Repeat-Free Codes

Ohad Elishco*, Ryan Gabrys†, Muriel Médard‡, Eitan Yaakobi§

* Massachusetts Institute of Technology, MA, 02139
† Spawar Systems Center, San Diego, CA, 92115
‡ Department of Computer Science, Technion – Israel Institute of Technology, Haifa, Israel, 3200009
Emails: {ohadeli,medard}@mit.edu, ryan.gabrys@navy.mil, yaakobi@cs.technion.ac.il

Abstract—In this paper we consider the problem of encoding data into repeat-free sequences in which sequences are imposed to contain any k-tuple at most once (for predefined k). First, the capacity and redundancy of the repeat-free constraint are calculated. Then, an efficient algorithm, which uses a single bit of redundancy, is presented to encode length-n sequences for \( k = 2 + 2 \log n \). This algorithm is then improved to support any value of k of the form \( k = a \log n \), for \( 1 < a < 2 \), while its redundancy is \( o(n) \). Lastly, we also calculate the capacity of this constraint when combined with local constraints which are given by a constrained system.

I. INTRODUCTION

Repeat-free sequences represent a generalization of the well-known De-Bruijn sequences where every length-k substring appears exactly once. De-Bruijn sequences have found applications in areas as diverse as cryptography, pseudorandomness, and information hiding in wireless communications [6]. However, one potential drawback to adopting De-Bruijn sequences for representing information is that De-Bruijn sequences have rate at most \( \frac{1}{2} \). In this work, we show that by relaxing the condition that every k-tuple appears exactly once, we can generate rate-1 code with efficient encoders/decoders for a variety of parameters.

One motivating application for this work is in DNA storage, and, in particular, the reading process of a DNA string. The reading process of a DNA string is as follows. At first, the long string is fragmented into substrings of a shorter length which may be read properly. Then, a multi-set of all the short strings is obtained in a form of their frequency. The long DNA string should then be reconstructed using only the knowledge of shorter length substrings.

There are two common lines of work on coding for DNA storage systems. The first assumes that the data is stored in a living organism. In this case, the major concern is to correct errors which are made by naturally occurring mutations. For analysis of the capacity of mutation strings see [5], [14] and the references therein. For coding algorithms related papers see, for example, [8], [11], [17].

The second line of work focuses on data storage outside a living organism and is called coding for string reconstruction. The goal of coding for the string reconstruction problem is to encode arbitrary strings into ones that are uniquely reconstructible. This problem is motivated by the reading process of DNA-based data storage, where the stored strings are to be reconstructed from information about substring appearing in the stored string. This problem motivated a series of papers regarding decoding of sequences from partial information on their substrings [1], [4], [7], [15].

In order to ensure unique reconstruction, studies were made on reconstruction of encoded sequences [3], [9], [12]. One method that may guarantee a unique reconstruction, is encoding the information sequence to a codeword which does not contain any k-tuple more than once. For two positive integers \( k < n \), we say that an length-n word \( w \) is k-repeat free if every subword of \( w \) of length \( k \) appears at most once. It is already known that k-repeat free words are uniquely reconstructible from their r-length substrings multi-set if \( r \geq k + 1 \) [16], and an encoding scheme that exploits this property has been recently proposed in [7]; however, the encoded words are not strictly repeat free. Thus, studying the repeat-free constraint and designing respective efficient encoding-decoding schemes is still an open research problem, which is the main topic of this paper.

Another important characteristic of the k-repeat free sequences is the number of sequences that exist as a function of the length of the sequence. Arguably, one of the most well-known family of k-repeat free sequences are De-Bruijn sequences of span \( k \) which play an important role in this paper. A De-Bruijn sequence of span \( k \) is a sequence over a finite alphabet, in which every k-tuple appears exactly once. Every De-Bruijn sequence of span \( k \) over an alphabet of size \( q \) is k-repeat free [2]. A close formula for the number of De-Bruijn sequences of length \( q^k + k - 1 \) exists [2], [6]. Unfortunately, there is no such formula for the general set of k-repeat free sequences. It is clear that a sequence of length \( n \) over an alphabet of size \( q \) cannot be k-repeat free if \( k < \log_q(n + k - 1) \), but even the size of k-repeat free sequences with \( k = a \log_q n \) with \( a > 1 \) has not been fully determined.

