \{(0,0,1,0),(0,1,0,0)\}.
\end{aligned}$$

This shows $N_2^{E_0}(4,0) \neq N_2^{O_1}(4,0)$.

For the cases $x_n \neq 2,3$ symmetrical formulas can be obtained. Consider the set $\mathcal{N}_r^{E_0}(n,w)$. Now replace every 0 by a 2, 1 by a 3 and vice-versa. Then the set is obtained containing all words with even weight not ending in a 2 that satisfy the runlength constraint r and have GC-weight $n-w$. The number of words in this set is the same as the number of words in the set $\mathcal{N}_r^{E_0}(n,n-w)$.

$$|\{\mathbf{x} \in \mathcal{B}_r(n,w) : \text{ wt}(\mathbf{x}) \text{ is even } x_n \neq i\}| = \begin{cases} N_r^{E_0}(n,w) & \text{if } i = 0,\\ N_r^{E_1}(n,w) & \text{if } i = 1,\\ N_r^{E_0}(n,n-w) & \text{if } i = 2,\\ N_r^{E_1}(n,n-w) & \text{if } i = 3. \end{cases} \tag{9}$$

$$|\{\mathbf{x} \in \mathcal{B}_r(n,w) : \text{ wt}(\mathbf{x}) \text{ is odd } x_n \neq i\}| = \begin{cases} N_r^{O_0}(n,w) & \text{if } i = 0, \\ N_r^{O_1}(n,w) & \text{if } i = 1, \\ N_r^{O_0}(n,n-w) & \text{if } i = 2, \\ N_r^{O_1}(n,n-w) & \text{if } i = 3. \end{cases} \tag{10}$$

Example 11. For the case $n = 4$, $r = 2$, and $w = 2$ which are the parameters used in Example 9, the last 2 symbols of every word in $\mathcal{R}_0$ and $\mathcal{R}_1$ are equal. By removing the last 2 symbols we obtain unique words with length $n - 2 = 2$ and GC-weight $w = 2$. For R_0 the resulting word has even weight and does not end in a 0. For R_1 the resulting word has odd weight and does not end with a 1. This gives:

$$\text{for } \mathcal{R}_0 : \mathcal{N}_r^{E_0}(n-r,w) = \mathcal{N}_2^{E_0}(2,2) = \{22, 33\},$$
$$\text{for } \mathcal{R}_1 : \mathcal{N}_r^{E_0}(n-r,w) = \mathcal{N}_2^{O_1}(2,2) = \{23, 32\}.$$

The sets $\mathcal{R}_2$ and $\mathcal{R}_3$ in this case are the empty set. Suppose there were words in these sets. Then for every word the last two symbols would be equal and the GC-weight would be $w - 1 = 1$. Removing the last two symbols would result in unique words with length $n - 2 = 2$ and GC-weight $w - 1 - r = -1$. Since a word can not have a GC-weight of -1 we expect the result to be 0 in this case. For $\mathcal{R}_2$ the resulting word has even weight and does not end in a 2. For $\mathcal{R}_3$ the resulting word has odd weight and does not end in a 3. Applying the formulas in (9) and (10) the number of words with these properties is:

$$\text{for } \mathcal{R}_2 : N_r^{E_0}((n-r),(n-r)-(w-1-r)) = N_2^{E_0}(2,3) = |\emptyset| = 0,$$
$$\text{for } \mathcal{R}_3 : N_r^{O_1}((n-r),(n-r)-(w-1-r)) = N_2^{E_1}(2,3) = |\emptyset| = 0.$$

The number of words in each of the sets in Definition 16 can be obtained using the following recursion formula. It follows a similar idea to the recursion formula for $N_r(n,w)$ obtained in [3].

Theorem 3. *For $0 \leq w \leq n$ and $r \geq 1$, it holds that*

$$N_r^{E_0}(n,w) = N_r^{E_1}(n,w) = N_r^{O_0}(n,w) = N_r^{O_1}(n,w) = \\ = 2^{n-2}\binom{n-1}{w} + 2^{n-1}\binom{n-1}{w-1}, \tag{11}$$

if $1 \leq n \leq r$,

$$\begin{aligned} N_r^{E_0}(n,w) = &\sum_{j=1}^{\min\{r,w\}} N_r^{E_0}(n-j,n-w) + \sum_{j=1,odd}^{\min\{r,n-w\}} N_r^{O_1}(n-j,w) \\ &+ \sum_{j=2,even}^{\min\{r,n-w\}} N_r^{E_1}(n-j,w) + \sum_{j=1,odd}^{\min\{r,w\}} N_r^{O_1}(n-j,n-w) \\ &+ \sum_{j=2,even}^{\min\{r,w\}} N_r^{E_1}(n-j,n-w), \end{aligned} \tag{12}$$

$$N_r^{O_0}(n,w) = \sum_{j=1}^{\min\{r,w\}} N_r^{O_0}(n-j,n-w) + \sum_{j=1,odd}^{\min\{r,n-w\}} N_r^{E_1}(n-j,w) + \sum_{j=2,even}^{\min\{r,n-w\}} N_r^{O_1}(n-j,w) + \sum_{j=1,odd}^{\min\{r,w\}} N_r^{E_1}(n-j,n-w) + \sum_{j=2,even}^{\min\{r,w\}} N_r^{O_1}(n-j,n-w), \tag{13}$$

$$N_r^{E_1}(n,w) = \sum_{j=1}^{\min\{r,n-w\}} N_r^{E_0}(n-j,w) + \sum_{j=1}^{\min\{r,w\}} N_r^{E_0}(n-j,n-w) + \sum_{j=1,odd}^{\min\{r,w\}} N_r^{O_1}(n-j,n-w) + \sum_{j=2,even}^{\min\{r,w\}} N_r^{E_1}(n-j,n-w), \tag{14}$$

$$N_r^{O_1}(n,w) = \sum_{j=1}^{\min\{r,n-w\}} N_r^{O_0}(n-j,w) + \sum_{j=1}^{\min\{r,w\}} N_r^{O_0}(n-j,n-w) + \sum_{j=1,odd}^{\min\{r,w\}} N_r^{E_1}(n-j,n-w) + \sum_{j=2,even}^{\min\{r,w\}} N_r^{O_1}(n-j,n-w), \tag{15}$$

if $n > r$, *and*

$$R_r(n,w) = N_r^{E_0}(n-r,w)+N_r^{O_1}(n-r,w)+N_r^{E_0}(n-r,n-w+1)+N_r^{O_1}(n-r,n-w+1) \tag{16}$$

if r *is even,*

$$R_r(n,w) = N_r^{E_0}(n-r,w)+N_r^{E_1}(n-r,w)+N_r^{E_0}(n-r,n-w+1)+N_r^{E_1}(n-r,n-w+1) \tag{17}$$

if r *is odd.*

Proof. In the case $n \leq r$ any word from the set $\mathcal{B}_r(n,w) = \mathcal{B}(n,w)$ satisfies the runlength constraint. To find $N_r^{E_0}(n,w)$ we can simply count the number of words in $\mathcal{B}(n,w)$ that have even weight and end in $i \in \{1,2,3\}$.

