Cut-Down de Bruijn Sequences

Ben Cameron
Wilfred Laurier University, Canada

Aysu Gündoğan
University of Guelph, Canada

Joe Sawada
University of Guelph, Canada

Abstract

A cut-down de Bruijn sequence is a cyclic string of length $1 \leq L \leq k^n$ such that every substring of length-n appears *at most* once. Etzion [*Theor. Comp. Sci* 44 (1986)] gives an algorithm to construct binary cut-down de Bruijn sequences that requires $o(n)$ n-bit operations per symbol generated. In this paper, we simplify the algorithm and improve the running time to $O(n)$ time per symbol generated using $O(n)$ space. Furthermore, we provide the first successor-rule approach for constructing a binary cut-down de Bruijn sequence by leveraging recent ranking algorithms for fixed-density Lyndon words. Finally, we outline how to extend our results to produce an algorithm to construct cut-down de Bruijn sequences for $k > 2$ that runs in $O(n)$ time per symbol generated using $O(n)$ space.

1 Introduction

A *de Bruijn sequence* is a cyclic sequence of length k^n such that every k-ary string of length n appears as a substring exactly once. For example, the following is a de Bruijn sequence for $n = 6$ and $k = 2$:

$$000000111111011110011100011011010011000010111010110010101000 1001.$$

For some applications it may be more convenient to produce a cycle of arbitrary length such that there are no repeated length-n substrings. For instance, it may be more natural to consider the de Bruijn card trick [4] using 52 cards rather than 32. Also, for applications in robotic vision and location detection [24, 23, 4], instead of forcing a location map to have length k^n, an arbitrary length may be more appropriate. This gives rise to the notion of a *cut-down de Bruijn sequence*, which is a cyclic sequence over an alphabet of size k having length $1 \leq L \leq k^n$ such that every substring of length n appears *at most* once. As an example, the following is a binary cut-down de Bruijn sequence of length 52:

$$0000001111001110001101101001100001011101011001010001.$$

Note that every substring of length 6, including in the wraparound, appears at most once. Any cut-down de Bruijn sequence of length L with respect to k and n is also a cut-down de Bruijn sequence with respect to k and $n + 1$ [5]. Thus, throughout this paper we assume $k^{n-1} < L \leq k^n$.

Cut-down de Bruijn sequences are known to exist for all lengths L and any alphabet of size k [18] (for k=2 see [25]). In bioinformatics, the alphabet $\{C, G, A, T\}$ of size $k = 4$ is of particular interest and there are a number of applications that apply de Bruijn sequences and their relatives [21, 2]. The primary focus of this paper is when $k = 2$, though our work can be extended to $k > 2$ as outlined in Section 6.

A simple algorithm to construct binary cut-down de Bruijn sequences based on linear feedback shift registers and primitive polynomials is given by Golomb [11, P. 193]; it runs in $O(n)$-amortized

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

time per symbol using $O(n)$ space. However, the construction has an exponential-time delay before producing the first symbol, requires a specific primitive polynomial for each order n, and there is no way to determine if a given length-n string appears as a substring without generating the entire cycle. This approach is generalized to construct cut-down de Bruijn sequences where k is a prime power, and subsequently extended to handle arbitrary sized alphabets by applying additional number theoretic results [16]. Although no formal algorithmic analysis is provided, the formulation appears to share properties no better than the related binary construction. An algebraic approach for when k is a prime power is also known [13].

A *generalized de Bruijn sequence* is a cut-down de Bruijn sequence with length $k^{n-1} < m \leq k^n$ with an additional property: *every k-ary string of length $n-1$ appears as a substring*. Their existence is known for all m and k [7]. An algorithm based on Lempel's D-morphism [17] can be used to construct these sequences [20] that have an even stronger property: *every k-ary string of length $j \leq L$ appears either $\lfloor L/k^j \rfloor$ or $\lceil L/k^j \rceil$ times as a substring*. The algorithm can generate the sequences in $O(1)$-amortized time per symbol, but requires it exponential space and there is an exponential time delay before outputting the first symbol.

A cycle-joining based approach to construct cut-down de Bruijn sequences was developed by Etzion [5] for $k = 2$; it requires $o(n)$ n-bit operations to generate each symbol. Their approach follows two main steps. First, they construct an initial main cycle with length $L+s$, where $0 \leq s < n$, using the well-known cycle-joining approach. Second, depending on s, they detect and remove up to $\lceil \log n \rceil$ small cycles to obtain a cycle of length L. The resulting algorithm can construct an exponential number of cut-down de Bruijn sequences for any given L; however, their algorithm is not optimized to generate a single cut-down de Bruijn sequence. Etzion's construction also has a downside for some applications: It starts with a specific length-n string and the context matters to produce successive symbols. This means the resulting cycle does not have a corresponding successor rule, and furthermore, testing whether or not a specific string belongs to the cycle may involve generating the entire cycle.

