HANOI UNIVERSITY OF SCIENCE AND TECHNOLOGY

GRADUATION THESIS

Enumerative Coding Techniques for Constrained Systems

NGUYỄN VĂN LỰC
luc.nv176812@sis.hust.edu.vn

Major: Computer Science
Specialization: Global ICT

Supervisor:	Dr. Bùi Quốc Trung	_____
Co-supervisor:	Assoc. Prof. Đỗ Phan Thuận	Signature
Department:	Computer Science	
School:	School of Information and Communications Technology	

HANOI, 03/2023

ACKNOWLEDGMENT

First and foremost, I would like to sincerely devote this BSc thesis and my love to my parents and my brother. To all your cares, your supports, your encouragements, your loves, your forgiveness.

I am fortunate enough to have an opportunity to work under the supervision of Associate Professor Đỗ Phan Thuận for his invaluable and enthusiastic guidance. The sincere also go to the Combinatorics research team which joins work with Dr. Vũ Văn Khu, Dr. Trần Thị Thu Hương and other talented friends. I always feel very energetic while working in the team. I also want to express my sincere gratitude to Dr. Bùi Quốc Trung for his tremendous guidance and support. Even though the time is short, that is extremely important to me.

My life from my forth year until now would be tough without the support from my closed friends. I would like to express this uttermost gratitude to Đỗ Văn Khải. Even though we got closed quite lately after the exchange semester with memorable experiences, that does not slow down the relationship growing between us. The thank also goes to Đỗ Hoàng Thuấn for many times making me be more realistic.

Going along with that, I also want to give this acknowledgement to my great class, ICT 01 - K62. We have shared the wonderful 5 years. More than that, our whole future is coming.

Back to my first years, I would like to express my special appreciation to Hoàng Văn Sâm for your passionate help and guidance. Discussing with you always blew my mind.

My university experiences would be very limited without one of my best friends, Nguyễn Thị Thanh Hương. Enjoying "xiên bẩn" with you from HNUE campus to HUST campus is a super great idea I have ever thought. Hanging out with you, and Lê Thị Thanh Huyền, the whole night, makes me feel I am losing track of the time.

My thanks are also given to every friends sharing volunteering experiences with me, especially, to Volunteer Club and volunteering friends in my high school, Ngô Quyền - Ba Vì high school. Volunteering with all you are always fun, meaningful, jolly and blithe.

Time flies. Things come and go, memories stay. Thank you for all your appearances in my journey. And this is not a good bye. When one door closes another opens.

ABSTRACT

In the digital world, everything is digitalized into sequences of number, called digits. Such sequences can be used to represent information, share ideas, analyze to mine new knowledge and understanding, etc. With those key roles, information in general, or sequences of number in particular, has been studied for a long time to explore their properties[1]. However, to protect the information integrity from every source of modification (environmental impacts, device errors, human's misunderstanding, etc), the demands of Error Correction Code (ECC) emerge. Many error correction algorithms, such as Hamming coding, Low-Density Parity-Check Code, etc, have been well studied [2] [3]. One remarkable ECC that is recently studied is the (b, k) locally-constrained de Bruijn code [4]. This code has been proved to have many applications in correcting digital constrained sequences. To do so, an encoder is required to encode the sequences in some ways. One of the main concerns prior to study the encoder is to evaluate the code capacity and asymptotic rate. Their recent study only mentions about evaluating the asymptotic rate for the case of boundary $b = 3$. Realising the pattern-forbidden base of the code, this thesis will try to apply Goulden-Jackson inverse theorem to enumerate the code with larger boundary b. To complete the contribution, this thesis also try to build a simple encoder to demonstrate its operation.

Aside from that, thermal-aware sequences are also studied in this thesis. To model the temperature changing inside a system, (a, b)-Dyck word can be used. The family of m-Dyck paths have been studied in many researches such as [5] [6]. In deed, the number of m-Dyck paths is enumerated by the well-known Fuss-Catalan number: $a_{(m+1)n} = \frac{1}{mn+1}\binom{(m+1)n}{n}$. In case $m = 1$, we have the Catalan number: $a_{2n} = \frac{1}{n+1}\binom{2n}{n}$. Because of negative impacts and performance degrade, the peak temperature of devices should be bounded. With this motivation, this thesis considers the more restricted cases of the Dyck path, in which global height is bounded by a maximum height, say k, with many variants. The main contributions in this part are to compute the channel capacity with prescribed parameters and configuration.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1. INTRODUCTION

1.1 Problem Statement

High temperature on electronic devices not only causes performance degrade, system failure, but also burnout the circuits board, reduce the devices lifetime [7], [8]. Many different techniques have been proposed to face this thermal challenge [7], [9], [10]. To guarantee the performance of systems, peak temperature should be restricted in some constraints. This motivates us to study the codes which can represent the thermal constraints. Treating temperature changes in the systems as steps on a graph, we can track the temperature using a Dyck word in which digits represent either up-step (temperature increases) or down-step (temperature decreases). In order to constrain the peak temperature, we are studying the Dyck path in which total height is bounded.

On the other hand, in information theory and information communication channel, information is represented as digital data. The data transferred through the network is usually damaged by many kinds of environmental impacts. Besides, even though recent data storage technologies are modern, durable and stable under environments, data reading process might not work perfectly, either. These modifications or mistakes make the received data is not the same as the data sent from the source channel, and the received data might become useless. This imposes a challenge in data transmission and data storage to ensure the data integrity under multiple human and environmental impacts. To resolve this challenge, the solution should be able to prevent errors and correct errors code, in order to ensure the data integrity and consistency of any systems.

Can we do so?

From these two recent concerns, we are going to study the *Enumerative Coding Techniques for Constrained Systems*, which can disclose how good we can do to resolve these challenges.

The thesis closes with some open questions to further researches in the near future.

1.2 Background and Problems of Research

Many state-of-the-art solutions have been studied on the Dyck paths and their enumeration. Thermal-aware code problems have also been introduced many times recently [11]. However, there are very limited studies on the Dyck path in which total height is limited, which is very crucial in temperature controlling. By that

reason, this thesis is trying to contribute to this branch.

To combat the challenge in data storage , many state-of-the-art and mathematically proved solutions have been proposed. One considerable branch of these solutions is to use Error Correction Codes (ECC) which takes advantages of coding-decoding principles on digital codewords to protect digital information from errors. The *de Bruijn graphs* and its sequences have been founded to have many applications in constraining these codes and provide efficient methods to encode-decode codewords, which also help define many Error Correction Code systems with variety of configurations. Group of author [4] make use of Graph representations of *locally-constrained de Bruijn sequences* which enforce the constrained conditions on a pre-defined length of each subsequences, called *windows*. However, they are only able to build a graph representation of the code for a particular case of boundary $b = 3$, and apply Perron-Frobenius theory to evaluate the asymptotic rate of the code. Due to the complexity of graph representation, their method were unable to enumerate the number of codewords for a larger boundary of b.

1.3 Research Objectives and Conceptual Framework

One of the most significant parameters that must be known in order to build a coding-encoding systems is the code capacity, which implies the total number of satisfied codewords. In this theses, I focus on applying the decomposition techniques in various direction to find the cardinality of the code. Specifically, I propose different techniques to enumerate Dyck words in Dyck languages with different variants. In addition, I also work with de Bruijn sequences and constrained systems based on de Bruijn code, in order to build an encoder which can protect digital data from modifications. The fruit of this thesis is to prepare for a submission in an ISIT coding theory journal.

1.4 Contributions

This thesis contains three main contributions:

1. Enumeration techniques on Dyck words using decomposition method.
2. Enumeration technique on de Bruijn sequences using decomposition method.
3. De Bruijn code encoder construction.

1.5 Organization of Thesis

The remains of this thesis is organized as following:

In chapter 2, we acknowledge some recent state-of-the-art contributions, remarkable related works and introduce some important concepts, terminologies which are helpful and used in this thesis, including concepts of generating function, Dyck

language, Goulden-Jackson inversion theorem for cluster decompositions of sequences with distinguished subsequences.

Chapter 3 mentions some procedures that we study in the next part, including step-by-step direction to work on the next sections.

In chapter 4, we will study the generating function of Dyck paths with multiple variants, locally-constrained de Bruijn code enumeration. From this rate inference, we also apply the Finite States-splitting algorithm to construct an encoder which can be used to encode digital data in prescribed constraints.

In chapter 5, we evaluate the results of $(3, k)$ and $(4, k)$-locally-constrained de Bruijn code by visualizing the rate convergences and compare the capacity with the asymptotic rate calculated from the cardinality of the code given its generating function.

CHAPTER 2. LITERATURE REVIEW

2.1 Scope of Research

In information theory, enumerating code and evaluating the code capacity is one of the first step to analyze and investigate any coding system. Through its cardinality and its rate, the code can be justified to know its efficiency compared to other codes.

In formal language, a Dyck word is a balanced string of brackets. The set of Dyck words form a Dyck language. The number of Dyck words with exactly n pairs of such parentheses is the n-th Catalan number C_n. However, this only considered the case in which each up-step/down-step increase/decrease height by 1. In this thesis, I focus on a more general case, in which up-step/down-step increase/decrease height by 1/m and vice versa. The limited height of the Dyck path is also taken into consideration.

In the study on de Bruijn sequences and locally-constrained de Bruijn sequences, this thesis is focusing on enumerating $(4, k)$-locally-constrained de Bruijn sequences and constructing $(3, 2)$-locally-constrained de Bruijn encoder to complement to work of group authors [4]. We also prove that the methods work accurately compared to their theoretical results by plotting asymptotic rate convergence graphs for particular cases of (b, k) pairs.

In both problems, we are going to evaluate the generating function which represents the these cardinalities using decomposition method. Maclaurin expansion will then be used to extract the coefficients of the series to find the exact number of satisfied codewords (or number of Dyck words in a Dyck language, and number of sequences in locally-constrained de Bruijn code) of a given length.

2.2 Related Works

On the search of Dyck paths, Yukiko Fukukawa [6] counted the general Dyck path as a general lattice path from $(0, 0)$ to (m, n) under the diagonal line $y = \frac{n}{m}x$. Gabriella Baracchini [12] introduced two subsets of Dyck Paths, which are *Alpha Up-Down Walks* and *Beta Up-Down Walks*. Keiichi Shigechi [5] studied the combinational properties of Dyck path by decomposing it into some small Dyck paths. However, all these researches have not considered about the maximum height of the Dyck paths given the case that the Dyck path composes of *up* and *down* walks. This influences a natural demand in device's peak temperature control that directly emerges the foundation of thermal-aware sequences, in which Dyck

path is a remarkable candidate in this study. Therefore, in this research, we are interested in enumerating the more restricted Dyck paths of *up* and *down* steps in which height is bounded by a maximum value k.

On the research of de Bruijn sequences, many researches about its enumeration are also proposed [4], [13]. In general, the number of de Bruijn sequences of length n on the alphabet A of size k is [13]:

$$B(k,n) = \frac{(k!)^{k^{n-1}}}{k^n}$$

In most application, binary digits are used, meaning that $A = \{0,1\}, k = 2$ and $B(2,n) = \frac{(2!)^{2^{n-1}}}{2^n}$. These codes, however, are not suitable in encoding because the length of the code is limited by n. Since the length of data (represented by a binary sequence) can be very long (an image of size 1MB contains $2^{20} \times 8 = 8,388,608$ bits), an (n,b,k) locally-constrained de Bruijn sequence is proposed [4] as a more relaxed version of original (k,n) de Bruijn sequence. An (n,b,k)-locally-constrained de Bruijn sequence is a sequence of length n in which each sub-string of size k appears exactly once around its $b-1$ neighbors. The q-ary code on alphabet of size q is also considered in [4]. In this research, we are only considering the case of binary alphabet, $q = 2, A = \{0,1\}$.

2.3 Generating function

Generating function, in mathematics, is an useful way to encode a infinite-length series into a form of a function by treating them as the coefficients of a formal power series. Specifically, given a series a_n (the index starts from 0) in which n goes to infinity, then the ordinary generating function (OGF) of a_n is:

$$g(a_n, x) = \sum_{n=0}^{\infty} a_n x^n$$

In many situations, the properties of function are easily applied into the series in the very creative way.

For example, consider the series of Fibonacci number $F_n, n \geq 0$:

$$f_n = \begin{cases} 0 & \text{if} \quad n = 0 \\ 1 & \text{if} \quad n = 1 \\ f_{n-1} + f_{n-2} & \text{if} \quad n \geq 2 \end{cases}$$

Let $F(x) = \sum_{n=0}^{\infty} f_n x^n$ denote the generating function representing the Fibonacci

series. Then, for $n \geq 2$, we have

$$\begin{aligned}
f_n &= f_{n-1} + f_{n-2} \quad \forall n \geq 2 \\
\Leftrightarrow f_n x^n &= f_{n-1}x^n + f_{n-2}x^n \quad \forall n \geq 2 \\
\Leftrightarrow \sum_{n=2}^{\infty} f_n x^n &= \sum_{n=2}^{\infty} f_{n-1}x^n + \sum_{n=2}^{\infty} f_{n-2}x^n \quad \forall n \geq 2 \\
\Leftrightarrow F(x) - f_0 - xf_1 &= x(F(x) - f_0) + x^2 F(x) \\
\Leftrightarrow F(x) - x &= xF(x) + x^2 F(x) \\
\Leftrightarrow F(x) &= \frac{x}{1 - x - x^2}
\end{aligned}$$

Therefore, the generating function representing the Fibonacci series is $F(x) = \frac{1}{1 - x - x^2}$. Extracting this function exactly results the Fibonacci series.

2.4 Dyck path

Dyck path is a balanced string of brackets, for example of one kind of brackets ("[" and "]"), in which brackets ['s and]'s are concatenated so that counting from the left to right, the number of ['s is always more than or equal to the number of]'s. This order of ['s and]'s can, therefore, form a valid operation. Thus these strings are especially considered in context-free grammar. In general, Dyck language can be generated from a single non-terminal S by the production:

$$S \to \epsilon | \text{“[”} S \text{“]”} S$$

or an alternative production:

$$S \to (\text{“[”} S \text{“]”})^*$$

As Dyck words are balanced, number of “[” and “]” are equal, resulting that their lengths are always even. Let us consider the graph starting from $(0, 0)$ to $(2n, 0)$ by treating each “[” as up-step (increasing height by 1: $(x_{n+1}, y_{n+1}) = (x_n + 1, y_n + 1)$) and “]” as down-step (decreasing height by 1: $(x_{n+1}, y_{n+1}) = (x_n + 1, y_n - 1)$). It is clear that the graph always goes beyond the x-axis.

The definition of Dyck path [14] can be generalized in definition 1.

Definition 1: (a, b)-*Dyck path*

> An (a, b)-Dyck path is a lattice path from $(0, 0)$ to $(n, 0)$, consisting of up steps (go along vector $(1, a)$) and down steps (go along vector $(1, -b)$), and never go below $x-$axis

For simplicity, the notation m-Dyck path is also used to denote the case of $(1, m)$-

Dyck path.

An example of Dyck path of length $n = 10$ (in case $a = b = 1$ is shown in figure 2.1.

