Goulden-Jackson cluster generating function method for locally-constrained de Bruijn code

Van Khu Vu, Tien Long Nguyen, Van Luc Nguyen, Phan Thuan Do
...email...

ABSTRACT

After Goulden and Jackson's introduction about cluster generating function[1], the method has been gradually applied in counting many particular types of combinatorics problem. Noonan and Zeilberger generalized and implemented the method to specific problems of counting words with forbidden prescribed words[2]. Chao-Jen Wang used the method to count Dyck paths by occurrences of subwords[3]. In this work, we will give a general method to count the locally-constrained de Bruijn sequences using cluster method. This method can be used in every boundary b and windows size k. We also give some basic concepts of state-splitting algorithm and apply it in locally-constrained de Bruijn code to construct an encoder of a prescribed asymptotic rate. In general, this method is very useful for encoding and decoding sequences in constrained systems based on forbidden patterns.

I. INTRODUCTION

This work gives a general method to enumerate locally-constrained de Bruijn sequences of a given length using cluster method. The enumeration of locally-constrained de Bruijn sequences is also mentioned in [4]. However, they are only able to construct a state diagram for the $(3,3)$-locally-constrained de Bruijn sequences and propose a method to find the cardinality of the code for a particular case of a boundary, $b = 3$. This limitation suggested us to apply Goulden-Jackson cluster method to enumerate the sequences for a larger boundary of b, $b = 4$, and even larger.

In the section II, we will mention some concepts of locally-constrained de Bruijn sequence and its desire to find its cardinality of words. We also explain and influence the Goulden-Jackson cluster method which is very useful to count these kinds of combinatorial problems. The details of matching problem and finding the generating function counting the locally-constrained de Bruijn code will be discussed deeply in section III.

The paper closes with some opening research directions related to the topic which can be refined and developed in the near future.

II. PRELIMINARIES

A. Reduced forbidden-words set

Working with constrained systems, we usually have to consider sets containing prohibited patterns. Such sets are called forbidden-words sets. We also notice that for a forbidden-words set containing two patterns u and v such that u is a sub-sequence of v, prohibiting u also means to prohibit v. So we can remove v without losing properties and constraints of the system. Therefore, for simplicity and to avoid duplication, we are interested in forbidden-words sets that do not contain any two forbidden words in which one is a sub-sequence of the other. In this paper, we will call such sets as *reduced forbidden-words set*.

B. Locally-constrained de Bruijn sequence and locally-constrained de Bruijn code

As defined in [4], an (n, b, k) locally-constrained de Bruijn sequence is a sequence of length n whose each sub-string of size k appears exactly once around its $b-1$ neighbors. In other words, a sequence $s = (s_1, s_2, ..., s_n)$ is called an (n, b, k) locally-constrained de Bruijn sequence if and only if:

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

where $s[i, j] = (s_i, s_{i+1}, ..., s_j)$ denotes the sub-sequence of s, from the index i to index j.

In this paper, we are interested in the enumeration of the (n, b, k) locally-constrained de Bruijn sequences for $b = 3$, $b = 4$, and open method to work with larger boundaries of b.

C. Goulden Jackson cluster generating function method

In this section, we present the foundation of Goulden-Jackson cluster method that we used to enumerate the $(n, 3, k)$, $(n, 4, k)$ locally-constrained de Bruijn sequences. The first concept that needs to be mentioned is the definition of q-clusters [1].

A q-cluster $(q \geq 1)$ on the alphabet $\mathcal{N}$, and associated with the *reduced set* $\mathscr{A}$, is a triple $(\sigma_1\sigma_2...\sigma_r,\ A_{i_1}, A_{i_2}...A_{i_q},\ (l_1, l_2, ..., l_q))$ such that $\sigma_1, \sigma_2, ..., \sigma_r \in \mathcal{N}^+$ and $(l_1, l_2, ..., l_q)$ satisfies the following conditions where 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 first part of the triple is the sequence of interest. The second part, $A_{i_1}, A_{i_2}, ..., A_{i_q}$, is the arrangement of elements of the reduced set $\mathscr{A}$ in the first part. The last part, $(l_1, l_2, ..., l_q)$, simply denotes the starting point of each element in the set $\mathscr{A}$. In order for the elements of the reduced set $\mathscr{A}$ to overlap, the second condition must hold. In the coming parts and examples, the last part is usually ignored in representation for simplicity.