If $i = 1$, the last symbol can be removed to form a word from $\mathcal{B}_r(n-1,w)$ with odd weight. In every such word there are w symbols from $\{2,3\}$ and $n-1-w$ symbols from $\{0,1\}$. In every position, if one symbol results in a word with even weight, the other symbol results in a word with odd weight. This means for every word with even weight there is a word with odd weight. Thus there are $2^{n-1}\binom{n-1}{w}/2 = 2^{n-2}\binom{n-1}{w}$ such words with even weight (and of odd weight) that end in a 1.

If $i = 2$, the same argument applies but instead of a word from $\mathcal{B}_r(n-1,w)$ with odd weight, it is a word from $\mathcal{B}_r(n-1,w-1)$ with even weight. This results in $2^{n-1}\binom{n-1}{w-1}/2 = 2^{n-2}\binom{n-1}{w-1}$ such words. If $i = 3$, the word is from $\mathcal{B}_r(n-1,w-1)$ with odd weight also resulting in $2^{n-1}\binom{n-1}{w-1}/2 = 2^{n-2}\binom{n-1}{w-1}$ words.

This results in $2^{n-2}\binom{n-1}{w}+2^{n-1}\binom{n-1}{w-1}$ words for $N_r^{E_0}(n,w)$. A symmetrical argument exists for $N_r^{E_1}(n,w)$, $N_r^{O_0}(n,w)$ and $N_r^{O_1}(n,w)$. Together this proves equation (11)

If $n > r$, then any word in $\mathcal{N}_r^{E_0}(n,w)$ can be decomposed into a word from $\mathcal{B}_r(n-j,v)$ not ending with an $i \in \{1,2,3\}$ followed by j equal symbols i.

If $i = 1$, then $v = w$ and $j \in \{1,2,..,\min\{r,n-w\}\}$, with $j \leq r$ due to the runlength constraint and $j \leq n-w$ as that is the maximum number of ones in the word. If j is odd then the word from $\mathcal{B}_r(n-j,v)$ has odd weight and does not end in a 1. There are $N_r^{O_1}(n-j,w)$ such words for each odd j, summing over the odd values of j gives a total of $\sum_{j=1,odd}^{\min\{r,n-w\}} N_r^{O_1}(n-j,w)$. If j is even then the word from $\mathcal{B}_r(n-j,v)$ has even weight and does not end in a 1. Summing over the even values of j there are $\sum_{j=2,even}^{\min\{r,n-w\}} N_r^{E_1}(n-j,w)$ such words.

If $i = 2$, then $v = w-j$ and $j \in \{1,2,..,\min\{r,w\}\}$ and the word from $\mathcal{B}_r(n-j,v)$ has even weight and does not end in a 2. By applying formula (9) there are $N_r^{E_0}(n-j,n-w)$ such words for each j, summing over j gives in total $\sum_{j=1}^{\min\{r,w\}} N_r^{E_0}(n-j,n-w)$ words.

If $i = 3$, then $v = w-j$ and $j \in \{1,2,..,\min\{r,w\}\}$. If j is odd then the word from $\mathcal{B}_r(n-j,v)$ has odd weight and does not end in a 3. If j is even it has even weight and does not end in a 3. Again applying (9), for the odd values of j the number of words is $N_r^{O_1}(n-j,n-w)$ and for even values of j the number of words is $N_r^{E_1}(n-j,n-w)$

Summing over all the possible values of j this proves equation (12).

The proof for equation (14) follows the same idea. This proof is shown in Appendix B.3.

To prove equation (13) we can use $N_r^{O_0}(n,w) = N_r(n,w) - N_r^{E_0}(n,w)$ and the formula in Theorem 1.

$$N_r^{O_0}(n,w) = \sum_{j=1}^{\min\{r,n-w\}} N_r(n-j,n-w) + 2 \sum_{j=1}^{\min\{r,w\}} N_r(n-j,n-w) - N_r^{E_0}(n,w) \quad (18)$$

$$\begin{aligned}
&= \sum_{j=1,odd}^{\min\{r,n-w\}} (N_r(n-j,w) - N_r^{O_1}(n-j,w)) \\
&+ \sum_{j=2,even}^{\min\{r,n-w\}} (N_r(n-j,w) - N_r^{E_1}(n-j,w)) \\
&+ \sum_{j=1}^{\min\{r,w\}} (N_r(n-j,n-w) - N_r^{E_0}(n-j,n-w)) \qquad (19) \\
&+ \sum_{j=1,odd}^{\min\{r,w\}} (N_r(n-j,n-w) - N_r^{O_1}(n-j,n-w)) \\
&+ \sum_{j=2,even}^{\min\{r,w\}} (N_r(n-j,n-w) - N_r^{E_1}(n-j,n-w))
\end{aligned}$$

Using formula (8) again we obtain equation (13). The proof for equation (15) is similar and is shown in Appendix B.3

Finally we prove (16) and (17). The number of words in the set $\mathcal{R}_r(n,w)$ is the sum of the number of words in the sets R_0,R_1,R_2 and R_3. We start with counting the number of words in R_0.

From (6) every word in R_0 can be reduced to a word in $\mathcal{B}_r(n-r,w)$ with even weight not ending in a 0 by removing the last r symbols. This means the number of words in R_0 is $N_r^{E_0}(n-r,w)$. For R_1 removing the last r symbols results in a word from $\mathcal{B}_r(n-r,w)$ not ending with a 1. If r is even the resulting word must have odd weight of which there are $N_r^{O_1}(n-r,w)$ words. If r is odd the resulting word has even weight resulting in $N_r^{E_1}(n-r,w)$.

For R_2 every word can be reduced to a word in $\mathcal{B}_r(n-r,w-1-r)$ with even weight not ending in a 2. Applying (9) there are $N_r^{E_0}(n-r,n-w+1)$ such words. Similarly for R_3 there are $N_r^{O_1}(n-r,n-w+1)$ words if r is even, and $N_r^{E_1}(n-r,n-w+1)$ words if r is odd. □

4.4 Evaluation of the Parity Symbol Construction

Using Theorem 3 to count the number of words in $\mathcal{R}_r(n,w)$ the results of values on the right side of (5) are given in Tables 4 and 5 for the values $r=1$ and $r=2$ respectively. The resulting lower bound is also shown. The Python code can be found in Appendix A.6.