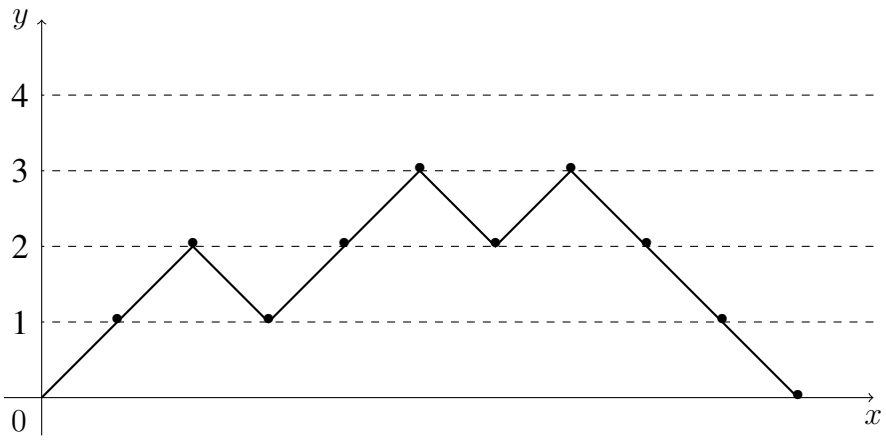


Figure 2.1: Example of a Dyck path of length 10

The n-th Catalan number C_n representing number of Dyck words with exactly n pairs of such parentheses ($a = b = 1$) is:

$$C_n = \frac{1}{n+1}\binom{2n}{n} = \frac{(2n)!}{(n+1)!n!} = \prod_{k=2}^{n}\frac{n+k}{k} = \binom{2n}{n} - \binom{2n}{n+1} \forall n \geq 0$$

2.5 Goulden-Jackson inversion theorem

In [15], I. P. Goulden and D. M. Jackson introduced an inversion theorem used to decompose sequences with distinguished subsequences. The method based on the fundamental definition of q-cluster on a *reduced set A* (i.e., a set in which no element appears as a sub-sequence of any other sequence) as below:

Definition 2: $q-cluster$

> A q-cluster($q \geq 1$) on the alphabet $\mathscr{N}$, and associated with the *reduced set* $\mathscr{A}$, is a triple $(\sigma_1\sigma_2...\sigma_r, A_{i_1}...A_{i_q}, (l_1, ..., l_q))$ such that $\sigma_1, \sigma_2, ..., \sigma_r \in \mathscr{N}^+$ and $(l_1, ..., l_q)$ satisfies with r_j is the length of A_{i_j}:
>
> 1. $\sigma_{l_j}\sigma_{l_j+1}...\sigma_{l_j+r_j-1} = A_{i_j}, j = 1, ...q$
> 2. $0 < l_{j+1} - l_j < r_j, j = 1, ...q-1$
> 3. $r = l_q + r_q - 1$ and $l_1 = 1$.

The third part of the triple, $(l_1, l_2, ..., l_q)$, simply denotes the starting point of each element in the set $\mathcal{A}$. In order for the elements to overlap, the second condition must hold. In the coming parts and examples, the last part is usually ignored in

representation for simplicity.

For example, given a word ABCDCBAD, if the reduced set $\mathscr{A} = \{ABC, BCDC, CBAD\}$, then $\{ABCDCBAD, \mathscr{A}\}$ is called a 3-cluster since two pairs of words in $\mathscr{A}$ overlap. On the other hand, if the reduced set $\mathscr{A} = \{ABC, BCDC, AD\}$, then $\{ABCDCBAD, \mathscr{A}\}$ is not a cluster because $BCDC$ and AD do not overlap in ABCDCBAD. $\{BABCDCBAD, \{ABC, BCDC, CBAD\}\}$ is not a cluster, either, because the first letter of $BABCDCBAD$, B, does not appear as the first letter of any words of the reduced set $\{ABC, BCDC, CBAD\}$.

Suppose that we need to find a generating function $f(t)$ representing the number of words of length n of alphabet $\mathcal{A}$ without any occurrence of words from a reduced set $\mathcal{B}$. It is also noticeable that the forbidden set $\mathcal{B}$ should be a reduced set. Otherwise, suppose that s_1, s_2 are two elements of $\mathcal{B}$ in which s_1 appears as a subsequence of s_2. This containment reduces the consideration of s_2 because if a word does not contain s_1, it will not contain s_2.

This is a good practice where the Goulden Jackson cluster generating function is powerfully and conveniently applied. Let $\mathcal{B}(w)$ denote the set of words in $\mathcal{B}$ which appears as a factor of w. As mentioned in [16] and explained in [17], we have the following result:

$$f(t) = \sum_{w \in \mathcal{A}^*} \sum_{S \subset B(w)} (-1)^{|S|} t^{|w|} \tag{2.1}$$

Where $\mathcal{A}^*$ is the set of all words created from the alphabet $\mathcal{A}$.

In the expression (2.1), the deep fact $0^0 = 1$ is exploited. If the word w contains k forbidden words from $\mathcal{B}$, then the sum that is counted is:

$$\texttt{count} = \sum_{i=0}^{k} (-1)^i \binom{k}{i} = (1 + (-1))^k = 0^k$$

If the word w does not contain any word from the set $\mathcal{B}$, then $k = 0$. This results $\texttt{count} = 0^0 = 1$. Hence, the word is counted once. On the other hand, if the word w does contain $k \geq 1$ words, the word is counted as $\texttt{count} = 0^k = 0$. Therefore, the word is not counted.

The fact above proves that the expression given in (2.1) correctly represents the generating function of words from alphabet $\mathcal{A}$ given forbidden words set $\mathcal{B}$. This suggests defining the weight of a word w given its sub-words set appearing in the forbidden words set $\mathcal{B}$, S, as following:

$$\text{weight}(w, S) = (-1)^{|S|} t^{|w|}. \tag{2.2}$$

Let $\mathcal{M}$ denote marked words that needs to be counted from the alphabet $\mathcal{A}$ with no occurrence of forbidden words from the set $\mathcal{B}$.

Using the concept of q-cluster defined in 2, by setting the reduced set $\mathscr{A}$ as the forbidden words set $\mathcal{B}$, we can see that a word in $\mathcal{M}$ can either be a empty word whose length is zero, start with a letter which is not a part of any cluster, or start with a letter that is a part of a cluster.

Let $\mathcal{C}$ denote the set of all clusters generated from the forbidden words set $\mathcal{B}$, we have the following decomposition:

$$\mathcal{M} = \{empty\_word\} \cup \mathcal{A}\mathcal{M} \cup \mathcal{C}\mathcal{M} \tag{2.3}$$

Taking weight of (2.3), with assumption that weight of the marked words set $\mathcal{M}$ is the generating function of words that needs to be counted, we have:

$$\text{weight}(\mathcal{M}) = 1 + |\mathcal{A}|t.\text{weight}(\mathcal{M}) + \text{weight}(\mathcal{C}).\text{weight}(\mathcal{M})$$

$$\Rightarrow f(t) = \text{weight}(\mathcal{M}) = \frac{1}{1 - |\mathcal{A}|t - \text{weight}(\mathcal{C})} \tag{2.4}$$

The problem now becomes to find weight($\mathcal{C}$) according to the first forbidden words of the cluster. Let $\mathcal{C}[v]$ denote the set of clusters starting with the forbidden word v. Then

$$\mathcal{C} = \bigcup_{v \in \mathcal{B}} \mathcal{C}[v]$$

which means

$$\text{weight}(\mathcal{C}) = \sum_{v \in \mathcal{B}} \text{weight}(\mathcal{C}[v]) \tag{2.5}$$

We now need to find weight($\mathcal{C}[v]$) for each v starting $\mathcal{C}$. By the definition of q-cluster, removing the starting forbidden word results either a empty word or a new cluster. In case of a new cluster, there can be more than one way that the new cluster can be formed.

For example, consider the 4-cluster $\{ABCDCBA, \{ABC, BCD, CDC, CBA\}\}$, removing the first forbidden word, ABC may result a 3-cluster $\{BCDCBA, \{BCD, CDC, CBA\}\}$ or a 2-cluster $\{CDCBA, \{CDC, CBA\}\}$. Therefore, all words in the

forbidden words set $\mathcal{B}$ should be considered while peeling off v.

By considering all possible overlaps of v with others forbidden words in $\mathcal{B}$, the decomposition of $\mathcal{C}[v]$ is obtained as described in [16]:

$$\mathcal{C}[v] = \{(v; \{v\})\} \cup \bigcup_{u \in \mathcal{B}} \bigcup_{r \in O(v,u)} (\mathcal{C}[u] \cdot (v \backslash r)) \tag{2.6}$$

where $O(v, u)$ denotes all possible overlaps of the end of v with the beginning of u, $(v \backslash r)$ is the result of peeling off r from the end of v, and $a \cdot b$ indicates the concatenation of b to the end of a.

The $\{(v; \{v\})\}$ set indicates the situation that the cluster contain only one forbidden word of the set $\mathcal{B}$. Therefore, $S = \{v\}$. Removing the word v returns an empty cluster.

Taking the weight of (2.6) gives:

$$\text{weight}(\mathcal{C}[v]) = \text{weight}((v; \{v\})) - \sum_{u \in \mathcal{B}} \left(\text{weight}(\mathcal{C}[u]) \cdot \sum_{r \in O(v,u)} \text{weight}(v \backslash r) \right) \tag{2.7}$$

Note that the sum over $u \in \mathcal{B}$ is negative (i.e. multiplied by -1) in order to compensate for having reduced the number of bad words in our cluster by one (because we are calculating weights of clusters containing one fewer bad word than the clusters in $\mathcal{C}[v]$).

We can easily calculate the weight of the first trivial set by the definition:

$$\text{weight}((v; \{v\})) = (-1)^{|\{v\}|} t^{|v|} = -t^{|v|}$$

Peeling off the r part for each $r \in O(v, u)$ results the beginning part, $v \backslash r$ which is no longer a cluster, becomes freely chosen from the alphabet $\mathcal{A}$. Therefore, we can calculate weight$(v \backslash r)$ as:

$$\text{weight}(v \backslash r) = t^{|v \backslash r|}$$

Hence, the equation (2.7) now looks simpler:

$$\text{weight}(\mathcal{C}[v]) = -t^{|v|} - \sum_{u \in \mathcal{B}} \left(\text{weight}(\mathcal{C}[u]) \cdot \sum_{r \in O(v,u)} t^{|v \backslash r|} \right) \tag{2.8}$$

The equation (2.8) is capable of manual calculation because u and v are used interchangeably. The interchangeability gives us a system of $|\mathcal{B}|$ linear equations of $|\mathcal{B}|$ variables where the solution of each variable can be determined resulting the desired representation of weight($\mathcal{C}$). Substitute weight($\mathcal{C}$) to (2.4), we obtain the generating function of the expected generating function of words from alphabet $\mathcal{A}$ given a forbidden words set $\mathcal{B}$.

2.6 de Bruijn sequences and locally-constrained de Bruijn sequences

2.6.1 Weak de Bruijn sequence and de Bruijn sequence

Formally, we have the definition of the $B(\sigma, k)$ de Bruijn sequence:

Definition 3: *de Bruijn sequences*

Let k be a positive integer and $\mathbb{F}_\sigma$ is an alphabet with σ symbols. A cyclic sequence $s = (s_1, s_2, ..., s_n)$, is called a *weak* $B(\sigma, k)$ de Bruijn sequence if each of its subsequences of length k appears exactly once. That is:

$$s[i, i+k-1] \neq s[j, j+k-1] \; \forall i \neq j, 1 \leq i, j \leq n-k+1$$

with an assumption that if $i + k - 1 > n$, we take $s_{i+k-1} = s_{i+k-1-n}$ as the sequence is cyclic.

There are σ^k distinct strings of length k that can be created from a size-σ alphabet. Therefore, there is a maximum of length $n = \sigma^k$ of the de Bruijn sequence order k formed from a size-σ alphabet. In this case, the sequence is called $B(\sigma, k)$*de Bruijn sequence of order* k.

The de Bruijn graph of $B(2, 3)$ is depicted in figure 2.2. Traversing every edges exactly once and returning to the same vertex yields a $B(2, 4)$-de Bruijn sequence. Traversing every vertices exactly once and returning to the same vertex yields a $B(2, 3)$-de Bruijn sequence.

For example, $s =$ "00010111" is a cyclic B(2,3) de Bruijn sequence since each triple of s are unique in s. As we can see, its length is 8, as the maximum of 2^3.

2.6.2 Locally-constrained de Bruijn sequence

As we have seen, the length of each de Bruijn sequence is limited by the length of each sub-string and the size of the alphabet. As a result, it is, in information theory, infeasible to guarantee the property of the de Bruijn sequence when the length of the transferred string is much longer and we only use a quite small value of k and σ. The property might only be committed on a short interval and should be periodic. This results a definition of *Locally-constrained de Bruijn sequence* as following:

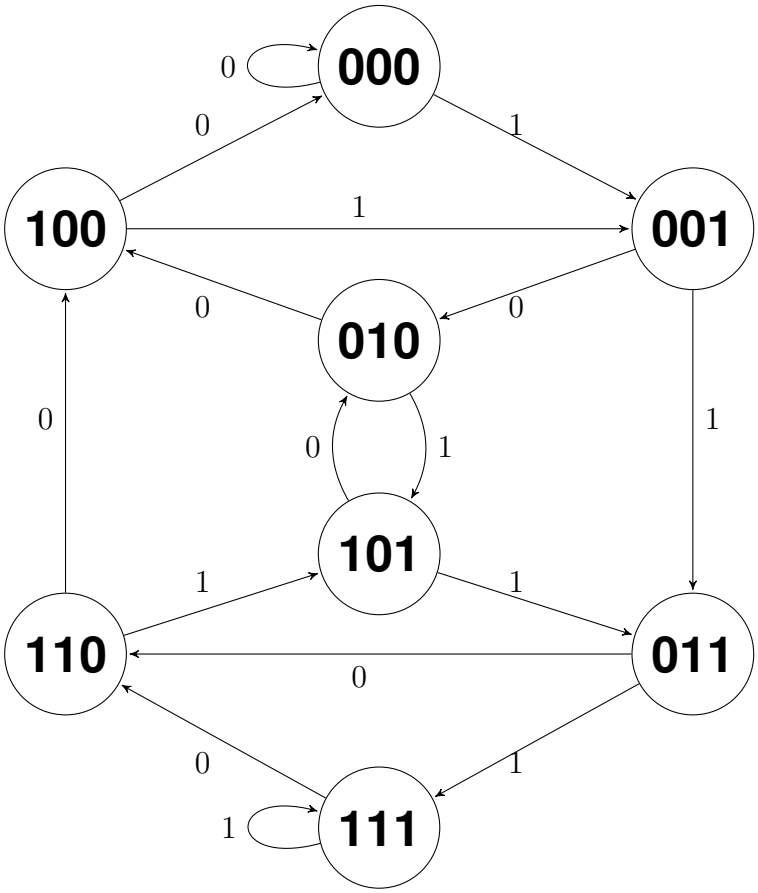


Figure 2.2: $B(2,3)$de Bruijn graph

Definition 4: *locally-constrained de Bruijn sequences*

Let b, k be two positive integers and $\mathbb{F}_\sigma$ is an alphabet with σ symbols. A cyclic sequence $s = (s_1, s_2, ..., s_n)$, is called a (b, k) locally-constrained de Bruijn sequence if each of its sub-string of length k appears exactly once around its $b-1$ neighbors. That is:

$$s[i, i+k-1] \neq s[j, j+k-1] \; \forall i \neq j, 1 \leq i, j \leq n-k+1, |i-j| \leq b-1$$

with an assumption that if $i+k-1 > n$, we take $s_{i+k-1} = s_{i+k-1-n}$ as the sequence is cyclic.