Using simple union bound arguments it is straightforward to show that in the binary case, the rate of the number of k-repeat free sequences is 1 when \( k = [a \log_2 n] \) and \( a \geq 2 \) (i.e., the redundancy is a constant). From the known enumeration results of De-Bruijn sequences [2], [6] it follows that the rate is at least 1/2 for \( a = 1 \). Therefore, it is left to find the rate for \( 1 \leq a < 2 \). By carefully calculating the probability that a word has two identical length-k subsequences, we show in this paper that the rate is indeed 1 for all \( a > 1 \), that is, the redundancy is \( o(n) \).

Motivated by several previous works [7], [13], we address the problem of calculating the capacity of k-repeat free sequences of length \( n \) where \( k = a \log n \) with \( a > 1 \). We provide, at first, an encoding algorithm that encodes into k-
repeat free words for $k = 2 + 2 \lfloor \log n \rfloor$, requiring only a single redundancy bit. The algorithm operates in two phases which also provide an intuitive guideline for the second algorithm which encodes into $k$-repeat free sequences for $k = a \log n$ with $a > 1$ and has rate 1. We also study the capacity of $k$-repeat free sequences which satisfy local constraints as well. For example, a combination of $k$-repeat free constraint and no adjacent zeros constraint ($(0,1)$ run length limited constraint). Perhaps surprisingly, we show that the $k$-repeat free constraint does not impose a rate penalty in either case.

The paper is organized as follows. In Section II, we present the notation and definitions which are used throughout the paper together with the capacity calculation of $k$-repeat free sequences. In Section III, we present an encoding algorithm for sequences of length $n$ with $k = a \log n$ for $a > 2$. An encoding algorithm for $k = a \log n$ with $1 < a \leq 2$ is presented in Section IV. In Section V, we discuss further results which include the capacity of $k$-repeat free sequences which also satisfy local constraints. Due to page limitation the proofs are

I. PRELIMINARIES

Let $\mathbb{N}$ denote the set of natural numbers. For $n \in \mathbb{N}$, we denote by $[n]$ the set $[n] = \{0, 1, \ldots, n-1\}$ and by $[-n]$ the set $[-n] = \{-1, -2, \ldots, -n\}$. Let $A$ be a subset of a group with a group operation $\cdot$. If $j$ is any group member, we define $j \cdot A = \{j \cdot a : a \in A\}$. For example, if $A = \{1,3,5,6\} \subseteq \mathbb{Z}$ and $r = -1$. Then, $A + r = \{0, 2, 4, 5\}$ and $A_r = \{-1, -3, 5, -6\}$.

Throughout the paper, we use $\Sigma$ to denote a finite alphabet. A word of length $n$ is a sequence of $n$ symbols from $\Sigma$ and is denoted by $w = (w_0, \ldots, w_{n-1})$. We denote by $\Sigma^n$ the set of all finite words of length $n$ over $\Sigma$ and by $\Sigma^* = \bigcup_{n \in \mathbb{N}} \Sigma^n$ the set of all finite words over $\Sigma$. We also denote by $\Sigma^k(\Sigma^*)$ the set of all infinite (bi-infinite) words over $\Sigma$. For a word $w \in \Sigma^*$, $|w|$ denotes the length of $w$ and $w_i = w(i)$ is simply the $i$th symbol in $w$.

For a word $w \in \Sigma^*$ and for a set (considered as a set of locations) $A \subseteq \Sigma^*$, $w_A$ denotes the restriction of $w$ to the symbols in the locations given by $A$. If $w, u \in \Sigma^*$ we denote by $wu \in \Sigma^{|w|+|u|}$ (or by $w \circ u$) the concatenation of $w$ and $u$. For $w \in \Sigma^*$ we write $w^\ell$ for the concatenation of $w$ with itself $\ell \in \mathbb{N}$ times. Unless otherwise is mentioned, coordinates of a word $w \in \Sigma^*$ are considered modulo $|w|$. Thus, if $w, u \in \Sigma^*$, we have $ww_{[0]} = w$ and $ww_{[-u]} = u$.

The main object studied in this paper is a set of words which we call a system. Specifically, we focus on sets which are defined using global constraints. One of the main characterizations of a system is given by the number of feasible words of length $n$. To be more specific, we would like to estimate the rate at which the number of $n$ length words grows with $n$. This number is called the capacity of the system and is defined as follows.