Table 4: Values of $B_1(n-1,w), B_1(n-1,w-1)$, $R_1(n-1,w)$ and the resulting lower bound for different values of n, $w = \lfloor \frac{n}{2} \rfloor$.

n	$B_1(n-1,w)$	$B_1(n-1,w-1)$	$R_1(n-1,w)$	Lower bound for $B_1(n,w,2)$
3	8	2	2	8
4	16	16	0	32
5	56	24	16	64
6	128	128	44	212
7	424	232	136	520
8	1040	1040	376	1704
9	3352	2104	1168	4288
10	8576	8576	3548	13604

Table 5: Values of $B_2(n-1,w), B_2(n-1,w-1)$, $R_2(n-1,w)$ and the resulting lower bound for different values of n, $w = \lfloor \frac{n}{2} \rfloor$.

n	$B_2(n-1,w)$	$B_2(n-1,w-1)$	$R_2(n-1,w)$	Lower bound for $B_2(n,w,2)$
3	8	4	0	12
4	24	24	0	48
5	96	56	4	148
6	296	296	12	580
7	1160	792	60	1892
8	3784	3784	216	7352
9	14696	10896	896	24696
10	49392	49392	3360	95424

The tables for $r = 3$ and $r = 4$ are given in Appendix B.2.

Remarkably the lower bounds calculated are always equal to the largest lower bound obtained from the algorithms in Chapter 3 for $r \geq 2$ even in cases where only one of the algorithms produced such a lower bound. For $r = 1$ the lower bound is smaller than the maximum size possible for some values of n, an explanation for this is provided the next section.

4.5 The case r=1

Comparing the results in Table 4 to the lower bounds for $B_1(n,w,2)$ in Table 2, shows that using the parity symbol method does not always produce the largest possible lower bound code when $r = 1$. For all $5 \leq n \leq 10$ except $n = 8$, the lower bound is too small. This suggests there are words that could be added to the set while maintaining $d = 2$. In this section we show that for $r = 1$ there are words with odd weight with a specific property that can be added in this way. We also show that this property does not occur for larger values of r.

When using the parity symbol method, $R_1(n,w)$ is the set of words with no repeated symbols and the property that with a parity symbol added the runlength constraint

is violated. However, it is possible that after adding the parity symbol, every symbol in the word is next to its opposite symbol (see Definition 9). If this is the case the parity symbol can be replaced by a different symbol while maintaining minimum distance 2 and satisfying the constraints. This is explained in three examples that also explain why for $n = 8$ the parity symbol construction does produce the largest possible DNA code.

Definition 17. Let $\mathbf{x}$ be a word and x_p its parity symbol for given parameters n, r and w as in Definition 14. Then the odd version of the word $\mathbf{x}x_p$ is $\mathbf{x}y_p$ with y_p being the opposite symbol of x_p:

$$y_p = \begin{cases} 0 & \text{if } x_p = 1, \\ 1 & \text{if } x_p = 0, \\ 2 & \text{if } x_p = 3, \\ 3 & \text{if } x_p = 2. \end{cases}$$

Example 12. In this example the case $n = 5, w = 2, r = 1$ and $d = 2$ is examined. Let $\mathcal{B}_1^P(5, 2, 2)$ be the DNA code generated using the parity symbol construction, and $\mathcal{R}_1(4, 2)$ the subset of words from $\mathcal{B}_1(4, 2)$ and $\mathcal{B}_1(4, 1)$ that would violate the runlength constraint if the parity symbol is added. From Table 4, $|\mathcal{B}_1^P(5, 2, 2)| = 64$ and $|\mathcal{R}_1(4, 2)| = 16$. The number of words in the code with maximum size is $B_1(5, 2, 2) = 68$, so it appears as if 4 words are missing.

Note that all the words in $\mathcal{B}_1^P(5, 2, 2)$ have even weight. Analysing every word in $\mathcal{R}_1(4, 2)$ with its parity symbol added, we observe that all have even weight and two equal symbols at the end. For example, $(2, 0, 2, 0, 0)$. The odd version, $(2, 0, 2, 0, 1)$, can not be added to $\mathcal{B}_1^P(5, 2, 2)$ without affecting the minimum distance because $(3, 0, 2, 0, 1)$ is already in the code. This can be observed by looking at the initial word $(2, 0, 2, 0, 0)$. The odd version, $(2, 0, 2, 0, 1)$ certainly has a distance of 1 to at least one word in $\mathcal{B}_1^P(5, 2, 2)$, since if the first 2 is flipped to its opposite symbol 3 the runlength and GC-weight constraints are satisfied, and the word has even weight.

Some words do not have this property, e.g. $(1, 0, 1, 2, 2)$. This word is not in $\mathcal{B}_1^P(5, 2, 2)$ since it violates the runlength constraint, so without its last symbol this word is in $\mathcal{R}_1(4, 2)$. In this case the odd version, $(1, 0, 1, 2, 3)$ does have at least distance 2 to every word in $\mathcal{B}_1^P(5, 2, 2)$. This occurs because no symbol in $(1, 0, 1, 2, 3)$ can be swapped to its opposite symbol without violating the runlength constraint. But swapping any symbol to a different symbol results in the GC-weight changing and so the word is not in $\mathcal{B}_1^P(5, 2, 2)$. Thus $(1, 0, 1, 2, 3)$ differs in at least two positions to every word in $\mathcal{B}_1^P(5, 2, 2)$ and can be added without changing the minimum distance.

There are three more words in $\mathcal{R}_1(4, 2)$ with this property: $(2, 3, 1, 0), (3, 2, 1, 0)$ and $(1, 0, 1, 3)$.

Adding the odd versions of these words to $\mathcal{B}_1^P(5, 2, 2)$ does not change the minimum distance. This DNA code has $64 + 4 = 68$ words of which 64 words have even weight and 4 words have odd weight and this is the maximum size possible.

Example 13. Now we look at the case $n = 7, w = 3, r = 1$ and $d = 2$. The set of words generated using the parity symbol construction has size 520 while $B_1(7, 3, 2) = 528$. Call the set of words generated using the parity symbol $\mathcal{B}_1^P(7, 3, 2)$. In $\mathcal{R}_1(6, 3)$ there are eight words where every symbol is next to its opposite symbol after the parity symbol is added:

$$(2,3,2,0,1,0,0),\ (2,3,2,1,0,1,1),\ (0,1,0,1,2,3,3\},\ (1,0,1,0,2,3,3),$$
$$(0,1,2,3,2,0,0),\ (0,1,2,3,2,1,1),\ (1,0,2,3,2,0,0\},\ (1,0,2,3,2,1,1).$$

With the same argument as in Example 12, the odd versions can be added to $\mathcal{B}_1^P(7, 3, 2)$ without affecting the minimum distance.

$$(2,3,2,0,1,0,1),\ (2,3,2,1,0,1,0),\ (0,1,0,1,2,3,2),\ (1,0,1,0,2,3,2),$$
$$(0,1,2,3,2,0,1),\ (0,1,2,3,2,1,0),\ (1,0,2,3,2,0,1),\ (1,0,2,3,2,1,0).$$

This means the DNA code including these words and the words from $\mathcal{B}_1^P(7, 3, 2)$ has size $520 + 8 = 528$ which is the maximum size possible.