The definition can also be understood that in each sub-string s' of length $b+k-1$, there is no repeated k-tuple.

For example, $s =$ "00010111010" $\in \{0,1\}^{12}$ is a (3,3) locally-constrained de Bruijn sequence. Notice that the sub-string 101 and 010 is repeated but in they are 4 and 6 digits apart from each other.

It is noticeable that a (b, k) cyclic locally-constrained de Bruijn sequence of length n is a de Bruijn sequence without any constraints if and only if $b \leq \sigma^k$, which is also the maximum length of the sequence. If the sequence is acyclic, its maximum length can be $\sigma^k + k - 1$.

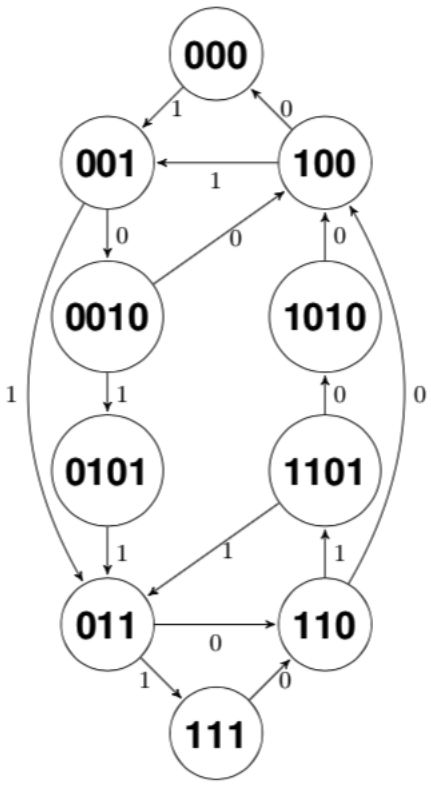


Figure 2.3: $(3, 3)$-locally-constrained De Bruijn code

We also call the set of distinct (b, k) locally-constrained de Bruijn sequences of length n is a (b, k) locally-constrained de Bruijn code, denoted by $\mathbb{C}^*_{DB}(n, b, k)$.

2.6.3 Construction

For the interesting property of de Bruijn sequence, one might ask about the easiest way to construct a de Bruijn sequence formally. As each sequence of size k is unique, we are focusing on building each these windows so that it different from all previously created one. Next, we need the last $k-1$ symbols of the precedent word to be the same as the first $k-1$ symbols in the successor word. To do this, we can multiple the precedent word with σ in order to shift left the whole word, and take to module on 2^k. This word is then simply added by one random digit so that it is unique over all of the previously created windows.

By the method above, we have the following official and simple algorithm to create a de Bruijn sequence for binary alphabet:

1. Start with the a binary sequence of size k, e.g. $a_0 = \{0, 0, ..., 0\}$.
2. For each $i \geq 1$, $a_i \equiv 2a_{i-1} (\text{mod} 2^k)$.
   If a_i exists before, then increase the current a_i to get a different number: $a_i \equiv 2a_{i-1} + 1 (\text{mod} 2^k)$
3. Keep looping until $i = 2^k - 1$.

In binary numeral system, multiplication by 2 simply means to left shift one position. The movement ensures that the $n-1$ least significant digits of the former number is the same as the the $n-1$ most significant digits of the later one. In other q-ary numeral system, we can multiple by q instead. There should be at most $q-1$ increments performed until new sequences are found.

2.6.4 Enumeration

The binary de Bruijn sequences given above are restricted by a rule that if the building k-size window already exists before, we should increase it so that the window is different. This rule accidentally limits the number of de Bruijn sequence compared to random binary string of n-bit. Therefore, an interesting criteria is to count the number of de Bruijn sequence we can build from n bits. This number increases as n increases. Therefore, we define the rate of the code as the following:

$$R(\mathbb{C}) = \frac{\log_\sigma |\mathbb{C}|}{n}$$

As each $\mathbb{C}$ might contain at least an empty set, this size cannot be zero. Besides, number of sequence in $\mathbb{C}$ can not exceed σ^n. As the result, we have obvious boundaries:

$$0 \leq R(\mathbb{C}) \leq 1$$

In the application of the sequence, we usually take care of numerous number of digits, in case $n \to \infty$. In order to deal with this infinity, only the locally-constrained de Bruijn code is capable of calculation. Hence, the rate in case $n \to \infty$, applied with locally-constrained de Bruijn code, such value is called *asymptotic rate*, is given as below:

$$R_{DB}(b,k) = \lim_{n\to\infty} \frac{\log |\mathbb{C}_{DB}(n,b,k)|}{n}$$

The asymptotic rate is more frequently concerned in real application as it indicates the effectiveness of the code. This effectiveness is usually obtained by finding a suitable value for a pair of (b,k).

2.6.5 Application

a, m-sequence

We start with defining the concept of m-sequence:

Suppose that we have the sequence: $x = (x_1, x_2, ..., x_n, ...)$ where the value of x_k

with $k \geq l$ is calculated recursively as below:

$$x_k = f(x_{k-l}, x_{k-l+1}, ..., x_{k-1})$$

Then we call x a *feedback shift register*. In case f is linear:

$$f(x_1, x_2, ...x_l) = \sum_{i=1}^{l} c_i x_{k+1-i}$$

We call x as a *linear feedback shift register*.

The sequence s with *maximum length* given n is called a m-sequence. We will now define a construction that can correct error in data transmission.

b, Error correcting sequence - Construction on $\mathbb{C}_{DB}(n, b, k)$

Let $\mathcal{S}_{\mathcal{P}_k}$ be the set of all cyclic m-sequences of order k of length $q^k - 1$ over the alphabet $\mathbb{F}_q$ where the unique run of k ones is at the end of each sequence. We define:

$$\mathbb{C} \triangleq \{(s_1 1^{\epsilon_1}, s_2 1^{\epsilon_2}, , ..., s_l 1^{\epsilon_l}), s_i \in \mathcal{S}_{\mathcal{P}_k}, 0 \leq \epsilon_i \leq k, 1 \leq i \leq l\}$$

This code is proved to be a $(q^k - 1, 2k)$ locally-constrained de Bruijn sequence in [4].

It is also proved in [4] that the size of $\mathbb{C}$ is M^l where $M = \dfrac{\phi(q^k - 1)(k + 1)}{k}$, $\phi(x)$ is the Euler function.

However, as the length of each codeword in $\mathbb{C}$ varies, we construct the following code whose codeword's length is fixed:

$$\mathbb{C}_1 = \{(s_1 s_2) | s_1 \in \mathbb{C}, s_2 \in \mathbb{C}', \text{length}(s_1 s_2) = l(q^k + k)\}$$

where $\mathbb{C}'$ is the set containing all prefixes of all codewords in $\mathbb{C}$.

As extend from $\mathbb{C}$, we have $|\mathbb{C}_1| \geq |\mathbb{C}|$.

It is also proved [4] that:

$$R(\mathbb{C}_1) \geq \frac{k}{q^k + k}$$

Given a (n, b, k)-locally-constrained de Bruijn sequences, it can correct any numbers of bursts of deletions of length at most $b - 1$ in the l-symbol read channel where $l = b + k - 1$. In case of $b = 2$, the $(n, 2, k)$-locally-constrained de Bruijn sequences can correct any number of stick-insertions in the $l-$symbol read channel in which

$k = l$ [4].

Example 2.1. Suppose that the binary sequence "00101101001" is intended to be read from the storage, and the read channel can read $l = 3$ digits once, the read sequence would be

$$001, 010, 101, 011, 110, 101, 010, \mathbf{100}, 001$$

If there is a sticky-insertion at the second and the forth position, and a deletion at the eighth position, the read output should be:

$$001, \mathbf{010, 010}, 101, \mathbf{011, 011}, 110, 101, 010, 001$$

CHAPTER 3. METHODOLOGY

3.1 Overview

My research in this thesis can be divided into three main works as following.

3.2 Dyck paths enumeration

The main variants of Dyck paths enumeration being considered in this thesis is as following.

1. Dyck paths up-1 down-m with bounded height enumeration.
   This part covers the case that the Dyck paths both start and end at the height 0, each up-step increase its height by 1, down-step decrease its height by m, and height is bounded by a maximum height k.
2. Dyck paths up-1 down-m with bounded height and flatten step enumeration.
   This part covers the case that the Dyck paths both start and end at the height 0, each up-step increase its height by 1, down-step decrease its height by m, height is bounded by a maximum height k, and flatten steps which remain the height is allowed at the height 0.
3. Dyck paths up-1 down-m with bounded height and bounded end enumeration.
   This part covers the case that the Dyck paths both start at the height 0, end at a height between 0 and μ, each up-step increase its height by 1, down-step decrease its height by m, and height is bounded by a maximum height k.

For each particular problem, the main steps used to determine the generating functions are:

1. Define a generating function with parameters that are suitable to count the desired words.
2. Decompose the generally desired word into compositions of empty word, digits and other shorter words.
3. Represent every word as its corresponding parametric generating function.
4. Solve the decomposition function to find out the general generating function.

Depending on particular problem, the generating function parameters will vary. However, the concepts will likely be the same. The results are also in the form of a fraction of two polynomials which is suitable to use the same Maclaurin series expansion method as mentioned in section 3.3.

3.3 Locally-constrained de Bruijn sequences enumeration

The main steps to enumerate the locally-constrained de Bruijn sequences using Goulden-Jackson inversion theorem can be listed as below:

1. Define *reduced forbidden words set* A.
   The *reduced forbidden words set* A should contain all elements that must not appear in counted locally-constrained de Bruijn sequences. In order for it to be called *reduced*, A should not contain two distinct elements such that one is the subsequence of the other. The set A must also be large enough so that every sequence that does not contain any element in A as subsequences is a locally-constrained de Bruijn sequence.
2. Establish equations system based on equation described in (2.8) and solve it.
   Depending on the boundary b, the number of element in A should increase as b increases. A uniquely parametric solution will be found. The solution will be used to calculated the cluster generating function. The uniqueness of solution will also be proved in next chapters.
3. Obtain cluster generating function and desired generating function.
   Feeding the cluster generating function gives the desired generating function counting the number of de Bruijn sequences of the same length.
4. Extract coefficients using Maclaurin series expansion method.
   Because all coefficients of equation (2.8) are polynomials of t, their determinants will also be polynomials. It means that the solution of the equations system for each weight($\mathcal{C}[v]$) will be rational, i.e. in the form of a fraction of two polynomials: $P(t)/Q(t)$ in which $P(t)$ and $Q(t)$ are two polynomials. The method used to extract coefficients is shown in Appendix A.2

3.4 State-splitting algorithm and encoder construction

The main steps to construct an encoder for locally-constrained de Bruijn code using state-splitting algorithms can be listed as below:

1. Choose a configuration of (b, k) and calculate the asymptotic rate r.
2. Find two numbers p, q such that $\frac{p}{q} \leq r$. This ratio will be the transfer rate of the communication channel which is being encoded using this configuration.
3. Let $\mathscr{G}$ denote the adjacency de Bruijn graph. Apply State-splitting algorithms to split vertices of the adjacency graph of $\mathscr{G}^q$ into a larger graph that is ready to build a encoder.
4. Tag each edge on the split graph to obtain an encoder.

4.1 Dyck path enumerations

In this section, we are studying Dyck path with many variants. In formal language, a Dyck word is a balanced string of brackets. The set of Dyck words is a Dyck language. In graph theory, the graph representation of Dyck word on the grid is a Dyck path. Depending on the weight each bracket is assigned, the Dyck path can varies in many ways.

In the following parts, we are studying the case of m-Dyck path in which total height is bounded.

4.1.1 1-Dyck path with bounded height

The number of 1-Dyck path is determined by the Catalan number:

$$a_{2n} = \frac{1}{n+1}\binom{2n}{n}$$

The $1-$Dyck path of bounded height is also studied using a transfer matrix method [18]. In this thesis, the study on $1-$Dyck path is generalized in $m-$Dyck path with bounded height in section 4.1.2 below.

4.1.2 m-Dyck path with bounded height

The number of m-Dyck path is evaluated by the Fuss-Catalan number:

$$a_{(m+1)n} = \frac{1}{mn+1}\binom{(m+1)n}{n}$$

In this part, we consider the case of m-Dyck path whose height is totally bounded by a maximum value, say k. To do so, we have the following Lemma:

Lemma 1. There is a recursive relation for the generating functions $g_k(x)$, for $k = m, m+1, \dots$ as follows

$$g_k(x) = 1 + x^m g_k(x) g_{k-1}(x) \cdots g_{k-m}(x). \tag{4.1}$$

Proof: Let w be a m-Dyck word, then either w is an empty word or it has at least one m-down step. If it is the latter case, by considering the last m-down step of w, we can decompose w uniquely as a concatenation of words in the following form

$$w = w_0 u w_1 u w_2 u w_3 .... u w_m d_m, \tag{4.2}$$

where w_i is an m-Dyck path weakly staying above the line $y = 0$. Figure 4.1 shows the unique decomposition of a Dyck path of maximum height k to other Dyck paths of smaller maximum heights.