Example 1. *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.* $\{ABCDCBAD, \{BC, BCDC, CBAD\}\}$ *is not a cluster, either, because the first letter of ABCDCBAD, A, does not appear as the first letter of any words of the reduced set* $\{BC, BCDC, CBAD\}$.

We could also imagine the cluster as a chain in which words in the reduced set play a role as rings connected together with some overlaps in between.

Let $\mathcal{C}$ denote all clusters that can be generated from the forbidden-words set $\mathcal{B}$, the generating function enumerating the words that forbid words in $\mathcal{B}$, (i.e., forbid all clusters in $\mathcal{C}$), by I. P. Goulden and D. M. Jackson [1], and Noonan and Zeilberger [2] is:

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

Noonan and Zeilberger [2] also established a general linear equation that can be solved to find the generating function representing the enumeration of combinatorial problems with prescribed forbidden-words set. Specifically, given a forbidden-words set $\mathcal{B}$, the equation can be written as:

$$\text{weight}(\mathcal{C}[v]) = -t^{|v|} - \sum_{u \in \mathcal{B}} \left(\text{weight}(\mathcal{C}[u]) \cdot \sum_{r \in O(v,u)} t^{|v \backslash r|} \right) \tag{2}$$

in which u and v are elements of $\mathcal{B}$, and $\mathcal{C}[v]$ is the cluster that starts with $v \in \mathcal{B}$, $O(v, u)$ denotes the overlap of the end of v on the beginning of u, and $v \backslash r$ denotes the removal of the path r from the end of v.

The equation (2) is capable of manual calculation because u and v are used interchangeably. This 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 (1), we obtain the generating function of the expected number of words from alphabet $\mathcal{A}$ given a forbidden-words set $\mathcal{B}$.

III. Main results

In this section, we show how we solve the problem of enumerating locally-constrained de Bruijn sequences of a given length with concepts in Goulden-Jackson method in finding cluster generating function. Then we infer the cardinality and the rate of the code for the boundary $b = 4$ and an arbitrary number of k.

A. $(n, 3, 2)$ *and* $(n, 4, k)$ *locally-constrained de Bruijn sequences enumeration*

1) Reduced forbidden-words set $\mathcal{B}$ *:* A forbidden-words set $\mathcal{B}$, by our expected behavior, 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 $s = (s_1, s_2, ..., s_n)$ 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:`

$$w[1, k] = w[l-k+1, l] \tag{3}$$

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:`

$$k+1 \leq |w| \leq b+k-1 \tag{4}$$

From these two observations, 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:

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

2) Application: 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.

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), we establish four equations as below:

$$\text{weight}(\mathcal{C}[000]) = -t^3 - t^2.[\text{weight}(\mathcal{C}[000]) + \text{weight}(\mathcal{C}[0101])] - t.\text{weight}(\mathcal{C}[000]) \tag{5}$$

$$\text{weight}(\mathcal{C}[111]) = -t^3 - t^2.[\text{weight}(\mathcal{C}[111]) + \text{weight}(\mathcal{C}[1010])] - t.\text{weight}(\mathcal{C}[111]) \tag{6}$$

$$\text{weight}(\mathcal{C}[0101]) = -t^4 - t^3[\text{weight}(\mathcal{C}[1010]) + \text{weight}(\mathcal{C}[111])] - t^2\text{weight}(\mathcal{C}[0101]) - t.\text{weight}(\mathcal{C}[1010]) \tag{7}$$

$$\text{weight}(\mathcal{C}[1010]) = -t^4 - t^3[\text{weight}(\mathcal{C}[0101]) + \text{weight}(\mathcal{C}[000])] - t^2\text{weight}(\mathcal{C}[1010]) - t.\text{weight}(\mathcal{C}[0101]) \quad (8)$$