**Definition 1.** Let $\mathcal{L} \subseteq \Sigma^*$ be a system. The capacity of $\mathcal{L}$ is denoted by $\text{cap}(\mathcal{L})$ and is defined as

$$
\text{cap}(\mathcal{L}) = \limsup_{n \to \infty} \frac{1}{n} \log |\mathcal{L} \cap \Sigma^n|
$$

where the logarithm is of base $|\Sigma|$ unless mentioned otherwise.

The systems we will consider will be defined mostly using constraints on the number of sub-word appearances. To this end, we define the notion of empirical frequency.

**Definition 2.** Let $w \in \Sigma^n$ and $k \leq n$. The empirical frequency of $k$-tuples in $w$ is denoted by $\text{fr}_w^k$ and is defined as follows. For a $k$-tuple, $u \in \Sigma^k$,

$$
\text{fr}_w^k(u) \triangleq \frac{1}{n-k+1} \sum_{m=n-k+1}^n \mathbb{1}_u(w_m+[k])
$$

where $\mathbb{1}$ denotes the indicator function defined by $\mathbb{1}_a(b) = 1$ if $a = b$ and 0 otherwise. We will sometimes consider $\text{fr}_w^k$ as a vector of length $|\Sigma|^k$ or as a probability distribution.

The support of $\text{fr}_w^k$, $\text{supp}(\text{fr}_w^k)$, is the set of all $k$-tuples which appear in $w$.

**Example 3.** Let $\Sigma = \{0,1\}$ be the binary alphabet and let $w \in \Sigma^n$ be a binary De-Bruijn sequence of span $k$. Hence $\text{fr}_w^k$ is the empirical frequency of pairs. We have that $\text{fr}_w^k(01) = \frac{1}{2}$, $\text{fr}_w^k(10) = \frac{1}{2}$, $\text{fr}_w^k(11) = \frac{1}{2}$, and $\text{fr}_w^k(01) = \text{fr}_w^k(10) = \text{fr}_w^k(11) = \frac{1}{2}$.

One of the most important set of words to this work, is the set of (one-dimensional) De-Brujin sequences. For a finite alphabet $\Sigma$, and for $1 \leq k \in \mathbb{N}$, we say that a word $w$ is a De-Brujin word of span $k$ if every $k$-tuple appears in $w$ exactly once. Note that $w$ must be of length $|\Sigma|^k + k - 1$ (where it is possible to show that the $k-1$ suffix equals to the $k-1$ prefix) since there are exactly $|\Sigma|^k$ different $k$-tuples. Using our notation, we may define the system as follows.

**Definition 4.** Let $\Sigma$ be a finite alphabet with $|\Sigma| = q$. A word $w \in \Sigma^*$ is called a De-Brujin sequence of span $k$ if every $k$-tuple appears exactly once, i.e., for every $u \in \Sigma^k$,

$$
\text{fr}_w^k(u) = \frac{1}{|w| - k + 1}
$$

A De-Brujin system over the alphabet $\Sigma$ with $|\Sigma| = q$ is denoted by $B_q$ and defined as the set of all De-Brujin sequences (over $\Sigma$) of span $k$ for all $k \in \mathbb{N}$. In a notational form, a De-Brujin system is the set

$$
B_q = \left\{ w \in \Sigma^* : \exists k \in \mathbb{N} \text{ s.t. } \forall u \in \Sigma^k \frac{1}{|w| - k + 1} \right\}
$$

Note that by definition, a De-Brujin system contains all the De-Brujin sequences of span $k$, for every $k \in \mathbb{N}$. In fact, a De-Brujin system does not contain words of length that is not $|\Sigma|^k + k - 1$ for some $k$. Moreover, for an alphabet $\Sigma$, if $w \in B_q$ then $w_{[k-1]} = w_{[-k+1]}$.

The number of binary De-Brujin sequences of span $k$ is known due to De-Brujin himself who used the doubling process to calculate the exact number [2]. Later, his result was generalized to any alphabet [6]. For a finite alphabet $\Sigma$ with $|\Sigma| = q$, the number of De-Brujin sequences of span $k$ is given by $\left( (q-1)! \right)^{q^{k-1}} \cdot q^{k-1}$. Using this formula, the capacity of the De-Brujin system is given by

$$
\text{cap}(B_q) = \frac{1}{q} (\log_q(q!)).
$$

Thus, when $q = 2$, $\text{cap}(B_2) = \frac{1}{2}$ but using Stirling’s approximation, we obtain $\lim_{q \to \infty} \text{cap}(B_q) = 1$.