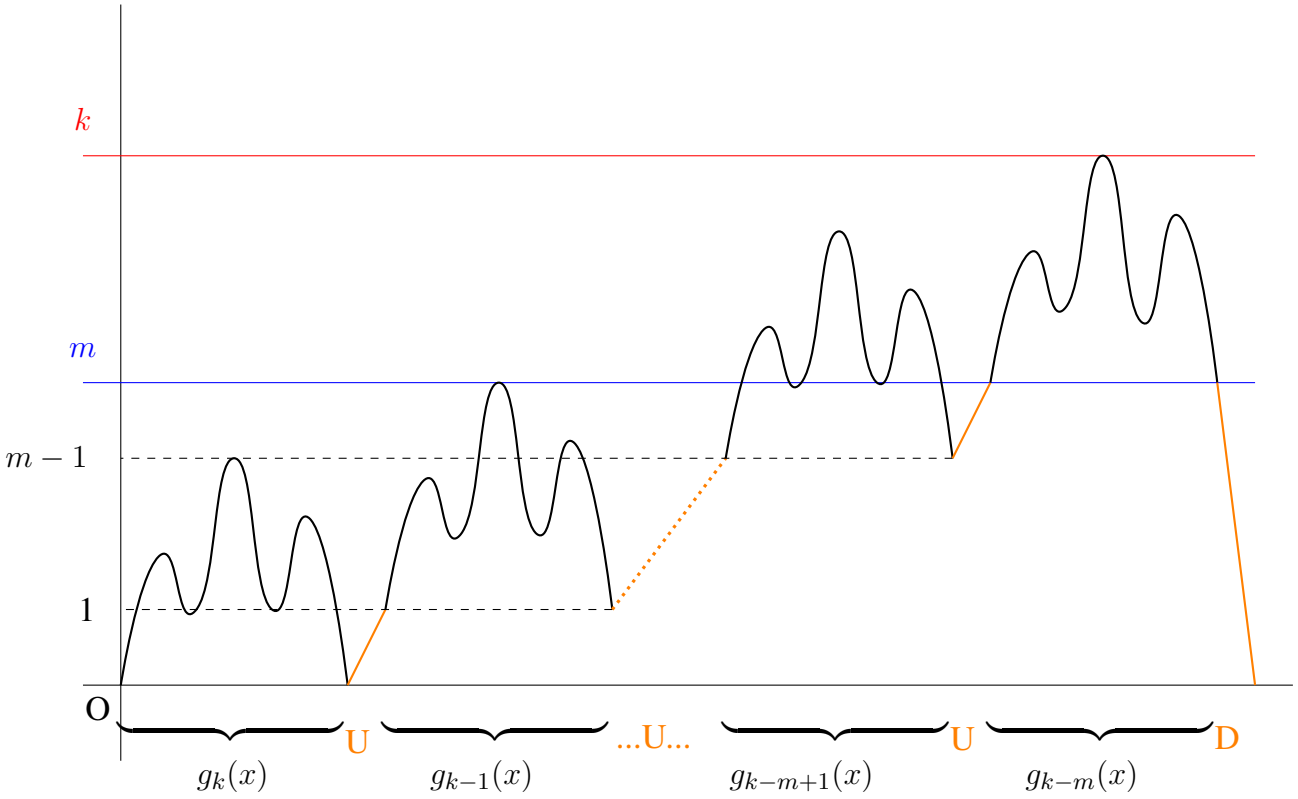


Figure 4.1: $(1, m)$-Dyck path decomposition

Translating this decomposition into terms of generating functions, we have

$$\begin{aligned} g_k(x) =& 1 + g_k(x) \cdot x g_{k-1}(x) x g_{k-2}(x) \cdot \cdots \cdot x g_{k-m}(x) \\ =& 1 + x^m g_k(x) g_{k-1}(x) \cdots g_{m-k}(x) \end{aligned} \tag{4.3}$$

■

Next we show explicitly $g_k(x)$ using the recursion above.

Theorem 1. The formula of generating function $g_k(x)$ is as below

$$g_k(x) = \frac{\sum\limits_{0 \le r(m+1) \le k} \binom{k - mr}{r} (-1)^r x^{(m+1)r}}{\sum\limits_{0 \le r(m+1) \le k+1} \binom{k + 1 - mr}{r} (-1)^r x^{(m+1)r}}.$$

Proof: We notice that for $k < m$, the only path satisfying that its height is less than m is the empty string, so $g_k(x) = 1 \forall k < m$. Hence, we recursively construct

functions $p_t(x)$, for $t = 0, 1, 2, \ldots$, as follows:

$$p_t(x) = \begin{cases} 1 & \text{if} \quad 0 \leq t \leq m \\ \dfrac{p_{t-1}(x)}{g_{t-1}(x)} & \text{if} \quad t > m \end{cases}$$

Then, $g_t(x) = \dfrac{p_t(x)}{p_{t+1}(x)}$ and (4.1) becomes

$$p_t(x) = p_{t+1}(x) + x^{m+1} p_{t-m}(x) \tag{4.4}$$

Let $P(z) = \sum_{t=0}^{\infty} p_t(x) z^t$. From (4.4) we have

$$\begin{aligned}
& p_t(x)z^{t+1} = p_{t+1}(x)z^{t+1} + x^{m+1}p_{t-m}(x)z^{t+1} \\
\Leftrightarrow \quad & \sum_{t=m}^{\infty} p_t(x)z^{t+1} = \sum_{t=m}^{\infty} p_{t+1}(x)z^{t+1} + \sum_{t=m}^{\infty} x^{m+1}z^{t+1}p_{t-m}(x) \\
\Leftrightarrow \quad & z\sum_{t=m}^{\infty} p_t(x)z^t = \sum_{t=m+1}^{\infty} p_t(x)z^t + x^{m+1}z^{m+1}\sum_{t=0}^{\infty} z^t p_t(x) \\
\Leftrightarrow \quad & z\left(P(z) - \sum_{t=0}^{m-1} p_t(x)z^t\right) = \left(P(z) - \sum_{t=0}^{m} p_t(x)z^t\right) + x^{m+1}z^{m+1}P(z) \qquad (4.5)
\end{aligned}$$

Substituting $p_t(x) = 1 \; \forall 0 \leq t \leq m$ into (4.5) yields:

$$\begin{aligned}
& zP(z) = (P(z) - 1) + (xz)^{m+1}P(z) \\
\Leftrightarrow \quad & P(z) = \frac{1}{1 - (z - (xz)^{m+1})} \\
\Leftrightarrow \quad & P(z) = \sum_{t=0}^{\infty} (z - (xz)^{m+1})^t \\
\Leftrightarrow \quad & P(z) = \sum_{t=0}^{\infty} \sum_{0 \leq r(m+1) \leq t} \binom{t - mr}{r} (-1)^r x^{r(m+1)} z^t. \qquad (4.6)
\end{aligned}$$

Therefore, for $t \geq m$:

$$p_t(x) = \sum_{0 \leq r(m+1) \leq t} \binom{t - mr}{r} (-1)^r x^{r(m+1)}. \tag{4.7}$$

Consequently,

$$g_k(x) = \frac{p_k(x)}{p_{k+1}(x)} \frac{\sum\limits_{0 \le r(m+1) \le k} \binom{k-mr}{r} (-1)^r x^{(m+1)r}}{\sum\limits_{0 \le r(m+1) \le k+1} \binom{k+1-mr}{r} (-1)^r x^{(m+1)r}}.$$

■

Remark 1. Due to symmetry, the behavior of $(1, m)$-Dyck path is the same as $(m, 1)$-Dyck path (just going from right to left on the $(1, m)$-Dyck path yields the $(m, 1)$-Dyck path. As a result, their generating functions are identical, which is $g_k(x)$ as evaluated above.

Example 4.1. In case $k = 3, m = 2$, the generating function is:

$$g_3(x) = \frac{p_3}{p_4} = \frac{\sum_{0 \le u \le 3, 3|u} \binom{3 - \frac{2u}{3}}{\frac{u}{3}} (-1)^{\frac{u}{3}} \times x^u}{\sum_{0 \le u \le 4, 3|u} \binom{4 - \frac{2u}{3}}{\frac{u}{3}} (-1)^{\frac{u}{3}} \times x^u}$$

$$= \frac{\binom{3}{0} (-1)^0 \times x^0 + \binom{1}{1} (-1)^1 \times x^3}{\binom{4}{0} (-1)^0 \times x^0 + \binom{2}{1} (-1)^1 \times x^3} = \frac{1 - x^3}{1 - 2x^3}$$

which expands to $1, 0, 0, 1, 0, 0, 2, 0, 0, 4, 0, 0, 8, 0, 0, 16, 0, 0, 32, 0, 0, 64, 0, 0, 128, 0, 0, 256, ...$ It seems that $a_{3n} = 2^{n-1}\ \forall n \ge 1$, this could be explained by inserting $n - 1$ triples of either UDD or DUD into the $3(n - 1)$ blanks of $U, D, \_, \_, .., \_, \_, D$ (in case of $(2, 1)-$Dyck path).

4.1.3 (m,1)-Dyck path with bounded height and flatten steps

In this part, we relax the $(m, 1)$-Dyck path by allowing it to have down steps at the x-axis. However, to protect the properties that the Dyck path never goes below the $x-$axis, we treat the down steps on $x-$axis as the *flatten steps* (the path moves horizontally and remains its height).

In other words, the Dyck paths now contain three kinds of steps: go up (along the vector $(1, 1)$), go down (along the vector$(1, -m)$), and go horizontally (along the vector $(1, 0)$), in which the flatten steps are only allowed in the $x-$axis.

To enumerate the $(m, 1)-$Dyck path with flatten steps at $x-$axis, we have the

following theorem:

Theorem 2. The generating function of cardinality of $(m,1)-$Dyck path with flatten steps at $x-$axis, height bounded by k is:

$$h_k(x) = \frac{p_k(x)}{p_{k+1}(x) - xp_k(x)}$$

Proof: Let W_n denote an $(m,1)$-Dyck path with n flatten steps, height bounded by k. Let R_k denote the set containing all these $(m,1)$-Dyck path. By treating $n+1$ positions among these flatten steps as $n+1$ possibly empty $(m,1)$-Dyck paths without flatten steps w_i. The figure demonstrating this treatment is illustrated in figure 4.2.

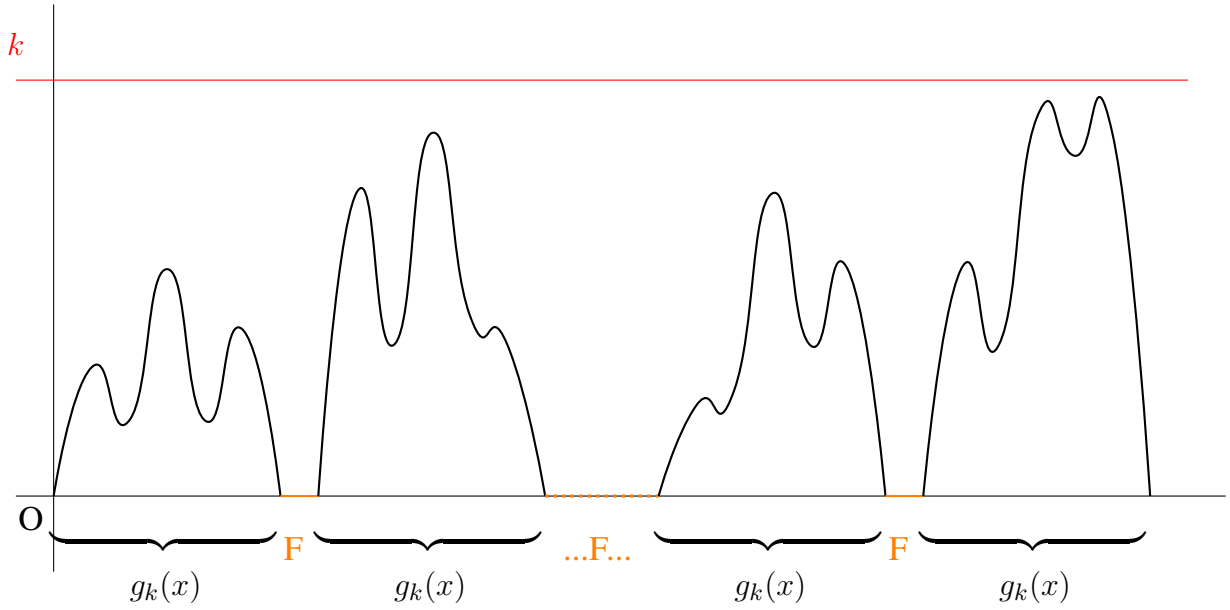


Figure 4.2: m-Dyck path decomposition with flat

Hence, we have the following decomposition:

$$W_n = w_1 F w_2 F w_3 F ... F w_n F w_{n+1} \tag{4.8}$$

Translating (4.8) into terms of generating functions, we have:

$$\begin{aligned} h_{k,n}(x) &= g_k(x) \times x g_k(x) \times x g_k(x) \cdots \times x g_k(x) \qquad \text{(contains } x \text{ } n \text{ times)} \\ &= g_k(x)(x g_k(x))^n \end{aligned}$$

Because a general Dyck path in R_k can contain arbitrary number of flatten steps,

we take the sum of $h_{k,n}$ with n from 0 to infinite as below:

$$h_k(x) = \sum_{i=0}^{\infty} h_{k,i}(x) = \sum_{i=0}^{\infty} g_k(x)(xg_k(x))^i = \frac{g_k(x)}{1 - xg_k(x)} = \frac{p_k(x)}{p_{k+1}(x) - xp_k(x)} \quad (4.9)$$

■

4.1.4 (1,m)-Dyck path with bounded height and bounded end

The natural Dyck path should not be restricted to end at a prescribed height (as being forced ending at 0 in the previous studies). Hence, in this part, we generalize the $(1, m)$-Dyck path by allowing it to end at any height $\mu(\mu < k)$

Firstly, we have the following proposition:

Proposition 1. *The generating function $g_{k,\mu}(x)$ for $(1, m)$-Dyck paths of height at most k and ending at $y = \mu$ is given by*

$$g_{k,\mu}(x) = \frac{p_{k-\mu}}{p_{k+1}} \times x^{\mu} = \frac{\sum\limits_{0 \le r(m+1) \le k-\mu} \binom{k-\mu-mr}{r} (-1)^r x^{r(m+1)+\mu}}{\sum\limits_{0 \le r(m+1) \le k+1} \binom{k+1-rm}{r} (-1)^r \times x^{r(m+1)}}.$$

Proof:

Let $g_{k,\mu}(x)$ denote the generating function counting $(1, m)$-Dyck paths which ends at an exact height of μ, and S denote its corresponding set of Dyck paths.

As each word $w_\mu \in S$ increases its height **one by one**, there must be some points that the path reaches every height from 1 to $\mu - 1$ before ending at μ. Let $a_i, 0 \le i \le \mu$ denote these last *break* positions the Dyck path reaches the height i. We have, obviously, $0 \le a_i < a_{i+1}\ \forall 0 \le i \le \mu - 1$.