We give another example to show what happens in the case $n = 8$. Note that the set generated using the parity symbol construction is already of maximum size 1704.

Example 14. The set $\mathcal{R}_1(7, 4)$ contains 376 words and with the parity symbol added the words have even weight, GC-weight 4 and end with two equal symbols. We are looking for a word where every symbol is next its opposite symbol and the word without the parity symbol is in $\mathcal{R}_1(7, 4)$. There are four symbols from $\{0, 1\}$, and four symbols from $\{2, 3\}$. For every word the symbols must be split in four pairs of symbols which are each others opposite symbol. If this is not the case then at least one symbol does not have a neighbouring opposite symbol.

If the four symbols from both sets are adjacent the word is any of the following:

$$(2,3,2,3,1,0,1,0),(2,3,2,3,0,1,0,1),(3,2,3,2,1,0,1,0),(3,2,3,2,0,1,0,1),$$
$$(1,0,1,0,2,3,2,3),(0,1,0,1,2,3,2,3),(1,0,1,0,3,2,3,2),(0,1,0,1,3,2,3,2).$$

Except all these words are already in the DNA code. This means without the last symbol the words are not in $\mathcal{R}_1(7, 4)$. If the pairs are split differently the same conclusion is reached because the first seven symbols can not be in $\mathcal{R}_1(7, 4)$.

The examples above explain why the sets generated using the parity symbol construction are smaller than the maximum size possible for $r = 1$. This construction only considers words with even weight, while the words with odd weight can be added while maintaining minimum distance 2. Moreover, Example 14 explains why it is of maximum size for $n = 8$.

Now consider $r = 2$, then two repeated symbols are permitted. If a word is in $\mathcal{R}_2(n - 1, w)$ then the word with its parity symbol ends with three equal symbols. Following Examples 12 and 13, if the odd version has a distance of at least two to every word in the generated code then it can be added to the code. However in this

case the second to last symbol can be changed to its opposite without violating the runlength constraint, unlike when $r = 1$.

Example 15. Consider the set $\mathcal{B}_2(5, 2)$ from Example 8 and the DNA code generated from this set using the parity symbol construction. The words in $\mathcal{R}_2(4, 2)$ with the parity symbol are

$$\{(2, 2, 0, 0, 0), (3, 3, 0, 0, 0), (2, 3, 1, 1, 1), (3, 2, 1, 1, 1)\}.$$

The odd versions are

$$\{(2, 2, 0, 0, 1), (3, 3, 0, 0, 1), (2, 3, 1, 1, 0), (3, 2, 1, 1, 0)\}.$$

These words can not be added to the DNA code while maintaining minimum distance 2 since changing the fourth symbol to its opposite produces a word already in the code.

$$\{(2, 2, 0, 1, 1), (3, 3, 0, 1, 1), (2, 3, 1, 0, 0), (3, 2, 1, 0, 0)\}$$

The odd versions will always have a corresponding word that only differs in the second to last position already in the DNA code.

For $r = 1$ we are able to find words with odd weight that when added to the DNA code produce codes of maximum size for some values of n and w. We assume this is also the case for other values of n and w but this is not proven in this thesis.

5 Upper and Lower bounds for $B_r(n, w, 2)$

In this Chapter an additional lower bound is proven and the upper and lower bounds obtained for $B_r(n, w, 2)$ so far are summarised. After comparing the bounds a conjecture is put forward which proposes a formula for $B_r(n, w, 2)$, however for $w = 0$ and $w = n$, a counter example has been found.

5.1 Even/odd weight lower bound

A lower bound for the size of $\mathcal{B}_r(n, w, 2)$ is obtained by considering the subset of $\mathcal{B}_r(n, w)$ containing only words with even weight and the subset containing only words with odd weight.

Theorem 4. *For $0 \leq w \leq n$ and $r \geq 1$, it holds that*

$$B_r(n, w, 2) \geq \frac{B_r(n, w)}{2}. \tag{20}$$

Proof. The subset of words in $\mathcal{B}_r(n, w)$ with even weight has minimum distance 2 since changing any one symbol either makes the word have odd weight or changes the GC-weight. The same argument applies for the subset of words in $\mathcal{B}_r(n, w)$ with odd weight. Since the two subsets are disjoint and together form the whole set $\mathcal{B}_r(n, w)$, at least one of the two subsets must have size $\frac{B_r(n,w)}{2}$ or larger resulting in a subset with minimum distance 2 with that size. □

The lower bound in Theorem 4 may seem trivial but it is important when comparing it to the lower bounds from the parity symbol construction in Chapter 4. The DNA code produced by using the parity symbol construction contains only words with even weight from $\mathcal{B}_r(n, w)$. This means if the number of words with even weight in $\mathcal{B}_r(n, w)$ is the same as the number of words with odd weight, the parity symbol construction will not produce a larger lower bound compared to Theorem 4.

Example 16. Consider the set $\mathcal{B}_3(4, 0)$. This is the set $\mathcal{B}(4, 0)$ without the words with 4 repeated symbols:

$$\mathcal{B}_3(4, 0) = \mathcal{B}(4, 0) \setminus \{(0, 0, 0, 0), (1, 1, 1, 1)\}.$$

$\mathcal{B}(4, 0)$ contains 8 words with even weight and 8 words with odd weight. This means $\mathcal{B}_3(4, 0)$ contains 6 words with even weight and 8 words with odd weight. So the DNA code with minimum distance 2 produced by the parity symbol construction for $n = 4$, $w = 0$ and $r = 3$ will contain at most 6 words, while Theorem 4 states

$$B_3(4, 0, 2) \geq \frac{B_3(4, 0)}{2} = 7.$$

Therefore the parity symbol construction produces a smaller lower bound for $B_3(4, 0, 2)$ than the lower bound in Theorem 4.

5.2 Parity symbol construction bounds

In Chapter 4.2 the parity symbol construction provides insights into other bounds for $B_r(n, w, 2)$ by constructing a DNA code using words with even weight. The number of words in the constructed DNA code provides a lower bound:

$$B_r(n, w, 2) \geq B_r(n-1, w) + B_r(n-1, w-1) - R_r(n-1, w). \tag{21}$$

This lower bound suggested the possibility of $B_r(n-1, w) + B_r(n-1, w-1)$ being an upper bound for $B_r(n, w, 2)$. In the next theorem we prove this is the case.