We may now introduce the first system we will consider in this work. One may regard it as a generalization of De-Brujin sequences.
Definition 5. A sequence \( w \in \Sigma^n \) is said to be \( k \)-repeat free (or, interchangeably, weak De-Bruijn of span \( k \)) if every \( k \)-tuple appears at most once. The set of length-\( n \) \( k \)-repeat free sequences is denoted by

\[
\mathcal{W}_k(n) = \left\{ w \in \Sigma^n : \forall u \in \Sigma^k, \text{fr}_k(u) \leq \frac{1}{n-k+1} \right\}.
\]

Note that \( \mathcal{W}_k(\Sigma^{|k|+k-1}) \) is exactly the set of all \( k \)-De-Bruijn sequences of span \( k \). For \( k = \lfloor a \log n \rfloor \) with \( a > 1 \), we define the weak De-Bruijn system as \( \mathcal{W}_a = \bigcup_{n \in \mathbb{N}} \mathcal{W}_k(n) \).

If \( k \leq \log_2(n+1-k) \) the set \( \mathcal{W}_k(n) = \emptyset \). Therefore, a natural question to ask is how the size \( |\mathcal{W}_k(n)| \) changes when \( k \) and \( n \) grow. Namely, we are interested in the set \( \mathcal{W}_k(n) \) where \( k > \log(n+1-k) \). The size \( |\mathcal{W}_k(n)| \) will be estimated using probability. Consider the uniform distribution over all \( n \) length sequences, then if \( w \) is chosen uniformly at random from \( \mathcal{W}_k(n) \), we have

\[
|\mathcal{W}_k(n)| = |\Sigma|^n \cdot \Pr(x \in \mathcal{W}_k(n)).
\]

The capacity, in this case, is given by

\[
\text{cap}(\mathcal{W}_a) = 1 + \lim_{n \to \infty} \frac{1}{n} \log \Pr(\mathcal{W}_k(n)). \tag{1}
\]

Although the capacity for some cases can be calculated using other methods (see for example [7]), we apply a different, general method for calculating the capacity. This method helps us to calculate the capacity for unknown cases as shown in this paper. We provide the following theorem and proof in order to present the basic idea.

Theorem 6. Let \( \Sigma \) be a finite alphabet of size \( q \) and consider \( \mathcal{W}_k(n) \) for \( k = \lfloor a \log n \rfloor \) where \( a > 1 \), then \( \text{cap}(\mathcal{W}_a) = 1 \).

Proof sketch: Let \( w \) be an infinite sequence such that the symbol in each position is chosen uniformly at random from an alphabet \( \Sigma \) of size \( q \). We will estimate the probability

\[
\Pr(w[a] \in \mathcal{W}_k(n)) = \frac{|\mathcal{W}_k(n)|}{q^n}.
\]

For \( 0 < \ell \in \mathbb{N} \) we define the random variable

\[
X_\ell = \sum_{k=(\ell-1)k+1}^{\ell k} \sum_{i=0}^{j-1} \mathbf{1}_{w_{i+k]}(w_{i+k})}
\]

and note that

\[
\Pr(\mathcal{W}_{(n+1)k} \in \mathcal{W}_a) = 1 - \Pr\left( \sum_{i=1}^{n} X_\ell \geq 1 \right).
\]

The next step is to show that \( \mathbb{E}(X_\ell) \leq \frac{(j+1)k^2}{q^\ell} \) by an explicit calculation. We then apply Markov inequality and obtain \( \Pr(X_\ell \geq 1) \leq \frac{(j+1)k^2}{q^\ell} \). This, together with the fact that for \( x \in (0,1) \), \( \log_q(1-x) \geq \frac{x}{(1-x)\log q} \) implies that

\[
\Pr(|\mathcal{W}_k(n)|) \geq q^{-\frac{1}{(\ell+1)k^2}} \geq q^{-(j+1)k^2} \frac{1}{(q^\ell - 1)\log q}.
\]

Taking logarithm and divide by \( n \) we obtain

\[
\frac{1}{n} \log_q(\Pr(|\mathcal{W}_k(n)|)) \geq \frac{1}{n(e - a \log_q n) \log q} - \frac{a \log_q n}{n(n_e - 1) \log q}
\]

which clearly goes to 0 as \( n \to \infty \). Thus, we obtain that \( \text{cap}(\mathcal{W}_a) = 1 \) for \( a > 1 \).