We can now uniquely decompose w_μ into words $w \in S$ using μ delimiters at each position a_i as depicted in figure 4.3. Therefore,

$$g_{k,\mu}(x) = g_k(x) \times xg_{k-1}(x) \times xg_{k-2}(x) \times \cdots \times xg_{k-\mu+1}(x) \times xg_{k-\mu}(x) = \frac{p_{k-\mu}}{p_{k+1}} \times x^{\mu},$$

where polynomials $p_{k-\mu}(x)$ and $p_{k+1}(x)$ are defined in 4.7. ■

Let $G_{k,\mu}(x) = \sum_{i=0}^{\mu} g_{k,i}(x)$ denote the generating function counting $(1, m)$-Dyck paths which ends at height at most μ. Using the proposition above, we can enumerate the $(1, m)$-Dyck path by the following theorem:

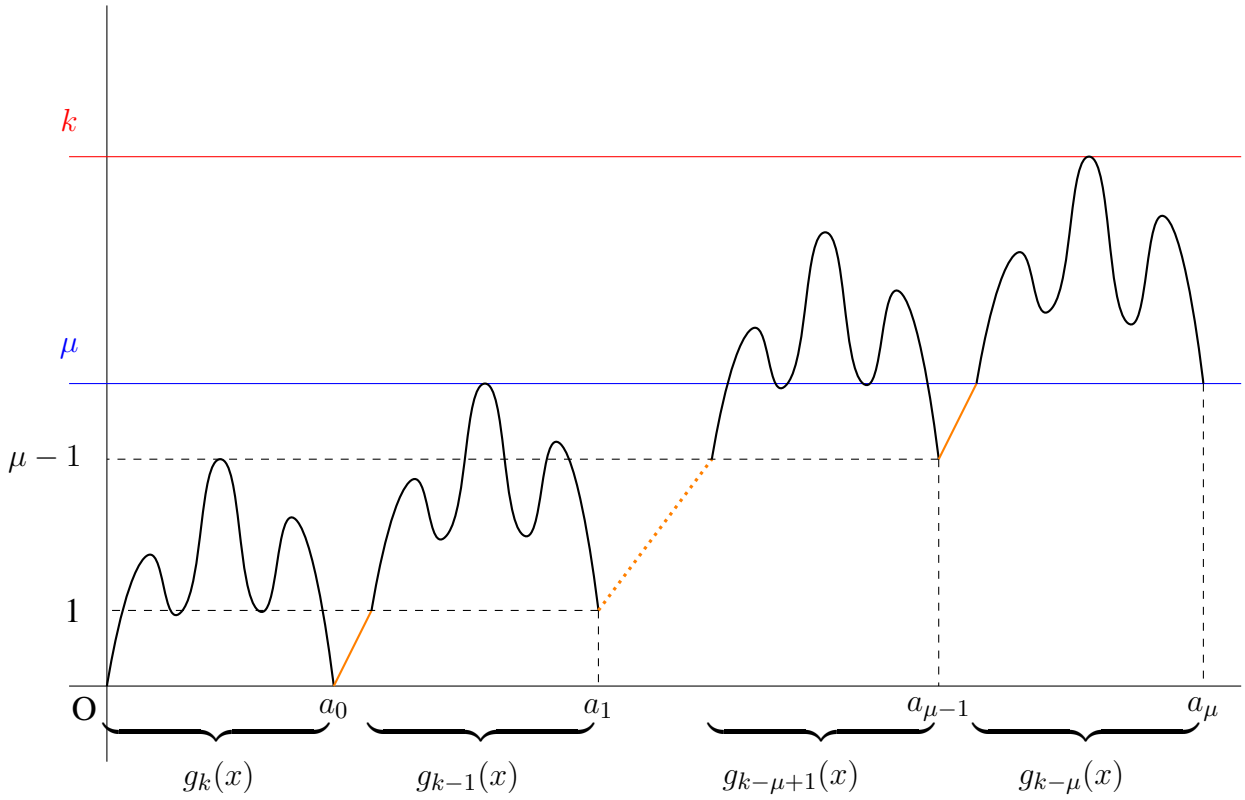


Figure 4.3: Bounded End Dyck path decomposition

Theorem 3. The formula of generating function $g_k(x)$ is as below:

$$G_{k,\mu}(x) = \frac{N(x)}{p_{k+1}} = \frac{\sum_{r=0}^{k} \sum_{s=max\left(0,\frac{r-\mu}{m+1}\right)}^{\frac{r}{m+1}} \binom{k-r+s}{s}(-1)^s \times x^r}{\sum_{0\le u\le \frac{k+1}{m+1}} \binom{k+1-um}{u}(-1)^u \times x^{u(m+1)}}$$

Proof: Substitute $g_{k,\mu}(x)$ from Proposition 1 to $G_{k,\mu}(x)$, we have:

$$G_{k,\mu}(x) = \frac{\sum_{i=0}^{\mu} p_{k-i} \times x^i}{p_{k+1}}$$

Consider the numerator $N(x) = \sum_{i=0}^{\mu} p_{k-i} \times x^i$, we have:

$$\begin{aligned}
N(x) &= \sum_{v=k-\mu}^{k} p_v \times x^{k-v} \\
&= \sum_{v=k-\mu}^{k} \sum_{0\le u\le v, m+1|u} \binom{v - \frac{um}{m+1}}{\frac{u}{m+1}} (-1)^{\frac{u}{m+1}} \times x^u \times x^{k-v} \\
&= \sum_{v=k-\mu}^{k} \sum_{u=0}^{\frac{v}{m+1}} \binom{v-um}{u}(-1)^u \times x^{k-v+u(m+1)}
\end{aligned}$$

Letting:

$$\begin{cases} u = s \\ k - v + u(m+1) = r \\ r, s \in \mathbb{N} \end{cases} \Rightarrow \begin{cases} u = s \\ v = k - r + s(m+1) \\ r, s \in \mathbb{N} \end{cases}$$

Then

$$\begin{cases} k - \mu \leq v \leq k \\ 0 \leq u \leq \frac{v}{m+1} \end{cases} \Rightarrow \begin{cases} k - \mu \leq k - r + s(m+1) \leq k \\ 0 \leq s \leq \frac{k-r}{m+1} + s \\ r, s \in \mathbb{N} \end{cases} \Rightarrow \begin{cases} 0 \leq r \leq k \\ max\left(0, \frac{r-\mu}{m+1}\right) \leq s \leq \frac{r}{m+1} \\ r, s \in \mathbb{N} \end{cases}$$

Therefore

$$N(x) = \sum_{r=0}^{k} \sum_{s=max\left(0, \frac{r-\mu}{m+1}\right)}^{\frac{r}{m+1}} \binom{k-r+s}{s} (-1)^s \times x^r$$

which yields the formula:

$$G_{k,\mu}(x) = \frac{N(x)}{p_{k+1}} = \frac{\sum_{r=0}^{k} \sum_{s=max\left(0, \frac{r-\mu}{m+1}\right)}^{\frac{r}{m+1}} \binom{k-r+s}{s} (-1)^s \times x^r}{\sum_{0 \leq u \leq \frac{k+1}{m+1}} \binom{k+1-um}{u} (-1)^u \times x^{u(m+1)}}$$

■

4.2 Locally-constrained de Bruijn sequences enumeration

4.2.1 Reduced forbidden words set

As mentioned in section 3.3, the *reduced forbidden words set*, let us call it $\mathcal{B}$, needs to be defined first. By our expected behavior, $\mathcal{B}$ is the set containing all words that are not allowed to appear in each word w we are counting (i.e., our locally-constrained de Bruijn sequence of length n). Therefore, the set $\mathcal{B}$ should contain all words that violate the condition of the locally-constrained de Bruijn sequence. Recall the condition, that a sequence s is called a (n, b, k) locally-constrained de Bruijn sequence if and only if each window of size k is unique around its $b-1$ neighbors to both side. In other words:

$$s[i, i+k-1] \neq s[j, j+k-1]$$

$$\forall i \neq j, 1 \leq i, j \leq n-k+1, |i-j| \leq b-1$$

In order to violate this constraint, we let a window of size k repeat itself within its $b-1$ neighbors. Besides, $\mathcal{B}$ is also reduced, so the repetition of each size-k window is unique, so we need the simplest form of repetition. That is, if w is a word in $\mathcal{B}$, then we can fix two size-k windows, one at the beginning and the other at the end, of the word w to be the same. Therefore, we have the following observation:

`Observation 1:`

$$w[0, k-1] = w[l-k, l-1] \tag{4.10}$$

As we are only concerned about the repetition of size-k windows within a boundary b, we can also limit the length of each forbidden word from $k+1$ (as it should contain at least two different windows of size k) to $b+k-1$ (as the farthest distance between two words is $b-1$). Therefore, we have the second observation:

`Observation 2:`

$$k+1 \leq |w| \leq b+k-1 \tag{4.11}$$

From these two observations, we run a simple program, for each word's length from $k+1$ to $b+k-1$ and generate their corresponding forbidden words if satisfied. We also check, for each word, whether there currently exists a word with shorter length that is one of its sub-sequence. In order to do so, the words need to be considered from the shortest length to the longest one. The simple algorithm is shown in Appendix A.1:

We have the set of fobidden words given boundary b and windows size k as following. For $b=3$ and $k=2$, $\mathcal{B}$ contains 4 forbidden words:

$$\mathcal{B}_{b=3,k=2} = \{000, 111, 0101, 1010\}$$

For $b=4$ and $k=3$, $\mathcal{B}$ contains 10 forbidden words:

$$\mathcal{B}_{b=4,k=3} = \{0000, 1111, 01010, 10101, 001001, 110110, 010010, 101101, 011011, 100100\}$$

For $b=5$ and $k=3$, $\mathcal{B}$ contains 22 forbidden words:

$$\begin{aligned}\mathcal{B}_{b=5,k=3} = \{&0000, 1111, 0001000, 1110111, 001001, 110110, 0010001, 1101110, 0011001,\\ &1100110, 01010, 10101, 010010, 101101, 0100010, 1011101, 011011,\\ &100100, 0110011, 1001100, 0111011, 1000100\}\end{aligned}$$

4.2.2 Weight of a word or marked word

With the same concept of forbidden-words, the weight of each word w on each subset S of $\mathcal{B}(w)$ is also defined in the same way as mentioned in (2.2). This defines the weight of a word w as:

$$\text{weight}(w) = \sum_{S \subseteq \mathcal{B}(w)} \text{weight}(w, S) = \sum_{S \subseteq \mathcal{B}(w)} (-1)^{|S|} t^{|w|}$$

4.2.3 Evaluating generating function

We are now ready to find the generating function representing the cardinality of the code. Firstly, we will illustrate the case of $b = 3$ and $k = 2$. Using this example, we will develop and generalize with a larger value of b and k in the next parts.

a, (3,2)-locally-constrained de Bruijn sequence

Problem 1. Find the generating function of cardinality of the $(n, 3, 2)$ locally-constrained de Bruijn code.

Solution: The forbidden-words set is $\mathcal{B}_{b=3,k=2} = \{000, 111, 0101, 1010\}$. Using the formula (2.8), we establish four equations as below:

$$\text{weight}(\mathcal{C}[000]) = -t^3 - t^2.[\text{weight}(\mathcal{C}[000]) + \text{weight}(\mathcal{C}[0101])] - t.\text{weight}(\mathcal{C}[000]) \tag{4.12}$$

$$\text{weight}(\mathcal{C}[111]) = -t^3 - t^2.[\text{weight}(\mathcal{C}[111]) + \text{weight}(\mathcal{C}[1010])] - t.\text{weight}(\mathcal{C}[111]) \tag{4.13}$$

$$\begin{aligned}\text{weight}(\mathcal{C}[0101]) = -t^4 - t^3[&\text{weight}(\mathcal{C}[1010]) + \text{weight}(\mathcal{C}[111])] \\ &- t^2\text{weight}(\mathcal{C}[0101]) - t.\text{weight}(\mathcal{C}[1010])\end{aligned} \tag{4.14}$$

$$\begin{aligned}\text{weight}(\mathcal{C}[1010]) = -t^4 - t^3[&\text{weight}(\mathcal{C}[0101]) + \text{weight}(\mathcal{C}[000])] \\ &- t^2\text{weight}(\mathcal{C}[1010]) - t.\text{weight}(\mathcal{C}[0101])\end{aligned} \tag{4.15}$$

We might notice that the equations (4.12) and (4.13) are similar. Just switching 0 and 1 in one equation gives the other one. Therefore, due to symmetry, we can state that weight($\mathcal{C}[000]$) = weight($\mathcal{C}[111]$) = $\mathscr{W}_1$.

The same thing happens to weight($\mathcal{C}[0101]$) and weight($\mathcal{C}[1010]$). Hence, we also have weight($\mathcal{C}[0101]$) = weight($\mathcal{C}[1010]$) = $\mathscr{W}_3$.

The equations system can be re-written as following:

$$\begin{cases} \mathscr{W}_1 &= -t^3 - t^2.(\mathscr{W}_1 + \mathscr{W}_3) - t.\mathscr{W}_1 \\ \mathscr{W}_3 &= -t^4 - t^3(\mathscr{W}_3 + \mathscr{W}_1) - t^2\mathscr{W}_3 - t.\mathscr{W}_3 \end{cases}$$

Solving this equation gives the parametric solution:

$$\begin{cases} \mathscr{W}_1 &= -\dfrac{t^5 + t^4 + t^3}{2t^4 + 3t^3 + 3t^2 + 2t + 1} \\ \mathscr{W}_3 &= -\dfrac{t^5 + t^4}{2t^4 + 3t^3 + 3t^2 + 2t + 1} \end{cases}$$

With these weights, the cluster generating function is:

$$\begin{aligned} \text{weight}(\mathcal{C}) &= \sum \mathscr{W} = 2(\mathscr{W}_1 + \mathscr{W}_3) \\ &= \frac{4t^5 + 4t^4 + 2t^3}{2t^4 + 3t^3 + 3t^2 + 2t + 1} \end{aligned}$$

Substitute this result to (2.4), with $\mathcal{A} = \{0, 1\}$, we obtain the generating function of cardinality of $(n, 3, 2)$ locally-constrained de Bruijn code:

$$\begin{aligned} f(t) = \text{weight}(\mathcal{M}) &= \frac{1}{1 - |\mathcal{A}|t - \text{weight}(\mathcal{C})} \\ &= \frac{2t^4 + 3t^3 + 3t^2 + 2t + 1}{-t^3 - t^2 + 1} \end{aligned}$$

■

b, (4,k)-locally-constrained de Bruijn sequence

Using the same approach, in this section, we will try to work with a larger boundary b and windows size k as following.