Theorem 5. *For $1 \leq w \leq n$ and $r \geq 1$, it holds that*

$$B_r(n, w, 2) \leq B_r(n-1, w) + B_r(n-1, w-1). \tag{22}$$

Proof. Let $\mathcal{B}_r(n, w, 2)$ be a DNA code of maximum size. For every $\mathbf{x} \in \mathcal{B}_r(n, w, 2)$ the last symbol x_n is removed resulting in a unique word $\mathbf{y}$ with length $n-1$. The word is unique since the minimum distance is 2 so removing any one symbol from two different words in $\mathcal{B}_r(n, w, 2)$ can never result in the same word. If $x_n \in \{0, 1\}$ then $\mathbf{y} \in \mathcal{B}_r(n-1, w)$ and if $x_n \in \{2, 3\}$ then $\mathbf{y} \in \mathcal{B}_r(n-1, w-1)$. Thus the number of words in $\mathcal{B}_r(n, w, 2)$ is bounded above by the sum of the number of words in $\mathcal{B}_r(n-1, w-1)$ and $\mathcal{B}_r(n-1, w-1)$ which results in (22). □

Equations (21) and (22) provide a lower bound and an upper bound for $B_r(n, w, 2)$ with a range of $R_r(n-1, w)$.

5.3 Comparing the bounds

The lower and upper bounds explained in the previous two sections are shown in Tables 6 and 7 for $r = 2$ and $r = 3$. The largest lower bound obtained from the algorithms in Chapter 3 is also shown.

Table 6: Upper and lower bounds for $B_2(n, w, 2)$ with $w = \lfloor \frac{n}{2} \rfloor$

	Lower bounds			Upper bound
n	$\frac{B_2(n,w)}{2}$	Algorithms	Parity symbol bound	$B_2(n-1, w) + B_2(n-1, w-1)$
3	12	12	12	12
4	48	48	48	48
5	148	148	148	152
6	580	580	580	592
7	1892	1892	1892	1952
8	7348	7352	7352	7568
9	24696	24696	24696	25592
10	95424		95424	98784

Table 7: Upper and lower bounds for $B_3(n, w, 2)$ with $w = \lfloor \frac{n}{2} \rfloor$

	Lower bounds			Upper bound
n	$\frac{B_3(n,w)}{2}$	Algorithms	Parity symbol bound	$B_2(n-1,w) + B_2(n-1,w-1)$
3	12	12	12	12
4	48	48	48	48
5	160	160	160	160
6	640	640	640	640
7	2208	2208	2208	2216
8	8804	8808	8808	8832
9	31204	31204	31204	31368
10	124180		124180	124816

The parity symbol construction does not produce larger lower bounds than $B_r(n, \lfloor \frac{n}{2} \rfloor)/2$ other than for $n = 8$. It also produces the same lower bound as the lower bound from the algorithms for $r \geq 2$. Since the bounds from the parity symbol construction and Theorem 4 depend on the number of even words, it is likely the differences occur when there is an unequal distribution of even and odd words (see Example 16). The upper bound is relatively close to the lower bounds which indicates if a larger DNA code exists then it is not much larger than the DNA code produced by the parity symbol construction. In Example 16 the code with odd weight produces one such larger DNA code.

Since for $r \geq 2$ no larger DNA code has been found than the one constructed in the parity symbol construction it is possible the constructed code is of maximum size. If this is the case then a formula exists for $B_r(n, w, 2)$ if $r \geq 2$. For the binary case $w = 0$ (and symmetrically $w = n$), Example 16 states a counterexample to this but for other values of w no counterexamples have been found yet. Because of this counterexample we believe the following conjecture does not hold for other values of w and n as well.

Conjecture 1. *For $1 \leq w \leq n-1$ and $r \geq 2$, it holds that*

For $r \leq max\{w, n-w\}$,

$$B_r(n, w, 2) = B_r(n-1, w) + B_r(n-1, w-1) - R_r(n-1, w). \tag{23}$$

For $r > max\{w, n-w\}$,

$$B_r(n, w, 2) = B_r(n-1, w) + B_r(n-1, w-1). \tag{24}$$

Note that for $r > max\{w, n-w\}$ we know $R_r(n-1, w) = 0$ which means with the upper bound in Theorem 5 this part of the conjecture is true.

6 Conclusions and Future Research

In this chapter the conclusions of the research done in this thesis are discussed. In the last section some recommendations for future research are discussed.

6.1 Conclusions

In Chapter 2 the set $\mathcal{B}_r(n, w)$ is defined as the set of all DNA words with parameters n the length, w the GC-weight and r the maximum runlength of the words. A DNA code with minimum Hamming distance d that is of maximum size is $\mathcal{B}_r(n, w, d)$ with size $B_r(n, w, d)$. The aim of this thesis was to investigate $B_r(n, w, 2)$ for $r > 1$. The case for $B_1(n, w, 2)$ was settled in [2],[3].

Six algorithms were discussed in Chapter 3 that obtained lower bounds for $B_r(n, w, 2)$. The first three algorithms produced the same maximum value for $B_1(n, w, 2)$ in [2] but produced different lower bounds for the case $r \geq 2$ and $n \geq 7$. The other three algorithms were an attempt to improve the lower bound but did not do so. Since there is no clear maximum lower bound there is the possibility that some or none of the algorithms produced DNA codes of maximum size.

A common method to produce codes with minimum distance 2 is constructing a code containing words with even weight. In Chapter 4 this idea is applied to the DNA context to further investigate $B_r(n, w, 2)$. A small subset of words $\mathcal{R}_r(n, w)$ does not satisfy the runlength constraint after adding a symbol. By excluding this subset a lower bound is obtained for $B_r(n, w, 2)$:

$$B_r(n, w, 2) \geq B_r(n-1, w) + B_r(n-1, w-1) - R_r(n-1, w). \tag{25}$$

A formula for the number of words in $\mathcal{R}_r(n, w)$ is given in Section 4.3. The results from using this method are evaluated in Section 4.4. In Section 5.3 a conjecture is put forward that suggests inequality 25 is equality for $1 \leq w \leq n-1$.

In Chapter 5 we prove an upper bound for $B_r(n, w, 2)$ which together with inequality (25) indicates $B_r(n, w, 2)$ falls within a range of $R_r(n-1, w)$ values:

$$B_r(n, w, 2) \leq B_r(n-1, w) + B_r(n-1, w-1) \tag{26}$$

Since $R_r(n-1, w)$ is relatively small compared to $B_r(n-1, w)$ and $B_r(n-1, w-1)$, this gives an accurate impression of the size of the maximum sized DNA codes with minimum distance 2.

6.2 Future Research

This thesis focuses on maximum sized DNA codes with minimum distance $d = 2$. There are multiple ideas for further research.

The research in this thesis was limited to DNA codes that can detect single substitution errors. It is of interest to investigate DNA codes with more error-detecting and correcting capabilities. The parity symbol method most likely cannot be used in this case as it focuses on words with even weight. Finding a different construction that allows for arbitrary values of d and r will give more insights into maximum sized DNA codes.

The formula proposed for $B_r(n, w, 2)$ remains a conjecture. More research can be done to find a proof or a counter-example. If disproven then $B_r(n, w, 2)$ is within a range of size $R_r(n-1, w)$ values but a larger DNA code exists than the code constructed with the parity symbol construction. If it exists, it would be of great interest to find this maximum sized code to settle how much information can be stored in DNA codes with these parameters.