III. ALGORITHM FOR \( k = 2 + 2 \log n \)

We now provide a coding algorithm for the binary weak De-Bruijn system. For simplicity, throughout the section we assume \( \log n \) is an integer. We also assume that \( k = 2 + 2 \log n \). The input to the algorithm is a binary sequence \( w \in \Sigma^n \). The output is a weak De-Bruijn sequence \( \bar{w} \in \mathcal{W}_k(n) \). The algorithm presented here will be the basic step towards an algorithm for \( k = a \log n \) where \( a > 1 \). The algorithm is divided into two procedures - elimination and expansion. We first give a short overview of the algorithm. Given a sequence \( w \in \Sigma^n \), append \( 10^{1+\log n} \) to its end. Then, search for identical subsequences of length \( 2 \log n + 2 \). For every such an occurrence, remove one of them (the first one) and encode at the beginning of the sequence the indices of these two subsequences followed by the bit 0. Note that such an operation reduces the length of the sequence by one, and therefore this procedure is guaranteed to terminate. The second procedure takes this compressed sequence and decompresses it into a longer sequence such that the constraint is not violated. At the end, the output is the first \( n \) bits of the sequence.

Before presenting the algorithm, we need a few more notations. For an integer \( i \in [n] \), we let \( b(i) \) be its binary representation using \( \log n \) bits. We assume for now that \( \log n \) is an integer. Let \( w \in \Sigma^n \) be any word. Recall that for \( i \in \mathbb{N} \), \( i \leq |w| \), \( w_{[i]} \) is the \( i \)-length suffix of \( w \), i.e., \( w_{[i]} = w_{[|w|-i+1]} \). Moreover, the support of \( \text{fr}_a \), \( \text{Supp}(\text{fr}_a) \), is the set of all \( k \)-tuples that appear in \( w \). For a word \( w \in \Sigma^n \) and for \( m \in \mathbb{N} \) we denote by \( \text{Cr}_m(w) \) the word of length \( m \) created by repeatedly concatenating \( w \) to itself and taking the \( m \)-length prefix, i.e., \( \text{Cr}_m(w) = (w^{(m)}_{[m]}) \). We say that \( (i,j) \) (where \( i < j \)) is a \( k \)-identical window in \( w \) if \( w_{[i+k]} = w_{[j+k]} \). Algorithm 1 is given in the next page. The correctness of the algorithm is not presented here due to pages limitations.

The decoding procedure is relatively simple. Look first for the leftmost sequence of \( 01 \) \( \circ \) \( \log n + 1 \). According to the algorithm, everything to the right of this sequence was added during the expansion procedure which means it may be removed. If there is no such \( 01 \circ \log n + 1 \) window, look for the rightmost 1. Since the algorithm returns a sequence which is longer than \( 1 \) than the input sequence, the rightmost 1 (and the zeros following that 1) is a part of the initial set-up of the algorithm. Next, if the first symbol is 1, let \( i \) and \( j \) be the positions indicated by \( \text{Cr}_1(w) \), i.e., \( b(i) = b(j) \). Delete the first \( \log n + 1 \) bits and put \( \circ \log n + 1 \) in the \( i \)th position. If the first symbol is 0, let \( i \) and \( j \) be the positions indicated by \( \text{Cr}_0(w) \) and by \( \text{Cr}_1(w) \) respectively. Let \( u = \text{Cr}_0(w)_{[-1]} \), delete the first \( 2 \log n + 1 \) bits and put \( u \) in the \( i \)th position. Repeat this process until obtaining a sequence of length \( n \).

Example 7. Let \( n = 32 \) \((k = 2 \log(32) + 2 = 12) \) and

\[
\bar{w} = 01001001001101001001011011100110 \in \Sigma^{31}.
\]

Following the encoding algorithm we end up with the word

\[
\bar{w} = 10000111101001001101110011010.
\]
Algorithm 1 No-Identical Windows Encoding

Input: Sequence \( w \in \Sigma^{n-1} \)

Output: Sequence \( \overline{w} \in \mathcal{W}_k(n) \) with \( k = 2 \log n + 2 \)