We might notice that the equations (5) and (6) 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 (1), with $\mathcal{A} = \{0, 1\}$, we obtain the generating function of cardinality of $(n, 3, 2)$ locally-constrained de Bruijn code:

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

■

Using the same approach, we can try 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:

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

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:

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

Solving this equations system yields the final generating function:

$$f(t) = \frac{1}{1 - |\mathcal{A}|t - \text{weight}(\mathcal{C})} = \frac{A}{B}$$

where

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

■

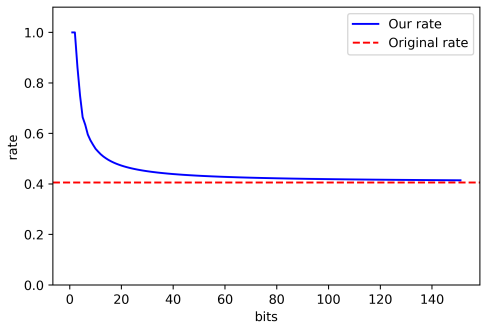
After obtaining the desired generating functions, we infer their corresponding coefficients representing the cardinalities of the de Bruijn code with word length increased from 1. In comparison with the rates previously calculated by the Perron-Frobenius theory from the group of authors in [4], we show the convergences of rates for each particular pair of (b, k) as illustrated in figure 1. These rates tend to converge faster at a higher value as the windows size k increases.

Extra result:

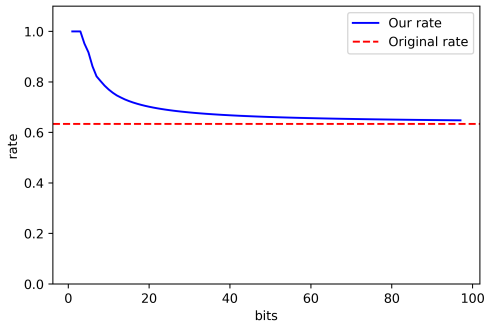
k	cardinality	rate
6	403560155003542492	0.974759
7	703913534002751264	0.988136
8	910372583183140946	0.994321
9	1028451404556522096	0.997253
10	1090597443018931520	0.998664
11	1122079208358220872	0.999348
12	1137745386094987008	0.999681
13	1145475550444045968	0.999844
14	1149274042706606608	0.999924
15	1151136540558651392	0.999963
16	1152048632843525008	0.999982
17	1152494924491259872	0.999991
18	1152713153656908194	0.999996
19	1152819801154518910	0.999998
20	1152871889430102016	0.999999

TABLE I: Rate comparison for the case of $b = 4$ at $n = 60$ bits for particular windows size k, result in the file /Files/b4n60.txt/

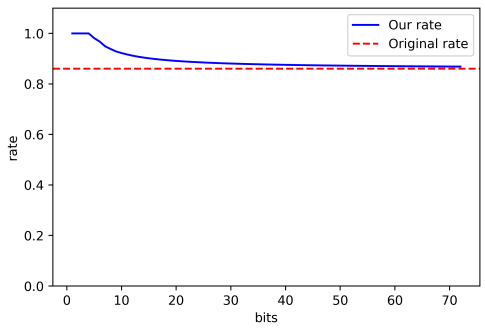
One might notice that the importance of this method is to solve the system of equations whose number of variables (half of the size of the forbidden-words set $\mathcal{B}$) increases significantly as b increases. For $b = 4$, we have to solve the system of 5 equations. For $b = 5$, in fact, we have to solve the system of 11 equations which takes us hours, compared to seconds in



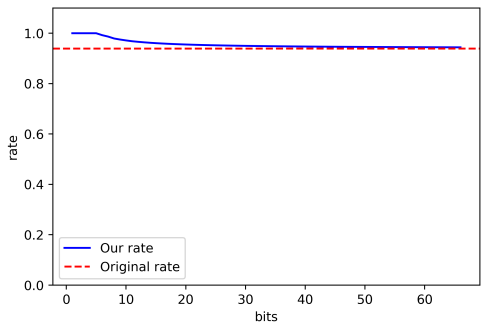
(a) $(b, k) = (3, 2)$



(b) $(b, k) = (4, 3)$



(c) $(b, k) = (4, 4)$



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

Fig. 1: Rate convergences for particular pairs of (b, k)

the case of $b = 4$. 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 1. *For any windows size k with boundary $b = 4$, the forbidden-words set $\mathcal{B}$ contains 10 words.*