Further research into the algorithms is also advised. The algorithms used in this thesis do not produce larger DNA codes than the conjecture suggests is possible. Perhaps a different algorithm would result in larger DNA codes which would disprove the conjecture. In addition the algorithms used could be optimised for run time to allow computation of DNA codes for larger values of n.

Bibliography

[1] B. Cao, S. Zhao, X. Li, and B. Wang, "K-means multi-verse optimizer (KMVO) algorithm to construct DNA storage codes," *IEEE Access*, vol. 8, pp. 29547-29556, 2020.

[2] C.J.(Lot)van Leeuwen,"Constrained Codes for DNA-Based Storage Systems,"Bachelor Thesis, Delft University of Technology, May 2020.

[3] J.H. Weber, J.A.M. de Groot, and C.J. van Leeuwen,"On Single-Error-Detecting Codes for DNA-Based Data Storage," accepted for publication in *IEEE Commun. Lett.*, 2020

[4] 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.

[5] S. M. H. T. Yazdi, H. M. Kiah, E. Garcia-Ruiz, J. Ma, H. Zhao and O. Milenkovic, "DNA-Based Storage: Trends and Methods," in *IEEE Transactions on Molecular, Biological and Multi-Scale Communications*, vol. 1, no. 3, pp. 230-248, Sept. 2015

[6] D. Limbachiya, M. K. Gupta,and V. Aggarwal,"Family of constrained codes for archival DNA data storage," *IEEE Commun. Lett.*, vol. 22, no. 10, pp. 1972–1975, Oct. 2018.

[7] K. A. S. Immink and K. Cai, "Efficient Balanced and Maximum Homopolymer-Run Restricted Block Codes for DNA-based Storage," *IEEE Commun. Lett.*, vol. 23, no. 10, pp. 1676–1679, Oct. 2019.

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

[9] Ziemer, RodgerE.; Tranter, William H. *Principles of communication : systems, modulation, and noise* (Seventh ed.). Hoboken, New Jersey. ISBN 9781118078914. OCLC 856647730

A Python Code

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

def runlength(word,r):
    #Checks for r repeated digits in a word, returns True if the word
    #agrees with the runlength constraint
    for i in range(len(word)-r):
        t=0
        for j in range(0,r):
            if(word[i] == word[i+j+1]):
                t += 1
        if(t >= r):
            return False
    return True

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 gcweight(word):
    #Returns the GC-weighs of a word
    return word.count(2) + word.count(3)

def distance(word1,word2):
    #Returns the hamming-distance between two words of equal length
    if len(word1) != len(word2):
        return "The words are not of equal length"
    else:
        return len([i for i in range(len(word1)) if word1[i] !=
        word2[i]])

def neighbours(code,codeword,d):
    #Returns all words within d-1 distance of the word
    return [w for w in code if 0<distance(codeword, w)<d]

def minimum_distance(code):
    #Returns the minimum distance of a code
    d = 1000
    for codeword1 in code:
        for codeword2 in code:
            if codeword1 != codeword2:
```

```
                min_distance = distance(codeword1, codeword2)
                if d > min_distance:
                    d = min_distance
    return d

def maximum1(distances):
    #Returns the word with the maximum value of the distances in the
    #dictionary
    max1 = max(distances, key=distances.get)
    return max1

def minimum1(distances):
    #Returns the word with the minimum value of the distances in the
    #dictionary
    min1 = min(distances, key=distances.get)
    return min1
```

A.1 Generating $\mathcal{B}_r(n, w)$

```
def DNA(n,w,r):
    #This function returns the DNA code that satisfies the run-length
    #constraint r, GC-weight and is of length n.
    Code = list(itertools.product(range(4),repeat=n))
    DNAcode = [codeword for codeword in Code if runlength(codeword, r)
    return DNAcode
```

A.2 Algorithms 1,2 and 3

```
################### ALGORITHM 1 ####################

def alg1_step2_list(DNAcode,d):
    #A list/dictionary is made of all the words in DNAcode with each wo
    words_in_sphere = {codeword: neighbours(DNAcode,codeword,d) for cod
    return words_in_sphere

def alg1_step2_dist(words_in_sphere):
    #A dictionary as in the previous function but the neighbours are sh
    distances = {key:len(value) for key,value in words_in_sphere.items(
    return distances

def alg1_step3(distances, words_in_sphere):
    #This function defines the word with maximum distance, deletes it f
    #and reduces the value of each d-1 neighbour by one. It also remove
    maximum = maximum1(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]

    def alg1_step4(distances, words_in_sphere):
        #While the dictionary distances is not empty keep iterating alg1_st
        while max(distances.values())>0:
            alg1_step3(distances, words_in_sphere)
        return words_in_sphere

    def alg1(n,w,r,d):
        DNAcode = DNA(n,w,r)
        step2list = alg1_step2_list(DNAcode,d)
        step2dist = alg1_step2_dist(step2list)
        step3 = alg1_step4(step2dist,step2list)
        return step3

    ################## ALGORITHM 2 ###################

    def alg2_step3(distances,words_in_sphere,code):
        minimum = minimum1(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

    def alg2_step4(distances, words_in_sphere,code):
        while words_in_sphere:
            alg2_step3(distances,words_in_sphere,code)
        return code

    def alg2(n,w,r,d):
        DNAcode = DNA(n,w,r)
        step2list = alg1_step2_list(DNAcode,d)
        step2dist = alg1_step2_dist(step2list)
        DNAdistance = []
        step3 = alg2_step3(step2dist,step2list,DNAdistance)
        step4 = alg2_step4(step2dist,step2list,step3)
        return len(step4)
```

```
################## ALGORITHM 3 ##################

def alg3_step1(distancecode,DNAcode,d):
    word = DNAcode[0]
    if all(distance(word,codeword)>=d for codeword in distancecode):
        distancecode.append(word)
    DNAcode.remove(word)
    return distancecode, DNAcode

def alg3_step2(distancecode,DNAcode,d):
    while DNAcode:
        alg3_step1(distancecode,DNAcode,d)
    return distancecode

def alg3(n,w,r,d):
    DNAcode = DNA(n,w,r)
    DNAdistance = []
    step1a,step1b = alg3_step1(DNAdistance,DNAcode,d)
    step2 = alg3_step2(step1a,step1b,d)
    return step2
```

A.3 Algorithms 4,5 and 6

```
################## ALGORITHM 4 ##################

def alg4(n,w,r,d):
    DNAcode = alg3(n,w,1,d)
    DNAcode2 = DNA(n,w,r)
    for word in DNAcode2:
        if all(distance(word,codeword)>=d for codeword in DNAcode):
            DNAcode.append(word)
            DNAcode2.remove(word)
    return len(DNAcode), len(DNAcode2)