First procedure (elimination):
1. Set \( \overline{w} = w \circ 1 \circ 0^{\log n + 1} \in \Sigma^{n+\log n+1} \)
2. while there are \( k \)-identical windows in \( \overline{w} \) or a \( 0^{\log n+1} \)-window in \( \overline{w}[\overline{w}[-1]] \) (check the 1st condition first) do
   3. Case 1: (there are \( k \)-identical windows in \( \overline{w} \))
      4. Let \((i, j)\) be a \( k \)-identical window in \( \overline{w} \)
      5. Set \( \overline{w} = \overline{w}[i] \circ \overline{w}[i+k+\overline{w}[i-k-1]] \) (remove the first \( k \)-length repeated window from \( \overline{w} \))
      6. Set \( \overline{w} = 0 \circ b(i) \circ b(j) \circ \overline{w} \) (append \( 0 \circ b(i) \circ b(j) \) to the left of \( \overline{w} \))
   7. Case 2: (there is a \( 0^{\log n+1} \)-window in \( \overline{w}[\overline{w}[-1]] \))
      8. Let \( i \) be the index of the \( 0^{\log n+1} \)-window in \( \overline{w} \)
      9. Set \( \overline{w} = \overline{w}[i] \circ \overline{w}[i+k+\overline{w}[i-i-k]] \) (remove the \( 0^{\log n+1} \)-window from \( \overline{w} \))
    10. Set \( \overline{w} = 1 \circ b(i) \circ b(j) \) (append \( 1 \circ b(i) \) to the left of \( \overline{w} \))
11. end while
12. if \(|\overline{w}| \geq n\) then
    13. Return \( \overline{w}[n] \)
14. end if

Second procedure (expansion):
15. while \(|\overline{w}| < n\) do
16. Set \( B = \text{Supp} \left( \hat{f}_{\overline{w}}^{\log n} \right) \cup \bigcup_{1 \leq i \leq \log n - 1} \text{Cr}_{\log n}((\overline{w}[i])) \).
17. Set \( S = \Sigma^{\log n} \backslash B \) and find \( u \in S \)
18. Set \( \overline{w} = \overline{w} \circ (\log n) \) (append \( u \) to the right of \( \overline{w} \))
19. end while
20. Return \( \overline{w}[n] \)

For the decoding process, we are to obtain \( w \) from \( \overline{w} \). First, we look for the left most 1000000 in \( \overline{w} \). Since there is no such sequence, we look for the right most 1 and we know that this bit with all the following zeros were added in the set-up. Meaning that the last 10 are not apart of \( w \). We eliminate those bits and we obtain

\[
\hat{w} = 100001 1 1101 001001001101110011001 \in \Sigma^{30}.
\]

Since \( \hat{w} \in \Sigma^{30} \) we know that there were only one identical pair of \( k \)-length windows. The first bit in \( \hat{w} \) is 1. Thus, we have \( i = b(00001) = 1 \). We eliminate the first 6 bits and put 6 zeros in the first position.

\[
\hat{w} = 1 000000 1101 001001001101110011001.
\]

Again, the first bit is 1 and next 5 bits indicate the position 0. We eliminate the first 6 bits and put 000000 in the 0 position.

\[
\hat{w} = 000000 0 1101 001001001101110011001.
\]

We are now having 0 for the first bit and the next 10 bits indicate two positions, \( i = 0, j = 13 \). We denote

\[
x = \pi_{[12]}(\sigma^{12}(\hat{w})) = 010010011011\).
\]

We now eliminate the first 11 bits and put \( x \) in the \( i \)th position and obtain

\[
\hat{w} = 010010011011 0010010011011100110.
\]

Since \( \hat{w} \in \Sigma^{31} \) we are done.

IV. ALGORITHM FOR \( k = a \log n \) WITH \( 1 < a < 2 \)

We now consider the case of \( k = a \log n \) where \( a < 2 \) and \( \log n \) is an integer. Similar to before, our coding scheme consists of two basic procedures - elimination and expansion. For the elimination phase, we compress an input sequence into an output sequence of length at most \( n \). At every step of the compression, we remove identical windows of size \( k' \) so that at the end of this step, the output sequence does not contain any identical windows of size \( k' \). For the elimination phase, we rely on an encoding procedure very similar to [7]. The process ensures that the sequence output from our encoder does not contain any all-zeros substrings of length greater than \( 2 \log \log n \).