Proof. Recall from (3) 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, there are always 10 forbidden-words given $b = 4$. □

From the demonstration of $b = 3$ and $b = 4$ given above, we are also interested in larger boundary of b. One might wonder whether the equations system obtained from (2) always has a unique solution. In this part, we will prove that this assumption is true.

Theorem 2. *The equations system obtained from the formula (2), 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. $\curlyvee(w)$ 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 $\curlyvee(w)$ 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 $\curlyvee(\mathcal{B}') = \{\curlyvee(w) | w \in \mathcal{B}'\}$. In other words, we split the set $\mathcal{B}$ into two equal and disjoint subsets, $\mathcal{B}'$ and $\curlyvee(\mathcal{B}')$, such that each element in $\curlyvee(\mathcal{B}')$ can be generated from one and only one corresponding element in $\mathcal{B}'$ by the $\curlyvee(w)$ operation.

Because of the symmetry between w_i and $\curlyvee(w_i)$, we always have weight$(\mathcal{C}[w_i])$ = weight$(\mathcal{C}[\curlyvee(w_i)]) = \mathscr{W}_i$ $(1 \le i \le n)$. Therefore, from (2), 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) is now re-written as following:

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

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

$$\Leftrightarrow \mathscr{W}_i + \sum_{w_j \in \mathcal{B}'} \left[\mathscr{W}_j \cdot \left(\sum_{r \in O(w_i, w_j)} t^{|w_i \backslash r|} + \sum_{r \in O(w_i, \curlyvee(w_j))} t^{|w_i \backslash r|} \right) \right] = -t^{|w_i|} \quad (9)$$

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

Let $\mathscr{c}_{ij}$ 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 $\curlyvee(w_j)$ (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 (9) non-zero. Let $\gamma(\mathscr{c}_{ij})$ be the minimum order of t in $\mathscr{c}_{ij}$ whose coefficient is non-zero, and $[t^k](\mathscr{c}_{ij})$ be the coefficient of t^k in the expression of $\mathscr{c}_{ij}$. As a result, $[t^{|w_i|-1}](\mathscr{c}_{ij}) = 1 \Rightarrow \gamma(\mathscr{c}_{ij}) \le |w_i| - 1$.

In the equation (9) 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 $r \Rightarrow |w_i \backslash r| \ge 1$ $\forall r \in O(w_i, w_j)$. Hence, $\gamma(\mathscr{c}_{ij}) \ge 1$ $\forall i \ne j \Rightarrow [t^0](\mathscr{c}_{ij}) = 0$ $\forall i \ne j$. Besides, as the coefficient of $\mathscr{W}_i$ at the first term is 1, $[t^0](\mathscr{c}_{ii}) = 1$, meaning that $\gamma(\mathscr{c}_{ii}) = 0$ $\forall i$.

Denote $A = [\mathscr{c}_{ij}]$ 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:

$$\det A = \sum_{j=1}^{n} (-1)^{i+j} \mathscr{c}_{ij} \times \det A_{ij} \quad (10)$$

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,

$$\gamma(\det A_{ij}) \ge 0$$
$$\Rightarrow \gamma(\mathscr{c}_{ij} \times \det A_{ij}) \ge 1 \ \forall i \ne j$$
$$\Rightarrow [t^0](\mathscr{c}_{ij} \times \det A_{ij}) = 0 \ \forall i \ne j$$

Consider the power of zero of t in both sides of (10), as $[t^0](\mathscr{c}_{ii}) = 1$, we have:

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

We can see a recursive relation that:

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

Therefore, $\det A \not\equiv 0$. This means that the equations system obtained from (2) always has a unique solution. □

Theorem 2 shows that this method can be used to find generating function counting the cardinality of (n, b, k) locally-constrained de Bruijn code for any particular boundary b. However, due to the computation complexity of solving the system of linear equations, we are currently able to solve for $b = 4$, and k is limited by roundly 25 as the limit of datatype in C++. The source code is available at github.com/nguyenluc99/GJ_LC_deBruijnCode.