################## ALGORITHM 5 ##################

def alg5_step1_a(n,w,d):
    DNAcode = alg1(n,w,1,d)
    dictlist = []
    for key,value in DNAcode.items():
        dictlist.append(key)
    return dictlist

def alg5_step1(n,w,d):
    DNAcode = alg3(n,w,1,d)
    return DNAcode
```

```
def alg5_step2_list(DNAcode,n,w,r):
    x = DNA(n,w,r)
    x2 = []
    y = DNAcode
    for element in x:
        if element not in y:
            x2.append(element)
    for element in x2:
        y.append(element)
    return y

def alg5(n,w,r,d):
    step1 = alg5_step1(n,w,d)
    step2_code = alg5_step2_list(step1,n,w,r)
    step3_alg1_list = alg1_step2_list(step2_code,d)
    step4_alg1_dist = alg1_step2_dist(step3_alg1_list)
    step5 = alg1_step4(step4_alg1_dist,step3_alg1_list)
    return step5
```

################## ALGORITHM 6 ##################

```
def alg6_step1_a(n,w,d):
    DNAcode = alg1(n,w,1,d)
    dictlist = []
    for key,value in DNAcode.items():
        dictlist.append(key)
    return dictlist

def alg6_step1(n,w,d):
    DNAcode = alg3(n,w,1,d)
    return DNAcode

def alg6_step2_list(DNAcode,n,w,r):
    x = DNA(n,w,r)
    x2 = []
    y = DNAcode
    for element in x:
        if element not in y:
            x2.append(element)
    for element in x2:
        y.append(element)
    return y

def alg6(n,w,r,d):
    step1 = alg6_step1(n,w,d)
```

```
    step2_code = alg6_step2_list(step1,n,w,r)
    step3_alg1_list = alg1_step2_list(step2_code,d)
    step4_alg1_dist = alg1_step2_dist(step3_alg1_list)
    step5 = alg1_step4(step4_alg1_dist,step3_alg1_list)
    return step5
```

A.4 Recursive formula for $B_r(n, w)$

```
#The function below calculates the values for N_r(n,w) and B_r(n,w)

def Nrnw(r,n,w):
    if(n==0 and w==0):
        return 1
    elif n <= r:
        x = (2**(n-1))*sp.special.binom(n-1,w) + (2**(n))*sp.special.bi
        return x
    else:
        part1 = 0
        part2 = 0
        for i in range(1, min(r,n-w)+1):
            part1 = part1 + Nrnw(r, n-i, w)
        for i in range(1, min(r,w)+1):
            part2 = part2 + 2*Nrnw(r, n-i, n-w)
        return part1+part2

def Brnw(r,n,w):
    x=0
    for i in range(0, min(r,n-w)+1):
        x += Nrnw(r,n-i,w)
    return x
```

A.5 Recursive formulas

```
def NrE0(n,w,r):
    if(n==1 and w==0):
        return 0
    elif n <= r:
        x = (2**(n-2))*sp.special.binom(n-1,w) + (2**(n-1))*sp.special.
        print(x)
        print("E0")
        return x
    else:
        part1 = 0
        part2 = 0
        part3 = 0
        part4 = 0
```

```
        part5 = 0
        for i in range(1,min(r,w)+1):
            part1 = part1 + NrE0(n-i,n-w,r)
        for i in range(1,min(r,n-w)+1,2):
            part2 = part2 + NrO1(n-i,w,r)
        for i in range(2,min(r,n-w)+1,2):
            part3 = part3 + NrE1(n-i,w,r)
        for i in range(1,min(r,w)+1,2):
            part4 = part4 + NrO1(n-i,n-w,r)
        for i in range(2,min(r,w)+1,2):
            part5 = part5 + NrE1(n-i,n-w,r)
        return part1+part2+part3+part4+part5

def NrO0(n,w,r):
    if(n==1 and w==0):
        return 1
    elif n <= r:
        x = (2**(n-2))*sp.special.binom(n-1,w) + (2**(n-1))*sp.special.
        print(x)
        print("O0")
        return x
    else:
        part1 = 0
        part2 = 0
        part3 = 0
        part4 = 0
        part5 = 0
        for i in range(1,min(r,w)+1):
            part1 = part1 + NrO0(n-i,n-w,r)
        for i in range(1,min(r,n-w)+1,2):
            part2 = part2 + NrE1(n-i,w,r)
        for i in range(2,min(r,n-w)+1,2):
            part3 = part3 + NrO1(n-i,w,r)
        for i in range(1,min(r,w)+1,2):
            part4 = part4 + NrE1(n-i,n-w,r)
        for i in range(2,min(r,w)+1,2):
            part5 = part5 + NrO1(n-i,n-w,r)
        return part1+part2+part3+part4+part5

def NrE1(n,w,r):
    if(n==1 and w==0):
        return 1
    elif n <= r:
        x = (2**(n-2))*sp.special.binom(n-1,w) + (2**(n-1))*sp.special.
        print(x)
        print("E1")
```

```
        return x
    else:
        part1 = 0
        part2 = 0
        part3 = 0
        part4 = 0
        for i in range(1,min(r,n-w)+1):
            part1 = part1 + NrE0(n-i,w,r)
        for i in range(1,min(r,w)+1):
            part2 = part2 + NrE0(n-i,n-w,r)
        for i in range(1,min(r,w)+1,2):
            part3 = part3 + NrO1(n-i,n-w,r)
        for i in range(2,min(r,w)+1,2):
            part4 = part4 + NrE1(n-i,n-w,r)
        return part1+part2+part3+part4


def NrO1(n,w,r):
    if(n==1 and w==0):
        return 0
    elif n <= r:
        x = (2**(n-2))*sp.special.binom(n-1,w) + (2**(n-1))*sp.special.
        print(x)
        print("O1")
        return x
    else:
        part1 = 0
        part2 = 0
        part3 = 0
        part4 = 0
        for i in range(1,min(r,n-w)+1):
            part1 = part1 + NrO0(n-i,w,r)
        for i in range(1,min(r,w)+1):
            part2 = part2 + NrO0(n-i,n-w,r)
        for i in range(1,min(r,w)+1,2):
            part3 = part3 + NrE1(n-i,n-w,r)
        for i in range(2,min(r,w)+1,2):
            part4 = part4 + NrO1(n-i,n-w,r)
        return part1+part2+part3+part4

def Rrnw(n,w,r):
    if(r%2 ==0):
        x = NrE0(n-r,w,r)+NrO1(n-r,w,r)+NrE0(n-r,n-w+1,r)+NrO1(n-r,n-w+
        return x
    else:
        x = NrE0(n-r,w,r)+NrE1(n-r,w,r)+NrE0(n-r,n-w+1,r)+NrE1(n-r,n-w+
```

```
    return x
```

A.6 Algorithm to obtain the number of words in $R_r(n, w)$

```
#This function outputs the number of words that would
#satisfy the runlength constraint followed by the number of words
#that would not.