The compression phase is the primary difference between the approach outlined here and previous work. The idea behind the compression phase is to concatenate a run-length limited de Bruijn sequence, which we refer to as \( d \in \{0,1\}^n \), with our compressed sequence \( e \in \{0,1\}^n \), and then insert within \( d \), all-zeros markers of length \( 4 \log \log n \). These markers will be used to distinguish (or to make different) the length \( k \) windows between \( d \) and \( e \). We will explain these ideas in more detail in what follows.

First, we introduce some notation. We say that a sequence \( u \in \{0,1\}^n \) is zero-constrained if there are no all-zeros substrings of length \( 2 \log \log |u| \). If \((i, j)\) is a \( k \)-identical window such that for any other \( k \)-identical window at \((i', j')\) in \( u \), we have \( j \leq j' \), we say that \((i, j)\) is a primal \( k \)-identical window. Let \( B_r(i) : [n] \rightarrow \{0,1\}^{2 \log \log n} \) be an invertible function whose image are zero-constrained sequences. Note that for \( n \) large enough, such a function exists.

The elimination encoder \( E_{el} \), described below, takes as input a sequence \( u \in \{0,1\}^n \) where \( u \) zero-constrained. The output of \( E_{el} \) is a sequence \( z \) of length at most \( n - (4 \log \log n + 3) \) that is zero-constrained and does not contain any repeated windows of length \( k' = \log n + 2 \log \log n + 5 \).

Algorithm 2 Elimination Encoder, \( E_{el} \)

1. Set \( \overline{u} = u \)
2. while there are identical length-\( k' \) windows in \( \overline{u} \) do
3. Suppose \((i, j)\) is a primal \( k' \)-identical window in \( \overline{u} \)
4. Remove the substring of length \( k' \) starting at position \( j \) and replace it with the sequence \((1, 0^{2 \log \log n}, 1, B_r(i), 1)\), so that
5. \( \overline{u} = \overline{u}[i] \circ (0^{2 \log \log n} 1 B_r(i) 1) \circ \overline{u}[i+k'+k'+1, .., |\overline{w}[-1]|] \)
6. end while
7. \( z = \overline{u} \)

Note that since at step 4) we replace substrings of length \( k' \) with substrings of length \( k' - 1 = \log n + 2 \log \log n + 4 \),
so that each time step 4) is executed, the length of $\overline{u}$ is decremented by one. We are now left with the “inflation” part. Lengthening the word $z$ without disobeying the restriction requires a different approach than Algorithm 1. We will show that there exists a word $d$ that may be concatenated to $z$ such that the concatenation will be $k$ repeat free. This is given in the next lemma.

**Lemma 8.** There exists a sequence $d$ (which may be obtained explicitly using concatenation of Lyndon words) such that 

$$w = \left( z^{10^4 \log \log n + 11d} \right)^{[n]}$$

does not contain any repeated windows of length $k = \log n + 10 \log \log n + 10 = k + 8 \log \log n + 5$.

We now present our main result, which follows from the previous discussion.

**Theorem 9.** There exists a rate-1 polynomial-time encoder which generates sequences with no-identical $k$-windows for any $k > a \log n$ where $a > 1$.

### V. Further Results

In this section we discuss further results regarding the combination of weak De-Bruijn sequences and local constraints. Due to lack of space, we will only mention the basic definitions and the main result of this section. Unfortunately, calculating the exact number of De-Bruijn sequences which also satisfy other constraints is not an easy problem [10]. Here, we calculate the capacity of weak De-Bruijn sequences with local constraints. For convenience, throughout this section we restrict $\Sigma$ to the binary alphabet but the same method can be used to greater alphabets.

Before stating the main result of this section, we remind the reader some known definitions and basic results on constrained systems. A constrained system $S \subseteq \Sigma^N$ is the set of all words obtained by reading the labels of paths in a labeled directed graph $G$. The constrained system is said to be irreducible if $G$ is strongly connected and deterministic if edges going out from the same vertex carry different labels. We denote by $B_n(S)$ the set of all words of length $n$ in $S$. For a graph $G$, the Perron eigenvalue is the maximal eigenvalue of the adjacency matrix of $G$.

We are interested in constrained systems which are also weak De-Bruijn. Meaning, if $S$ is a constrained system, we are interested in the following set of words.

**Definition 10.** Let $S$ be an irreducible deterministic constrained system and let $W_k(n)$ denote the weak De-Bruijn sequences of span $k$. The weak De-Bruijn system with constraints $S$ is defined by the following sets.

$$X_k(n) = \{w \in \Sigma^n : w \in W_k(n) \cap B_n(S)\}.$$ 

For $k = k(n)$, we define the system $X = \bigcup_{n \in \mathbb{N}} X_k(n)$.