Problem 2. *Find the generating function of cardinality of the* $(n, 4, 3)$ *locally-constrained de Bruijn code.*

Solution: For $b = 4$ and $k = 3$, $\mathcal{B}$ contains 10 forbidden-words:

$$\mathcal{B}_{b=4,k=3} = \{0000, 1111, 01010, 10101, 001001, 110110, 010010, 101101, 011011, 100100\}$$

We also notice that due to symmetry, there are five pairs whose weights are equal. They are:

- weight($\mathcal{C}$[0000]) = weight($\mathcal{C}$[1111]) = $\mathscr{W}_1$
- weight($\mathcal{C}$[01010]) = weight($\mathcal{C}$[10101]) = $\mathscr{W}_2$
- weight($\mathcal{C}$[001001]) = weight($\mathcal{C}$[110110]) = $\mathscr{W}_3$
- weight($\mathcal{C}$[010010]) = weight($\mathcal{C}$[101101]) = $\mathscr{W}_4$
- weight($\mathcal{C}$[011011]) = weight($\mathcal{C}$[100100]) = $\mathscr{W}_5$

Using the same manner, the equations system is established as below:

$$\begin{cases} \mathscr{W}_1 = -t^4 - (t^3+t^2+t)\mathscr{W}_1 - t^3\mathscr{W}_2 - (t^3+t^2)\mathscr{W}_3 - t^3\mathscr{W}_4 - t^3\mathscr{W}_5 \\ \mathscr{W}_2 = -t^5 - t^4\mathscr{W}_1 - (t^4+t^3+t^2+t)\mathscr{W}_2 - t^4\mathscr{W}_3 - (t^4+t^3+t^2)\mathscr{W}_4 - (t^4+t^3)\mathscr{W}_5 \\ \mathscr{W}_3 = -t^6 - t^5\mathscr{W}_1 - (t^5+t^4)\mathscr{W}_2 - (t^5+t^3)\mathscr{W}_3 - (t^5+t^4+t)\mathscr{W}_4 - (t^5+t^4+t^2)\mathscr{W}_5 \\ \mathscr{W}_4 = -t^6 - t^5\mathscr{W}_1 - (t^5+t^4+t^3)\mathscr{W}_2 - (t^5+t^2)\mathscr{W}_3 - (t^5+t^4+t^3)\mathscr{W}_4 - (t^5+t^4+t)\mathscr{W}_5 \\ \mathscr{W}_5 = -t^6 - (t^5+t^4)\mathscr{W}_1 - t^5\mathscr{W}_2 - (t^5+t^4+t)\mathscr{W}_3 - (t^5+t^2)\mathscr{W}_4 - (t^5+t^3)\mathscr{W}_5 \end{cases}$$

Solving this equation gives the parametric solution:

$$\begin{cases} \mathscr{W}_1 &= \dfrac{-t^{13}+t^8+2t^7+t^6+t^5+t^4}{2t^{13}+4t^{12}-t^{10}-2t^9-2t^8-3t^7-8t^6-9t^5-7t^4-5t^3-3t^2-2t-1} \\ \mathscr{W}_2 &= \dfrac{-t^{14}+t^{11}+t^{10}+t^7+t^6+t^5}{2t^{13}+4t^{12}-t^{10}-2t^9-2t^8-3t^7-8t^6-9t^5-7t^4-5t^3-3t^2-2t-1} \\ \mathscr{W}_3 &= \dfrac{-t^{14}-t^{13}+t^{12}+t^{11}+t^{10}+t^9+t^8+t^7+t^6}{2t^{13}+4t^{12}-t^{10}-2t^9-2t^8-3t^7-8t^6-9t^5-7t^4-5t^3-3t^2-2t-1} \\ \mathscr{W}_4 &= \dfrac{t^{12}-t^{11}-t^{10}+t^7+t^6}{2t^{13}+4t^{12}-t^{10}-2t^9-2t^8-3t^7-8t^6-9t^5-7t^4-5t^3-3t^2-2t-1} \\ \mathscr{W}_5 &= \dfrac{-t^{13}+t^{10}+t^7+t^6}{2t^{13}+4t^{12}-t^{10}-2t^9-2t^8-3t^7-8t^6-9t^5-7t^4-5t^3-3t^2-2t-1} \end{cases}$$

With these weights, the cluster generating function is:

$$\text{weight}(\mathcal{C}) = \sum \mathscr{W} = 2(\mathscr{W}_1 + \mathscr{W}_2 + \mathscr{W}_3 + \mathscr{W}_4 + \mathscr{W}_5)$$

which yields the final generating function: $f(t) = \dfrac{1}{1 - |\mathcal{A}|t - \text{weight}(\mathcal{C})} = \dfrac{A}{B}$
where

$$\begin{aligned} A &= 2t^{13}+4t^{12}-t^{10}-2t^9-2t^8-3t^7-8t^6-9t^5-7t^4-5t^3-3t^2-2t-1 \\ B &= -t^{10}+t^7+t^5+t^4+t^3+t^2-1 \end{aligned}$$

■

4.2.4 Feasibility of the method

a, The stability of the forbidden word set

One might notice that the importance of this method is to solve the system of equations whose number of variables (half of the size of the forbidden-words set $\mathcal{B}$) increases significantly as b increases. For $b = 4$, we have to solve the system of 5 equations. For $b = 5$, in fact, we have to solve the system of 11 equations which takes us hours, compared to seconds in the case of $b = 4$ (the source code is available at *github.com/nguyenluc99/GJ_LC_deBruijnCode*). However, for a particular b, the number of variables remains as k increases. We will prove that for $b = 4$, there are always 10 forbidden-words in $\mathcal{B}$.

Theorem 4. For any windows size k with boundary $b = 4$, the forbidden-words set $\mathcal{B}$ contains 10 words.

Proof: Recall from (4.10) that a sequence $s = (s_1, s_2, ..., s_k)$ repeats itself in $w = (w_1, w_2, ..., w_l)$ within a boundary $b = 4$. This means that $1 \leq l - k \leq b - 1 = 3$. Let $w[i]$ denote the i-th digit of w, and $w[i, j] = (w_i, w_{i+1}, ..., w_j)$ $\forall 1 \leq i \leq j \leq l$. We have three following cases:

- $l - k = 1$, it means that $w[1, k] = w[2, k + 1] \Rightarrow w[1] = w[2] = ... = w[k] = w[k + 1]$. Letting each digit be 0 and 1 gives two desired words of length $k + 1$ containing all zero and one, 000...00 and 111...11.
- $l - k = 2$, it means $w[1, k] = w[3, k + 2]$, yielding:

$$\begin{cases} w[1] = w[3] = ... = w[2m + 1] & (2m \leq k + 1) \\ w[2] = w[4] = ... = w[2m'] & (2m' \leq k + 2) \end{cases}$$

  There are four ways to initialize the pair of $(w[1], w[2])$. However, two pairs $(0, 0)$ and $(1, 1)$ form two sequences of zero-s and one-s of length $k + 2$, which respectively contains two words in the first case, $l - k = 1$, making the forbidden-words set $\mathcal{B}$ reducible. The other pairs, $(0, 1)$ and $(1, 0)$, result two satisfied sequences of length $k+2$, $010101...$ and $101010.....$ Depending on the length k of each window, the last digit can be either zero or one.

- $l - k = 3$, it means $w[1, k] = w[4, k + 3]$, yielding:

$$\begin{cases} w[1] = w[4] = ... = w[3m + 1] & (3m \leq k + 2) \\ w[2] = w[5] = ... = w[3m' + 2] & (3m' \leq k + 1) \\ w[3] = w[6] = ... = w[3m''] & (3m'' \leq k + 3) \end{cases}$$

Similar to the case $l-k=2$, we also exclude two initialization of $(w[1], w[2], w[3])$, $(0,0,0)$ and $(1,1,1)$, giving six satisfied triples of $(w[1], w[2], w[3])$ and their corresponding forbidden-words of length $k+3$ as following:

1. $(w[1], w[2], w[3]) = (0,0,1) \Rightarrow w = 001001...$
2. $(w[1], w[2], w[3]) = (0,1,0) \Rightarrow w = 010010...$
3. $(w[1], w[2], w[3]) = (0,1,1) \Rightarrow w = 011011...$
4. $(w[1], w[2], w[3]) = (1,0,0) \Rightarrow w = 100100...$
5. $(w[1], w[2], w[3]) = (1,0,1) \Rightarrow w = 101101...$
6. $(w[1], w[2], w[3]) = (1,1,0) \Rightarrow w = 110110...$

Once again, the last digits of each forbidden word are not important.

In conclusion, for any value of k given $b = 4$, $\mathcal{B}$ always contains 10 forbidden-words. ■

b, The existence of the generating function

In this part, we are concerning about the reliability of the method. After obtaining the forbidden word set $\mathcal{B}$, we can establish the equations system and solve it. The major concern is that the general equations system might not have a unique solution if its determinant is zero. To clarify this point, we have the following theorem:

Theorem 5. *The equations system obtained from the formula* (2.8)*, given $\mathcal{C}$ is the cluster on (n,b,k) locally-constrained de Bruijn code, has a unique solution.*

Proof: Let $\mathcal{B}$ be a forbidden word set of a (b,k) locally-constrained de Bruijn code. (w) is an operation that switches each digit of w from 0 to 1 and 1 to 0. Suppose that w is an element in $\mathcal{B}$. Due to symmetry, the word (w) is also a different forbidden word. Therefore, the size of $\mathcal{B}$ is always even. Let $|\mathcal{B}| = 2n$, $\mathcal{B}' = \{w_1, w_2, ..., w_n\}$ denotes the subset of $\mathcal{B}$ whose words start with 0, and $(\mathcal{B}') = \{(w) | w \in \mathcal{B}'\}$. In other words, we split the set $\mathcal{B}$ into two equal and disjoint subsets, $\mathcal{B}'$ and $(\mathcal{B}')$, such that each element in $(\mathcal{B}')$ can be generated from one and only one corresponding element in $\mathcal{B}'$ by the (w) operation.

Because of the symmetry between w_i and (w_i), we always have weight$(\mathcal{C}[w_i]) =$ weight$(\mathcal{C}[(w_i)]) = \mathscr{W}(1 \leq i \leq n)$. Therefore, from (2.8), n equations generated from n words of $\mathcal{B}'$ is sufficient to solve and find the desired cluster generating function. For each of these words, the equation (2.8) is now re-written as following:

$$\text{weight}(\mathcal{C}[v]) = -t^{|v|} - \sum_{u \in \mathcal{B}'} \left(\text{weight}(\mathcal{C}[u]) \cdot \sum_{r \in O(v,u)} t^{|v \backslash r|} + \text{weight}(\mathcal{C}[(u)]) \cdot \sum_{r \in O(v,(u))} t^{|v \backslash r|} \right)$$

$$\Rightarrow \mathscr{W} = -t^{|w_i|} - \sum_{w_j \in \mathcal{B}'} \left[\mathscr{W}_j \cdot \left(\sum_{r \in O(w_i,w_j)} t^{|w_i \backslash r|} + \sum_{r \in O(w_i,(w_j))} t^{|w_i \backslash r|} \right) \right]$$

$$\Leftrightarrow \mathscr{W} + \sum_{w_j \in \mathcal{B}'} \left[\mathscr{W}_j \cdot \left(\sum_{r \in O(w_i,w_j)} t^{|w_i \backslash r|} + \sum_{r \in O(w_i,(w_j))} t^{|w_i \backslash r|} \right) \right] = -t^{|w_i|} \tag{4.16}$$

where $\mathscr{W} = \text{weight}(\mathcal{C}[w_i])$

Let ${}_{ij}$ denote the t-parametric coefficient of $\mathscr{W}_j$ in the equation generated from the word $w_i \in \mathcal{B}'$. No matter what the last digit w_i is, it will overlap with either w_j (a word starting with 0) or (w_j) (a word starting with 1). This means the overlap parts must contain at least one part, $r = w_i[|w_i|] \Rightarrow w_i \backslash r = w_i[1 : |w_i| - 1] \Rightarrow t^{|w_i \backslash r|} = t^{|w_i|-1}$. This fact makes coefficients of all $\mathscr{W}_j$ of (4.16) non-zero. Let $\gamma({}_{ij})$ be the minimum order of t in ${}_{ij}$ whose coefficient is non-zero, and $[t^k]({}_{ij})$ be the coefficient of t^k in the expression of ${}_{ij}$. As a result, $[t^{|w_i|-1}]({}_{ij}) = 1 \Rightarrow \gamma({}_{ij}) \leq |w_i| - 1$.

In the equation (4.16) obtained from a word w_i, without loss of generality, suppose that $r \in O(w_i, w_j)$. Because the set $\mathcal{B}'$ is reduced and the last part of w_i, r, is the first part of w_j, at least one digit of w_i, the first digit, must not be contained in $r \Rightarrow |w_i \backslash r| \geq 1 \; \forall r \in O(w_i, w_j)$. Hence, $\gamma({}_{ij}) \geq 1 \; \forall i \neq j \Rightarrow [t^0]({}_{ij}) = 0 \; \forall i \neq j$. Besides, as the coefficient of $\mathscr{W}_i$ at the first term is 1, $[t^0]({}_{ii}) = 1$, meaning that $\gamma({}_{ii}) = 0 \; \forall i$.

Denote $A = [{}_{ij}]$ to be the squared t-parametric coefficients matrix of all equations. We now consider its determinant by Laplace expansion on the i-th row as following:

$$\det A = \sum_{j=1}^{n} (-1)^{i+j} {}_{ij} \times \det A_{ij} \tag{4.17}$$

where A_{ij} is the matrix obtained by removing the i-th row and j-th column of A. It is also a matrix of polynomials whose degrees are non-negative. Hence:

$$\gamma(\det A_{ij}) \geq 0$$

$$\Rightarrow \gamma({}_{ij} \times \det A_{ij}) \geq 1 \; \forall i \neq j$$

$$\Rightarrow [t^0]({}_{ij} \times \det A_{ij}) = 0 \; \forall i \neq j$$

Consider the power of zero of t in both sides of (4.17), as $[t^0]({}_{ii}) = 1$, we have:

$$\begin{aligned}[t^0](\det A) &= [t^0]({}_{ii} \times \det A_{ii}) + \sum_{1 \leq j \leq n, i \neq j} [t^0]\left((-1)^{i+j}{}_{ij} \times \det A_{ij}\right) \\ &= [t^0](\det A_{ii})\end{aligned}$$

We can see a recursive relation that:

$$\begin{aligned}[t^0](\det A) &= [t^0](\det A_{11}) \\ &= [t^0](\det (A_{11})_{11}) \\ &= ... \\ &= [t^0](\det ((...(A_{11})_{11})...)_{11}) \quad (n-1 \text{ times}) \\ &= [t^0]({}_{nn}) = 1\end{aligned}$$

Therefore, $\det A \not\equiv 0$. This means that the equations system obtained from (2.8) always has a unique solution. ■

The proof above also shows that the method can be theoretically applied for any boundary b and windows size k.

4.3 State-splitting algorithm and encoder construction

Let S be the $(3, 2)$ locally-constrained de Bruijn constrained system. The labeled graph $\mathscr{G}$ representing S is shown in figure 4.4. As the rate for this case is proved [4] to be about 0.4056 and observing that $\frac{2}{5} = 0.4 < 0.4056$, we choose $p = 2, q = 5$ to construct an encoder.

Firstly, the adjacency matrix of $\mathscr{G}^5$ is given in figure 4.5. In order for the graph to be an encoder with asymptotic rate $\frac{p}{p} = \frac{2}{5} = 0.4$, the minimum out-degree of every vertex must be at least $2^p = 4$. Let $\delta(\mathscr{G})$ denote the minimum out-degree of $\mathscr{G}$. At the moment, $\delta(\mathscr{G}) = 3$, so we need to split some more times.

Apply the Franaszek algorithm to find an $(A_G^5, 2^2)$-approximate eigence-vector, we get $\boldsymbol{x} = (3, 4, 4, 3, 2, 2)$. Thus each vertex can be split into either two, three or four vertices. After completely splitting all vertices, making the approximate eigence-vector become an all-one vector of length 18, we obtain a labeled graph

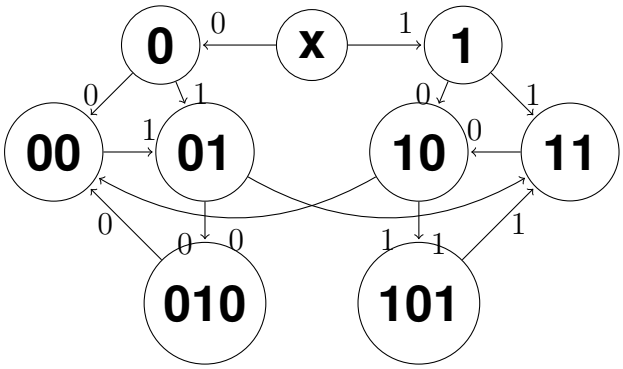


Figure 4.4: State diagram for the (3, 2)-locally-constrained de Bruijn sequences

$$\begin{pmatrix} 0 & 1 & 0 & 2 & 1 & 0 \\ 1 & 0 & 2 & 1 & 1 & 0 \\ 1 & 2 & 0 & 1 & 0 & 1 \\ 2 & 0 & 1 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 & 1 & 0 \end{pmatrix}$$

Figure 4.5: Adjacency matrix of $\mathscr{G}^5$

$\mathscr{G}'$ satisfying that $\delta(\mathscr{G}'\} = 4$, which is just enough to be an finite state encoder with rate $2 : 5$. The detailed process on splitting these vertices is shown in Appendix A.4

After tagging and removing redundant edges of $\mathscr{G}'$ (which makes number of out-going edges of a states is at least than 4), $\mathscr{G}'$ can be used to build an encoder. The lookup table of the encoder is represented in table 4.1.