B. Finite State Splitting Algorithm

Let S be a constrained system. In this section, we give the pseudo code of State Splitting Algorithm, a very useful algorithm used to find a finite state encoder of constrained systems.

1) Theory: In real application, we need to find an approximation of the rate that is feasible for calculation and application, i.e., in the form of a ratio $\frac{p}{q}$ where p and q are two whole numbers. This ratio is more practical because they can be treated as discrete quantities which is very easy in circuit and graph design, implementation.

To do so, we introduce a procedure to find the ratio, $\frac{p}{q}$, such that $0 \le cap(S) - \frac{p}{q} < \epsilon$ in pseudo of algorithm 1. In general, there exists a trade-off between the asymptotic of $\frac{p}{q}$ to $cap(S)$

and the complexity of the algorithm. Specifically, the closer $\frac{p}{q}$ to $cap(S)$ is, the larger p and q becomes, and the longer the algorithm executes. In order to control the complexity, the variable ϵ mentioned above is used to tune the difference between $\frac{p}{q}$ and $cap(S)$. If we need p and q to be relatively small, ϵ can be adjusted to be a bit larger.

Algorithm 1: ApproximateRate(cap, ϵ)

Input : cap : the capacity of S,
ϵ : the precision of the procedure
Output: (p, q) such that $0 \leq cap(S) - \frac{p}{q} < \epsilon$

$p \leftarrow 1$
$q \leftarrow 1$
$l_1 \leftarrow \left(cap < \frac{p}{q}\right)$ `// l1 is a logic variable implying if` $cap < \frac{p}{q}$
while $l_1 = true$ **do**
  while $l_1 = true$ **do**
    $q \leftarrow q + 1$
    $l_1 \leftarrow \left(cap < \frac{p}{q}\right)$
  end while
  $l_2 \leftarrow \left(cap - \frac{p}{q} > \epsilon\right)$ `// l2 is a logic variable implying if` $cap - \frac{p}{q} > \epsilon$
  while $l_1 = false$ *and* $l_2 = true$ **do**
    $p \leftarrow p + 1$
    $l_1 \leftarrow \left(cap < \frac{p}{q}\right)$
    $l_2 \leftarrow \left(cap - \frac{p}{q} > \epsilon\right)$
  end while
end while
return (p, q)

We also give the pseudo code to eliminate redundant edges and tag the remaining edges for a labeled graph G by p-block in algorithm 2.

We are now in the page of State Splitting Algorithm discussion whose pseudo code is shown in algorithm 3.

2) Finite state encoder for $(3, 2)$ *locally-constrained de Bruijn code:* Let S be the $(3, 2)$ locally-constrained de Bruijn constrained system. The labeled graph $\mathcal{G}$ representing S is shown in figure 2. 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 $\mathcal{G}^5$ is given in figure 3. 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 $\delta(\mathcal{G})$ denote the minimum out-degree of $\mathcal{G}$. At the moment, $\delta(\mathcal{G}) = 3$, 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

Algorithm 2: 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 */`
      $A_G[v, u] \leftarrow 2^p - count$
      $count \leftarrow 2^p$
    end if
    tag$(A_G[v, u])$
  end forall
end forall
return A_G

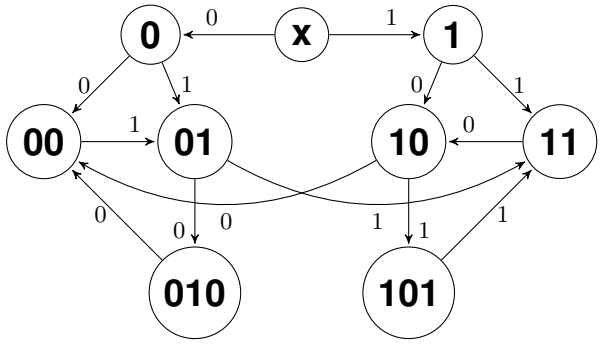


Fig. 2: State diagram for the (3, 2)-locally-constrained de Bruijn sequences

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, ***This graph is big enough to be excluded on this paper. Do you think I should include it here?*** we obtain a labeled graph $\mathcal{G}'$ satisfying that $\delta(\mathcal{G}'\} = 4$, which is just enough to be an finite state encoder with rate $2 : 5$.