We now state the main result of this section.

**Theorem 11.** Let $S$ be an irreducible constrained system given by the graph $G$ with Perron eigenvalue $\lambda$. For every $n \in \mathbb{N}$ let $k = a \log \lambda n$ with $a > 1$. Then 

$$\text{cap}(X) = \text{cap}(S).$$

We will also mention that using methods similar to the ones presented in this paper, it is possible to calculate the capacity of weak De-Bruijn multidimensional arrays and show that under mild conditions over $k$, the capacity is also 1. The multidimensional case impose more difficulties since even the case of enumerating the De-Bruijn arrays is not fully solved.

### VI. Acknowledgment

The authors would like to thank Ofer Zeitouni for helpful discussion.

### References

[1] J. Acharya, H. Das, O. Milenkovic, A. Orlitsky, and S. Pan, “On reconstructing a string from its substring compositions,” in Information Theory Proceedings (ISIT), 2010 IEEE International Symposium on. IEEE, 2010, pp. 1238–1242.

[2] N. G. D. Brijui, “A combinatorial problem,” Koninklijke Nederlandse Akademie v. Wetenschappen, vol. 49, no. 49, pp. 758–764, 1946.

[3] Z. Chang, J. Chrisnata, M. F. Ezerman, and H. M. Kiah, “Rates of dna sequence profiles for practical values of read lengths,” IEEE Transactions on Information Theory, vol. 63, no. 11, pp. 7166–7179, 2017.

[4] M. Dukud and L. J. Schulman, “Reconstruction from subsequences,” Journal of Combinatorial Theory, Series A, vol. 103, no. 2, pp. 337–348, 2003.

[5] O. Elishco, F. Farnoud, M. Schwartz, and J. Bruck, “The capacity of some polya string models,” CoRR, vol. abs/1808.06062, 2018. [Online]. Available: http://arxiv.org/abs/1808.06062.

[6] H. Fredricksen, “A survey of full length nonlinear shift register cycle algorithms,” SIAM review, vol. 24, no. 2, pp. 195–221, 1982.

[7] R. Gabrys and O. Milenkovic, “Unique reconstruction of coded sequences from multiset substring spectra,” in Proceedings of the IEEE International Symposium on Information Theory, Vail, Colorado, USA, 2018, pp. 2540–2544.

[8] S. Jain, F. Farnoud, M. Schwartz, and J. Bruck, “Duplication-correcting codes for data storage in the dna of living organisms,” IEEE Transactions on Information Theory, vol. 63, no. 8, pp. 4996–5010, 2017.

[9] H. M. Kiah, G. J. Puleo, and O. Milenkovic, “Codes for dna sequence profiles,” IEEE Transactions on Information Theory, vol. 62, no. 6, pp. 3125–3146, 2016.

[10] O. Kupferman and G. Vardi, “Eulerian paths with regular constraints,” in mfcsl16, ser. Leibniz International Proceedings in Informatics (LIPIcs), vol. 58, 2016, pp. 1–62.

[11] A. Lenz, A. Wachter-Zeh, and E. Yaakobi, “Bounds on codes correcting tandem and palindromic duplications,” arXiv preprint arXiv:1707.00052, 2017.

[12] V. I. Levenshtein, “Efficient reconstruction of sequences from their subsequences or supersequences,” J. Combin. Theory Ser. A, vol. 93, no. 2, pp. 310–332, Feb. 2001.

[13] M. Levy and E. Yaakobi, “Mutually uncorrelated codes for dna storage,” IEEE Transactions on Information Theory, To appear.

[14] H. Lou, F. Farnoud, M. Schwartz, and J. Bruck, “Evolution of evolutionary frequencies and entropy in duplication and substitution mutation algorithms,” arXiv preprint arXiv:1812.02250, 2018.

[15] A. D. Scott, “Reconstructing sequences,” Discrete Mathematics, vol. 175, no. 1-3, pp. 231–238, 1997.

[16] E. Ukkonen, “Approximate string-matching with q-grams and maximal matches,” Theoretical computer science, vol. 92, no. 1, pp. 191–211, 1992.

[17] A. Wachter-Zeh, “List decoding of insertions and deletions,” IEEE Transactions on Information Theory, vol. 64, no. 9, pp. 6297–6304, 2018.