Example 4.2. Consider $\boldsymbol{x} = 0010011111$, with a starting state is $00c$, we split $\boldsymbol{x}$ into five words of length $p = 2$ (00, 10, 01, 11, and 11). Then the encoded outputs together with corresponding next states after $00c$ are $(10011, 11c)$, $(01100, 00c)$, $(11011, 11c)$, $(01101, 101b)$, and $(10010, 010a)$. Hence, the encoded words is $1001101100110110110110010$.

Current state	Input							
	00		01		10		11	
	Output	Next state	Output	Next state	Output	Next state	Output	Next state
00a	11001	01a	11001	01b	11001	01c	11001	01d
00b	10011	11a	11011	11a	10011	11b	11011	11b
00c	10011	11c	11011	11c	10010	010a	10010	010b
01a	00110	10a	10110	10a	00110	10b	10110	10b
01b	00110	10c	10110	10c	00110	10d	10110	10d
01c	00100	00a	00100	00b	00100	00c	10011	11a
01d	10011	11b	10011	11c	10010	010a	10010	010b
10a	01001	01a	01001	01b	01001	01c	01001	01d
10b	11001	01a	11001	01b	11001	01c	11001	01d
10c	11011	11a	11011	11b	11011	11c	01101	101a
10d	01100	00a	01100	00b	01100	00c	01101	101b
11a	00110	10a	00110	10b	00110	10c	00110	10d
11b	00100	00a	00100	00b	00100	00c	01101	101a
11c	01100	00a	01100	00b	01100	00c	01101	101b
010a	01001	01a	01001	01b	01001	01c	01001	01d
010b	01100	00a	01100	00b	01100	00c	01101	101a
101a	10110	10a	10110	10b	10110	10c	10110	10d
101b	10011	11a	10011	11b	10011	11c	10010	010a

Table 4.1: Lookup table for finite state encoder constructed from (3, 2) locally-constrained de Bruijn code.

5.1 Evaluation Parameters

In this section, we model the results of enumerating the $(3, k)$ and $(4, k)$-locally-constrained de Bruijn sequences in comparison with the rate that is evaluated from the Perron-Frobenius theory [19].

5.2 Simulation Method

To evaluate the result, we extract the coefficients of the (b, k)-locally-constrained de Bruijn sequences generating function given in 4.2.3. Recalling that the generating function of (b, k)-locally-constrained de Bruijn sequences has the form of fraction of polynomials, the coefficient extraction method is shown in Appendix A.2.

The series are then be used to compute the asymptotic rates which simulate their rate convergences in particular cases.

5.3 (3,k) and (4,k)-locally-constrained de Bruijn code rate convergences.

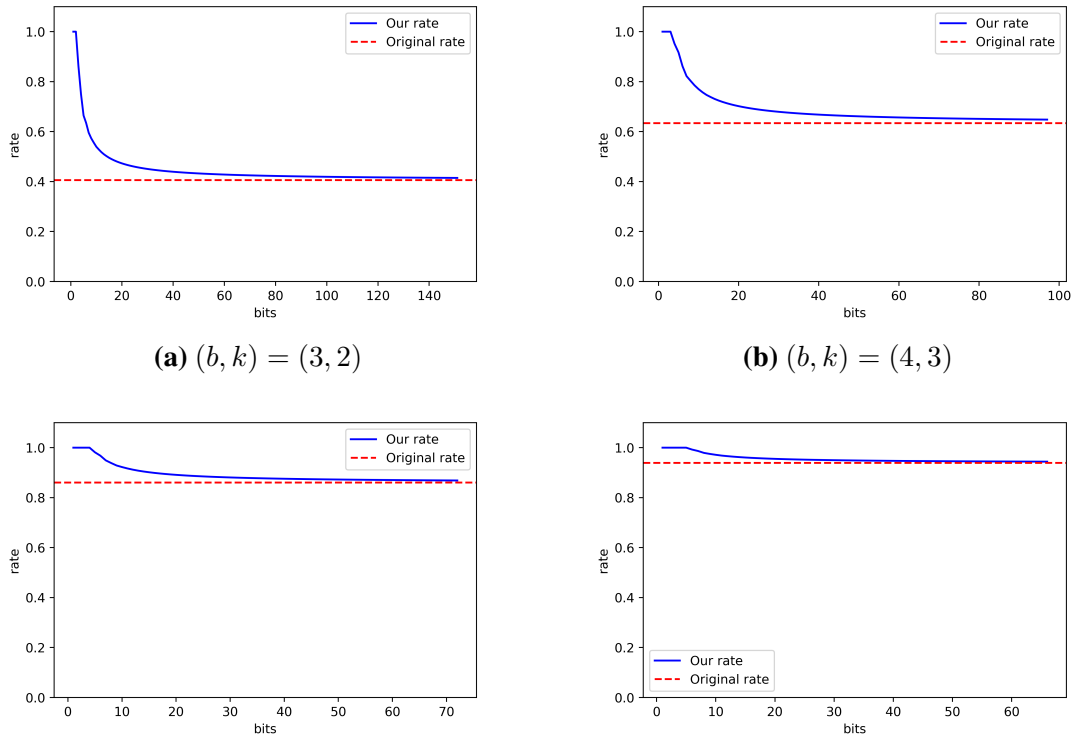
The generating functions are now used to extract the coefficients in each particular cases of $(3, k)$ and $(4, k)$. In combination with the rate evaluated by the Perron-Frobenius theorem from the group of authors [4], we summarize the result as following:

1. $(b, k) = (3, 2)$:
   - Capacity: 0.4056
   - Extracted series: 1, 2, 4, 6, 8, 10, 14, 18, 24, 32, 42, 56, 74, 98, 130, 172, 228, 302, 400, 530, 702, 930, ...
2. $(b, k) = (4, 3)$:
   - Capacity: 0.6341
   - Extracted series: 1, 2, 4, 8, 14, 24, 36, 54, 86, 134, 208, 322, 498, 776, 1202, 1866, 2896, 4496, 6976, 10824, 16802, 26072, 40466, ...
3. $(b, k) = (4, 4)$:
   - Capacity: 0.8600
   - Extracted series: 1, 2, 4, 8, 16, 30, 56, 100, 182, 328, 598, 1086, 1970, 3574, 6488, 11776, 21374, 38798, 70420, 127810, 231982, 421068, ...
4. $(b, k) = (4, 5)$:
   - Capacity: 0.9392

- Extracted series: 1, 2, 4, 8, 16, 32, 62, 120, 228, 438, 840, 1608, 3086, 5918, 11346, 21754, 41712, 79980, 153358, 294052, 563822, ...

Their generating functions are further described in the appendix A.3.

From the cardinality represented by the series coefficients extracted above, we can draw the figure showing the convergences rate of capacity in comparison with the capacity evaluated in [4] as shown in figure 5.1. By these graph, we can see that the method work accurately and the results are reasonable due to the asymptotic that the convergence curves approach the theoretical rates.



(a) $(b, k) = (3, 2)$ **(b)** $(b, k) = (4, 3)$

(c) $(b, k) = (4, 4)$ **(d)** $(b, k) = (4, 5)$

Figure 5.1: Rate convergences for particular pairs of (b, k)

CHAPTER 6. CONCLUSIONS

6.1 Summary

To summary, this theses proposed a modern method to enumerate $m-$Dyck paths of arbitrary step height m in which limited total height is in consideration. This proposal can be used to model temperature changes as thermal-aware codes which can directly help control temperature in electronic devices, since peak temperature of devices should be seriously bounded to guarantee the overall system's performance.

In the research on locally-constrained de Bruijn sequences, we have evaluated the generating function and infer their coefficients to obtain exact number of codewords of a given length in both case of boundary $b = 3$ and $b = 4$. We have also successfully built an encoder which can be used to encode a codeword which is ready to be transferred and guaranteed to be recoverable in case of a number of sticky-insertions or deletions[4].

However, working with generating function in engineering field is not very ideal, as we still have to extract the coefficient of the function to obtain the exact cardinality and it takes $O(n)$ times (in which n is the length of the codewords that is being counted). Besides, enumerating the more general case of Dyck paths (i.e., $(a.b)$-Dyck paths), as well as the problems on enumerating the locally-constrained de Bruijn sequences for larger boundaries of b remain unsolved. They seems, unfortunately, out of reach of methodology proposed in this thesis.

6.2 Suggestions for Future Works

From the limitations above, we may try to enrich the contribution in this field by:

Simplify the Dyck path: As the formula to enumerate the $m-$Dyck path is quite complex, this might accidentally prevent us from discovering new and more generalized version of Dyck language in particular and thermal-aware codes in general. Thus, one of the noticeable direction is to enhance this enumeration and work with more abstracted Dyck path.

Enumerate (b, k)-locally-constrained de Bruijn sequence for larger boundary b: This case poses a huge challenge in computation complexity for method proposed in this study.

REFERENCE

[1] N. Xiang, "Digital sequences," *Handbook of Signal Processing in Acoustics*, **pages** 87–106, 2008.

[2] R. W. Hamming, "Error detecting and error correcting codes," *The Bell System Technical Journal*, **jourvol** 29, **number** 2, **pages** 147–160, 1950. DOI: 10.1002/j.1538-7305.1950.tb00463.x.

[3] J. Mosheiff, N. Resch, N. Ron-Zewi, S. Silas **and** M. Wootters, "Low-density parity-check codes achieve list-decoding capacity," *SIAM Journal on Computing*, **number** 0, FOCS20–38, 2021.

[4] Y. M. Chee, T. Etzion, H. M. Kiah **andothers**, "Locally-constrained de bruijn codes: Properties, enumeration, code constructions, and applications," *IEEE Transactions on Information Theory*, **jourvol** 67, **number** 12, **pages** 7857–7875, 2021. DOI: 10.1109/TIT.2021.3112300.

[5] K. Shigechi, *Rational dyck paths and decompositions*, 2021. DOI: 10.48550/ARXIV.2104.01877. **url**: https://arxiv.org/abs/2104.01877.

[6] Y. Fukukawa, "Counting generalized dyck paths," *arXiv: Combinatorics*, 2013.

[7] J. Huang, W. Sun, Z. Zhang, Z. Ling **and** X. Fang, "Thermal protection of electronic devices based on thermochemical energy storage," *Applied Thermal Engineering*, **jourvol** 186, **page** 116 507, 2021.

[8] A. Arreghini, M. Zahid, A Suhane, L Breuil, A Cacciato, J Van Houdt **andothers**, "Effect of high temperature annealing on tunnel oxide properties in tanos devices," *Microelectronic engineering*, **jourvol** 88, **number** 7, **pages** 1155–1158, 2011.

[9] C. Nadjahi, H. Louahlia **and** S. Lemasson, "A review of thermal management and innovative cooling strategies for data center," *Sustainable Computing: Informatics and Systems*, **jourvol** 19, **pages** 14–28, 2018.

[10] V. Midhun, S Suresh, B Praveen **and** C. R. Raj, "Effect of vacuum insulation panel on active thermal management for electronics system exposed to thermal radiation," *Thermal Science and Engineering Progress*, **jourvol** 26, **page** 101 117, 2021.

[11] W.-L. Hung, Y. Xie, N. ViJ'aykrishnan, M Kandemir **and** M. J. Irwin, "Thermal-aware task allocation and scheduling for embedded systems," **in***Design, Automation and Test in Europe* IEEE, 2005, **pages** 898–899.

[12] G. Baracchini, "Dyck paths and up-down walks," 2016.

[13] D. Coppersmith, R. C. Rhoades **and** J. M. VanderKam, *Counting de bruijn sequences as perturbations of linear recursions*, 2017. DOI: 10.48550/ARXIV.1705.07835. **url**: https://arxiv.org/abs/1705.07835.

[14] K. Shigechi, "Rational dyck paths and decompositions," *arXiv preprint arXiv:2104.01877*, 2021.

[15] I. P. Goulden **and** D. M. Jackson, "An inversion theorem for cluster decompositions of sequences with distinguished subsequences," *J. London Math. Soc*, **jourvol** 20, **pages** 567–576, 1979.

[16] E. Kupin **and** D. Yuster, "Generalizations of the goulden-jackson cluster method," *Journal of Difference Equations and Applications*, **jourvol** 16, **november** 2008. DOI: 10.1080/10236190902841976.

[17] J. Noonan **and** D. Zeilberger, "The goulden-jackson cluster method: Extensions, applications and implementations," *J. Differ. Equations Appl.*, **jourvol** 5, **july** 1998. DOI: 10.1080/10236199908808197.

[18] A. Bacher, "Generalized dyck paths of bounded height," *arXiv preprint arXiv:1303.2724*, 2013.

[19] P. H. S. Brian H. Marcus Ron M. Roth, *An Introduction to Coding for Constrained Systems*. 2001.

APPENDIX

A. DETAILED RESOLUTIONS AND RESULTS

A.1 Reduced forbidden words set simple generator

Given a boundary b and a window size k, the simple algorithm below generates a reduced set containing all patterns that violate locally-constrained de Bruijn sequence constraints.

Algorithm 1: Reduced forbidden-words set generator

```
Input  : b and k
Output : Reduced forbidden-words set B
:
forall length l from k+1 to b+k-1 do
    forall num from 0 to 2^l − 1 do
        bin ← binary(num) /* We need to force bin to be of length l */
        if bin[0:k-1] == bin[l-k:l-1] then
            sub ← False
            forall w ∈ B do
                if bin is sub-sequence of w then
                    sub ← True
                    break
            if sub is False then
                B.append(bin)
return B
```

A.2 Generating function coefficient extraction

Given a generating function $f(x) = \dfrac{P(x)}{Q(x)}$ in which $P(x)$ and $Q(x)$ are polynomials of variable x. The algorithm 2, based on Maclaurin coefficients expansion method, recursively extracts the coefficients of series represented by $f(x)$.