def paritybiteven(n,w,r):
    x = DNA(n-1,w,r)
    y = DNA(n-1,w-1,r)
    x1 = []
    y2 = []
    for i in x:
        if(i[n-1-r] == 0 and i[n-r] ==0 and ... and  i[n-2] == 0):
            t=0
            for j in range(0,n-1):
                t += i[j]
            if(t % 2)==0:
                x1.append(i)
        if(i[n-1-r] == 0 and i[n-r] ==0 and ... and  i[n-2] == 0):
            t=0
            for j in range(0,n-1):
                t += i[j]
            if(t % 2)==1:
                x1.append(i)
    for ele in x1:
        x.remove(ele)
    for i in y:
        if(i[n-1-r] == 0 and i[n-r] ==0 and ... and  i[n-2] == 0):
            t=0
            for j in range(0,n-1):
                t += i[j]
            if(t % 2)==0:
                y2.append(i)
        if(i[n-1-r] == 0 and i[n-r] ==0 and ... and  i[n-2] == 0):
            t=0
            for j in range(0,n-1):
                t += i[j]
            if(t % 2)==1:
                y2.append(i)
    for ele in y2:
        y.remove(ele)
    return len(x),len(y),len(x1),len(y2)
```

B Parity Symbol construction

B.1 Lower bounds for $B_r(n, w, d)$ for other values of w

Table 8: Lower bounds for $B_2(n, w, 2)$ from algorithms 1,2 and 3 for $w = \lfloor \frac{n}{2} \rfloor - 1$ and $w = \lceil \frac{n}{2} \rceil + 1$

	$w = \lfloor \frac{n}{2} \rfloor - 1$			$w = \lceil \frac{n}{2} \rceil + 1$		
n	Alg 1	Alg 2	Alg 3	Alg 1	Alg 2	Alg 3
2	2	2	2	2	2	2
3	3	3	3	3	3	3
4	28	28	48	28	28	28
5	56	60	56	56	60	56
6	396	396	384	396	396	384
7	948	960	900	948	960	900
8	5420	5448	5236	5420	5448	5236
9	14176	14360	13424	14176	14360	13424

Table 9: Lower bounds for $B_2(n, w, 2)$ from algorithms 1,2 and 3 for $w = \lfloor \frac{n}{2} \rfloor - 2$ and $w = \lceil \frac{n}{2} \rceil + 2$

	$w = \lfloor \frac{n}{2} \rfloor - 2$			$w = \lceil \frac{n}{2} \rceil + 2$		
n	Alg 1	Alg 2	Alg 3	Alg 1	Alg 2	Alg 3
4	6	6	3	6	6	3
5	7	8	6	7	8	6
6	118	120	108	118	120	108
7	222	232	208	222	232	208
8	2132	2180	1988	2132	2180	1988
9	4536	4716	4200	4536	4716	4200

B.2 Parity symbol table for $r = 3$ and $r = 4$

Table 10: Values of $B_3(n-1, w), B_3(n-1, w-1), R_3(n-1, w)$ and the resulting lower bound for different values of n, $w = \lfloor \frac{n}{2} \rfloor$.

n	$B_3(n-1, w)$	$B_3(n-1, w-1)$	$R_3(n-1, w)$	Lower bound for $B_3(n, w, 2)$
4	24	24	0	48
5	96	64	0	160
6	320	320	0	640
7	1280	936	8	2208
8	4416	4416	24	8808
9	17608	13872	164	31204
10	62408	62408	636	124180

Table 11: Values of $B_4(n-1,w), B_4(n-1,w-1)$, $R_4(n-1,w)$ and the resulting lower bound for different values of n, $w = \lfloor \frac{n}{2} \rfloor$.

n	$B_4(n-1,w)$	$B_4(n-1,w-1)$	$R_4(n-1,w)$	Lower bound for $B_4(n,w,2)$
4	24	24	0	48
5	96	64	0	160
6	320	320	0	640
7	1280	960	0	2240
8	4480	4480	0	8960
9	17920	14272	16	32176
10	64352	64352	60	128644

B.3 Recursive formula proof

Proof for $N_r^{E_1}(n,w)$:

Any word in $\mathcal{N}_r^{E_1}(n,w)$ can be decomposed into a word from $\mathcal{B}_r(n-j,v)$ not ending with an $i \in \{0,2,3\}$ followed by j equal symbols i.

If $i=0$, then $v=w$ and $j \in \{1,2,..,\min\{r,n-w\}\}$, with $j \leq r$ due to the run-length constraint and $j \leq n-w$ as that is the maximum number of ones in the word. The word is from $\mathcal{B}_r(n-j,w)$, has even weight and does not end in a 0. There are $N_r^{E_0}(n-j,w)$ such words for each j, summing over j gives a total of $\sum_{j=1}^{\min\{r,n-w\}} N_r^{E_1}(n-j,w)$ words.

If $i=2$, then $v=w-j$ and $j \in \{1,2,..,\min\{r,w\}\}$ and the word from $\mathcal{B}_r(n-j,v)$ has even weight and does not end in a 2. By applying formula (9) there are $N_r^{E_0}(n-j,n-w)$ such words for each j, summing over j gives in total $\sum_{j=1}^{\min\{r,w\}} N_r^{E_0}(n-j,n-w)$ words.

If $i=3$, then $v=w-j$ and $j \in \{1,2,..,\min\{r,w\}\}$. If j is odd then the word from $\mathcal{B}_r(n-j,v)$ has odd weight and does not end in a 3. If j is even it has even weight and does not end in a 3. Again applying (9), for the odd values of j the number of words is $N_r^{O_1}(n-j,n-w)$ and for even values of j the number of words is $N_r^{E_1}(n-j,n-w)$

Summing over all the possible values of j this proves equation (13).

Proof for $N_r^{O_1}(n,w)$:

To prove equation (15) we can use $N_r^{O_1}(n,w) = N_r(n,w) - N_r^{E_1}(n,w)$ and the formula in Theorem 1.

$$N_r^{O_1}(n,w) = \sum_{j=1}^{\min\{r,n-w\}} N_r(n-j,n-w) + 2 \sum_{j=1}^{\min\{r,w\}} N_r(n-j,n-w) - N_r^{E_1}(n,w) \quad (27)$$

$$
\begin{aligned}
&= \sum_{j=1}^{\min\{r,n-w\}} (N_r(n-j,w) - N_r^{E_0}(n-j,w)) \\
&+ \sum_{j=1}^{\min\{r,w\}} (N_r(n-j,n-w) - N_r^{E_0}(n-j,n-w)) \\
&+ \sum_{j=1,odd}^{\min\{r,w\}} (N_r(n-j,n-w) - N_r^{E_1}(n-j,n-w)) \\
&+ \sum_{j=2,even}^{\min\{r,w\}} (N_r(n-j,n-w) - N_r^{O_1}(n-j,n-w))
\end{aligned}
\tag{28}
$$

Using formula (8) again we obtain equation (13).