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

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

Fig. 3: Adjacency matrix of $\mathcal{G}^5$

Algorithm 3: State-Splitting $(A_{S(G)}, \epsilon)$

Input : $A_{S(G)}$: Adjacency matrix of labeled graph G, representing a constrained system S;
ϵ: the precision of this algorithm

Output: Finite-state encoder for S

/* Calculate S's capacity and choose $p,\ q$ such that $0 \leq cap - \frac{p}{q} \leq \epsilon$ */

$A_G \leftarrow$determinizing$(A_{S(G)})$

$cap \leftarrow \log(\alpha(A_G))$

$p, q \leftarrow$ApproximateRate(cap, ϵ)

Construct G^q

$\mathbf{x} \leftarrow$Franaszek$(A_G^q, 2^p)$

// restrict $\mathbf{x}$

$\mathbf{x},\ A_G \leftarrow$eliminate$(\mathbf{x}, A_G)$/* eliminate all 0 components of $\mathbf{x}$ and their corresponding components in A_G */

$A_H \leftarrow$irr_sink(A_G)/* determine the irreducible sinks of eliminated A_G */

$\mathbf{x} \leftarrow$restrict$(A_H, \mathbf{x})$// restrict $\mathbf{x}$ to be indexed by states of A_H

while *min_degree*$(A_H) < 2^p$ **do**

  /* split H by a non_trivial $\mathbf{x}$_consistent partition of the edges, resulting a labeled graph H' and approximate eigenvector $\mathbf{x}'$ */

  $A_H, \mathbf{x} \leftarrow$ non_trivial_$\mathbf{x}$_consistent$(A_H, \mathbf{x})$

end while

/* tag 2^p out-going edges at each state with binary p-block and remove other edges */

$A_H \leftarrow$Tagging(A_H, p)

return A_H

encoder is represented in table II.

Example 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* $1001101100110110110110010$*.*

IV. Conclusion

We have introduced the application of Goulden-Jackson cluster generating function method to count the number of sequences in locally-constrained de Bruijn code for a larger boundary $b = 4$. This improvement not only opens up new opportunities for us to count for even larger boundaries of b, but also contributes to the great capability of the Goulden-Jackson cluster method in enumerating many kinds of patterns-forbidden combinatorial problems, which is very frequently considered in constrained systems.

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

References

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

[2] J. Noonan and D. Zeilberger, "The goulden-jackson cluster method: Extensions, applications and implementations," 1998. [Online]. Available: https://arxiv.org/abs/math/9806036

[3] C. Wang, *Applications of the Goulden-Jackson Cluster Method to Counting Dyck Paths by Occurrences of Subwords*. Brandeis University, 2011. [Online]. Available: https://books.google.com.vn/books?id=asRQMwEACAAJ

[4] Y. M. Chee, T. Etzion, H. M. Kiah, S. Marcovich, A. Vardy, V. K. Vu, and E. Yaakobi, "Locally-constrained de Bruijn codes: Properties, enumeration, code constructions, and applications," *IEEE Transactions on Information Theory*, vol. PP, pp. 1–1, 09 2021.

[5] E. Kupin and D. Yuster, "Generalizations of the goulden-jackson cluster method," *Journal of Difference Equations and Applications*, vol. 16, 11 2008.

[6] Y. M. Chee, J. Chrisnata, H. M. Kiah, and T. T. Nguyen, "Efficient encoding/decoding of irreducible words for codes correcting tandem duplications," 2018. [Online]. Available: https://arxiv.org/abs/1801.02310