A.3 Some generating functions of particular (b,k)

Let $f_{b,k}(x) = \dfrac{P_{b,k}(x)}{Q_{b,k}(x)}$ denote the generating function of (b, k)-locally-constrained de Bruijn sequences. Then:

1. $f_{3,2} = \dfrac{P_{3,2}(x)}{Q_{3,2}(x)} = \dfrac{2x^4 + 3x^3 + 3x^2 + 2x + 1}{-x^3 - x^2 + 1}$
2. $f_{4,3} = \dfrac{P_{4,3}(x)}{Q_{4,3}(x)}$ in which:

   - $P_{4,3}(x) = 2x^{13} + 4x^{12} - x^{10} - 2x^9 - 2x^8 - 3x^7 - 8x^6 - 9x^5 - 7x^4 - 5x^3 - 3x^2 - 2x - 1$

Algorithm 2: Generating function coefficients extraction

Input : A generating function $f(x) = \dfrac{P(x)}{Q(x)}$, A positive integer n

Output : An extracted series $s = (s_1, s_2, s_3, ..., s_n)$ of $f(x)$

$P_0(x) \leftarrow P(x)$

forall *idx from 0 to n* **do**

$s_{idx} \leftarrow \dfrac{P_i(0)}{Q(0)}$

$P_{idx+1} \leftarrow \dfrac{P_i(x) - s_{idx} \times Q(x)}{x}$

return *The series* $s = (s_0, s_1, s_2, ..., s_n)$

- $Q_{4,3}(x) = x^{10} + x^7 + x^5 + x^4 + x^3 + x^2 - 1$

3. $f_{4,4} = \dfrac{P_{4,4}(x)}{Q_{4,4}(x)}$ in which:

- $P_{4,4}(x) = 2x^{18} + 6x^{17} + 6x^{16} + 3x^{15} + 2x^{14} + 5x^{13} + 6x^{12} + 2x^{11} - 5x^9 - 9x^8 - 10x^7 - 11x^6 - 9x^5 - 7x^4 - 5x^3 - 3x^2 - 2x - 1$
- $Q_{4,4}(x) = -x^{15} - 2x^{14} - x^{13} - x^9 - x^8 + 2x^7 + 3x^6 + 3x^5 + 3x^4 + x^3 + x^2 - 1$

4. $f_{4,5} = \dfrac{P_{4,5}(x)}{Q_{4,5}(x)}$ in which:

- $P_{4,5}(x) = 2x^{23} + 6x^{22} + 8x^{21} + 9x^{20} + 8x^{19} + 9x^{18} + 7x^{17} + 5x^{16} + 8x^{15} + 5x^{14} - 2x^{12} - 6x^{11} - 13x^{10} - 14x^9 - 14x^8 - 14x^7 - 11x^6 - 9x^5 - 7x^4 - 5x^3 - 3x^2 - 2x - 1$
- $Q_{4,5}(x) = -x^{20} - 2x^{19} - 3x^{18} - 3x^{17} - 3x^{16} - 2x^{15} - x^{14} + x^{10} + 2x^9 + 4x^8 + 4x^7 + 5x^6 + 5x^5 + 3x^4 + x^3 + x^2 - 1$

A.4 State splitting algorithm on (3,2)-locally-constrained de Bruijn encoder

The procedure to split states on the $(3, 2)$-locally-constrained de Bruijn graph to build a $(3, 2)$-locally-constrained de Bruijn encoder is demonstrated in adjacency matrices corresponding to adjacency table below.

1. The original adjacency matrix, $\mathscr{G}^5$, is shown in table A.1. The out-degree of each vertex is $(4, 5, 5, 4, 3, 3)$, the approximate eigence-vector is $(3, 4, 4, 3, 2, 2)$.

2. After splitting the state f into two vertices, we get the matrix shown in table A.2. The out-degree of each vertex is $(4, 5, 6, 5, 4, 1, 2)$, the approximate eigence-vector is $(3, 4, 4, 3, 2, 1, 1)$.

3. After splitting the state e into two vertices, we get the matrix shown in table

	a	b	c	d	e	f
a	0	1	0	2	1	0
b	1	0	2	1	1	0
c	1	2	0	1	0	1
d	2	0	1	0	0	1
e	1	1	0	0	0	1
f	0	0	1	1	1	0

Table A.1: Original adjacency matrix $\mathscr{G}^5$

	a	b	c	d	e	f1	f2
a	0	1	0	2	1	0	0
b	1	0	2	1	1	0	0
c	1	2	0	1	0	**1**	**1**
d	2	0	1	0	0	**1**	**1**
e	1	1	0	0	0	**1**	**1**
f1	0	0	**1**	0	0	0	0
f2	0	0	0	**1**	**1**	0	0

Table A.2: Adjacency matrix of $\mathscr{G}^5$ after state f is split

A.3. The out-degree of each vertex is $(5, 6, 6, 5, 1, 3, 1, 3)$, the approximate eigence-vector is $(3, 4, 4, 3, 1, 1, 1, 1)$.

	a	b	c	d	e1	e2	f1	f2
a	0	1	0	2	**1**	**1**	0	0
b	1	0	2	1	**1**	**1**	0	0
c	1	2	0	1	0	0	1	1
d	2	0	1	0	0	0	1	1
e1	0	**1**	0	0	0	0	0	0
e2	**1**	0	0	0	0	0	**1**	**1**
f1	0	0	1	0	0	0	0	0
f2	0	0	0	1	**1**	**1**	0	0

Table A.3: Adjacency matrix of $\mathscr{G}^5$ after state e is split

4. After splitting the state d into three vertices, we get the matrix shown in table A.4. The out-degree of each vertex is $(9, 8, 8, 1, 2, 2, 1, 3, 1, 5)$, the approximate eigence-vector is $(3, 4, 4, 1, 1, 1, 1, 1, 1, 1)$.

5. After splitting the state c into four vertices, we get the matrix shown in table A.5. The out-degree of each vertex is $(9, 14, 1, 1, 4, 2, 1, 2, 2, 1, 3, 1, 5)$, the approximate eigence-vector is $(3, 4, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1)$.

6. After splitting the state b into four vertices, we get the matrix shown in table A.6. The out-degree of each vertex is $(12, 4, 4, 2, 4, 4, 4, 4, 2, 4, 2, 2, 4, 3, 4, 5)$, the

	a	b	c	d1	d2	d3	e1	e2	f1	f2
a	0	1	0	**2**	**2**	**2**	1	1	0	0
b	1	0	2	**1**	**1**	**1**	1	1	0	0
c	1	2	0	**1**	**1**	**1**	0	0	1	1
d1	0	0	**1**	0	0	0	0	0	0	0
d2	**1**	0	0	0	0	0	0	0	**1**	0
d3	**1**	0	0	0	0	0	0	0	0	**1**
e1	0	1	0	0	0	0	0	0	0	0
e2	1	0	0	0	0	0	0	0	1	1
f1	0	0	1	0	0	0	0	0	0	0
f2	0	0	0	**1**	**1**	**1**	1	1	0	0

Table A.4: Adjacency matrix of $\mathscr{G}^5$ after state d is split

	a	b	c1	c2	c3	c4	d1	d2	d3	e1	e2	f1	f2
a	0	1	0	0	0	0	2	2	2	1	1	0	0
b	1	0	**2**	**2**	**2**	**2**	1	1	1	1	1	0	0
c1	0	**1**	0	0	0	0	0	0	0	0	0	0	0
c2	0	**1**	0	0	0	0	0	0	0	0	0	0	0
c3	0	0	0	0	0	0	**1**	**1**	**1**	0	0	**1**	0
c4	**1**	0	0	0	0	0	0	0	0	0	0	0	**1**
d1	0	0	**1**	**1**	**1**	**1**	0	0	0	0	0	0	0
d2	1	0	0	0	0	0	0	0	0	0	0	1	0
d3	1	0	0	0	0	0	0	0	0	0	0	0	1
e1	0	1	0	0	0	0	0	0	0	0	0	0	0
e2	1	0	0	0	0	0	0	0	0	0	0	1	1
f1	0	0	**1**	**1**	**1**	**1**	0	0	0	0	0	0	0
f2	0	0	0	0	0	0	1	1	1	1	1	0	0

Table A.5: Adjacency matrix of $\mathscr{G}^5$ after state c is split

approximate eigence-vector is $(3, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1)$.

7. After splitting the state a into three vertices, we get the matrix shown in table A.7. The out-degree of each vertex is $(4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 5, 4, 5)$, the approximate eigence-vector is $(1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1)$.

The graph now satisfies the constraints that out-degree of each vertex is at least 4, which is just enough to be tagged to be an encoder. Apply the tagging algorithm3 to tag edges of the adjacency matrix, and relabel vertices a, b, c, d, e, f to $00, 01, 10, 11, 010, 101$, and $1, 2, 3, 4$ to a, b, c, d, we have a completed encoder as demonstrated in figure A.1.

	a	b1	b2	b3	b4	c1	c2	c3	c4	d1	d2	d3	e1	e2	f1	f2
a	0	**1**	**1**	**1**	**1**	0	0	0	0	2	2	2	1	1	0	0
b1	0	0	0	0	0	**2**	**2**	0	0	0	0	0	0	0	0	0
b2	0	0	0	0	0	0	0	**2**	**2**	0	0	0	0	0	0	0
b3	**1**	0	0	0	0	0	0	0	0	**1**	0	0	0	0	0	0
b4	0	0	0	0	0	0	0	0	0	0	**1**	**1**	**1**	**1**	0	0
c1	0	**1**	**1**	**1**	**1**	0	0	0	0	0	0	0	0	0	0	0
c2	0	**1**	**1**	**1**	**1**	0	0	0	0	0	0	0	0	0	0	0
c3	0	0	0	0	0	0	0	0	0	1	1	1	0	0	1	0
c4	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
d1	0	0	0	0	0	1	1	1	1	0	0	0	0	0	0	0
d2	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
d3	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
e1	0	**1**	**1**	**1**	**1**	0	0	0	0	0	0	0	0	0	0	0
e2	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1
f1	0	0	0	0	0	1	1	1	1	0	0	0	0	0	0	0
f2	0	0	0	0	0	0	0	0	0	1	1	1	1	1	0	0

Table A.6: Adjacency matrix of $\mathscr{G}^5$ after state b is split

	a1	a2	a3	b1	b2	b3	b4	c1	c2	c3	c4	d1	d2	d3	e1	e2	f1	f2
a1	0	0	0	**1**	**1**	**1**	**1**	0	0	0	0	0	0	0	0	0	0	0
a2	0	0	0	0	0	0	0	0	0	0	0	**2**	**2**	0	0	0	0	0
a3	0	0	0	0	0	0	0	0	0	0	0	0	0	**2**	**1**	**1**	0	0
b1	0	0	0	0	0	0	0	2	2	0	0	0	0	0	0	0	0	0
b2	0	0	0	0	0	0	0	0	0	2	2	0	0	0	0	0	0	0
b3	**1**	**1**	**1**	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
b4	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	0	0
c1	0	0	0	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0
c2	0	0	0	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0
c3	0	0	0	0	0	0	0	0	0	0	0	1	1	1	0	0	1	0
c4	**1**	**1**	**1**	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
d1	0	0	0	0	0	0	0	1	1	1	1	0	0	0	0	0	0	0
d2	**1**	**1**	**1**	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
d3	**1**	**1**	**1**	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
e1	0	0	0	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0
e2	**1**	**1**	**1**	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1
f1	0	0	0	0	0	0	0	1	1	1	1	0	0	0	0	0	0	0
f2	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	0	0

Table A.7: Adjacency matrix of $\mathscr{G}^5$ after state a is split

Algorithm 3: Tagging(A_G, p)

Input : A_G : a labeled graph that needs tagging;
p : size of block used to tag

Output Tagged graph G

:

forall *state* v *in* A_G **do**
    $count \leftarrow 0$
    forall *state* u *in* A_G **do**
        /* Do we need to tag all out-going edges of $A_G[v, u]$? */
        if $count + A_G[v, u] < 2^p$ **then**
            $count \leftarrow count + A_G[v, u]$
        else
            /* No, only some of them are needed */
            $A_G[v, u] \leftarrow 2^p - count$
            $count \leftarrow 2^p$
        tag($A_G[v, u]$)

return A_G

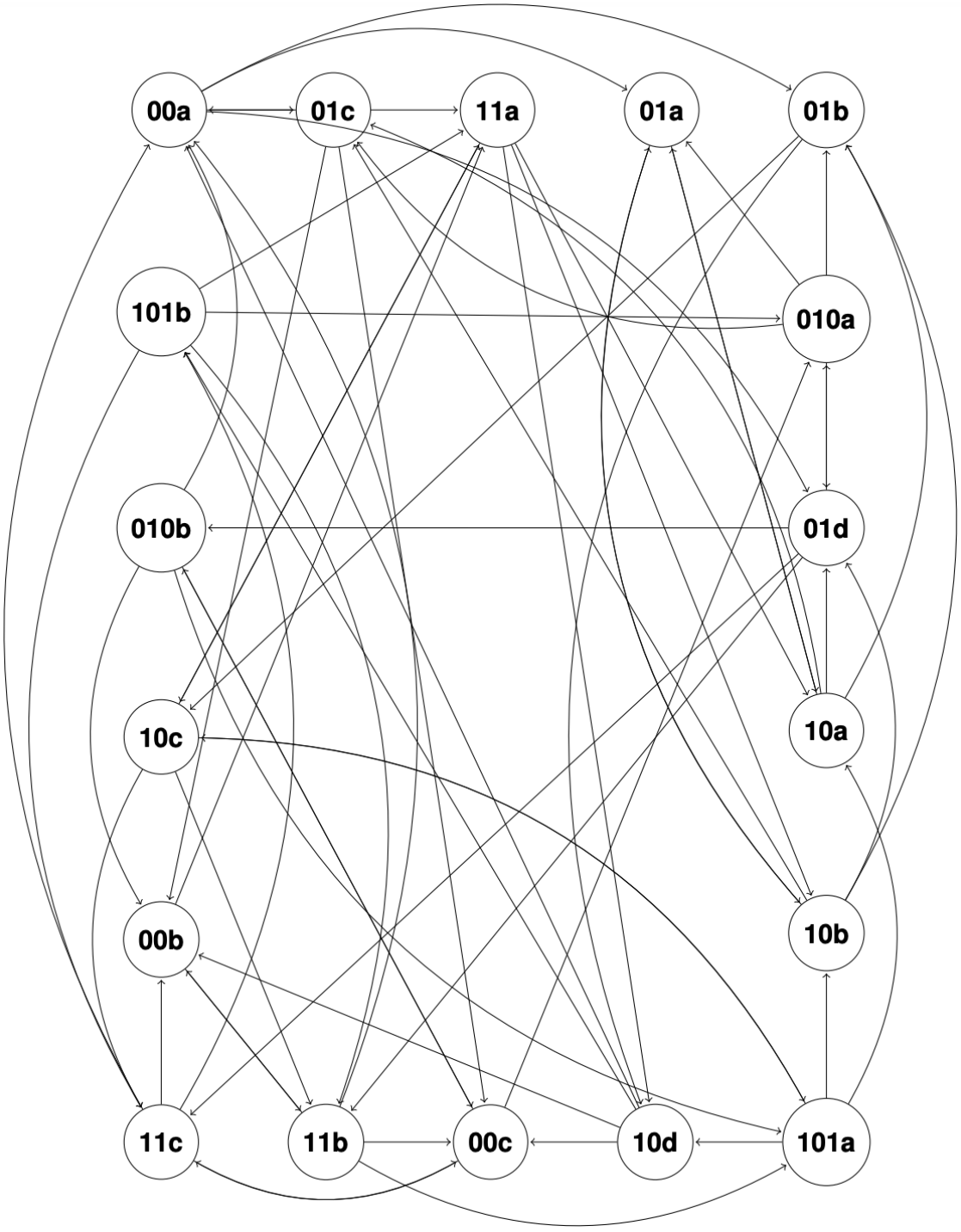


Figure A.1: $(3, 2)$-locally-constrained de Bruijn encoder