The main results of this paper are as follows:

1. We simplify Etzion's approach and develop an algorithm to construct a binary cut-down de Bruijn sequence in $O(n)$ time per symbol using $O(n)$ space.
2. We develop a successor-rule approach to construct a binary cut-down de Bruijn sequence in $O(n^{1.5})$-amortized time per symbol based on the unit-cost RAM model. The algorithm can start with any string on the cycle and the context does not matter when producing successive symbols.
3. We develop an algorithm to generate cut-down de Bruijn sequences for $k > 2$ that runs in $O(n)$ time per symbol using $O(n)$ space. A number of non-trivial adaptations to the binary algorithm are required to generalize to larger alphabets. Only an overview of this result is presented.

Outline of paper. In Section 2, we provide some background on the cycle-joining method and a simple successor rule to construct de Bruijn sequences. In Section 3, we review Etzion's approach for constructing cut-down de Bruijn sequences. In Section 4, we present in detail our simplified algorithm to construct binary cut-down de Bruijn sequences. In Section 5, we present the first successor-rule algorithm for constructing cut-down de Bruijn sequences. We conclude in Section 6 outlining the steps required to extend our work when $k > 2$. Implementation of our algorithms, written in C, are available for download at `http://debruijnsequence.org` [1]; this resource also provides a background on different ways de Bruijn sequences can be constructed.

2 Background

Let Σ denote an alphabet of size $k \geq 2$. Let Σ^n denote the set of all length-n strings over Σ. Let $\alpha = a_1 a_2 \cdots a_n$ be a string in Σ^n. Let α^t denote t copies of α concatenated together. The *period*

of α, denoted $\mathrm{per}(\alpha)$, is the smallest integer p such that $\alpha = (a_1 \cdots a_p)^t$ for some $t > 0$. If α has period less than n it is said to be *periodic*; otherwise it is *aperiodic*. The lexicographically smallest element in an equivalence class of words under rotation is called a *necklace*. A *Lyndon word* is an aperiodic necklace. The *weight* (or density) of a binary string is the number of 1s it contains.

A *feedback function* is a function $f : \Sigma^n \to \Sigma$. A *feedback shift register* (FSR) is a function $F : \Sigma^n \to \Sigma^n$ defined as $F(\alpha) = a_2 a_3 \cdots a_n f(\alpha)$, given a feedback function f. An FSR is said to be *nonsingular* if it is one-to-one.

The *pure cycling register* (PCR) is the FSR with feedback function $f(\alpha) = a_1$. It partitions Σ^n into an equivalence class of strings under rotation. Thus, the cycles induced by the PCR are in one-to-one correspondence with the necklaces of order n and also with Lyndon words whose lengths divide n. We use the notation $[\alpha]$ to denote a cycle. Let the weight of a cycle induced by the PCR be the number of 1s in the length-n strings obtained by traversing the cycle. For example, when $n = 6$ the weight of $[000001]$ is one and the weight of $[01]$ is three.

Example 1 Let $\Sigma = 0, 1$ and let $n = 6$. The following are the 14 equivalence classes of Σ^6 under rotation, where the first string in each class is a necklace. The symbols in the first column of each class (read top down) are the cycles induced by the PCR which also correspond to all the Lyndon words with length 1, 2, 3 and 6 (lengths that divide $n = 6$).

000000	000001	000011	000101	000111	001001	001011
	000010	000110	001010	001110	010010	010110
	000100	001100	010100	011100	100100	101100
	001000	011000	101000	111000		011001
	010000	110000	010001	110001		110010
	100000	100001	100010	100011		100101

001101	001111	010101	010111	011011	011111	111111
011010	011110	101010	101110	110110	111110	
110100	111100		011101	101101	111101	
101001	111001		111010		111011	
010011	110011		110101		110111	
100110	100111		101011		101111	

The following 14 cycles induced by the PCR are in one-to-one correspondence with the set of Lyndon words of lengths 1,2,3, and 6:

[0], [1], [01], [001], [011], [000001], [000011], [000101], [000111], [001011], [001101], [001111], [010111], [011111].

One of the most common ways to construct a binary de Bruijn sequence is by applying the *cycle-joining method*, which is akin to Hierholzer's method for finding Euler cycles in graphs [14]. This approach repeatedly joins pairs of cycles through *conjugate pairs*; one cycle contains $a_1 a_2 \cdots a_n$, and the other contains $\overline{a}_1 a_2 \cdots a_n$, where $\overline{x}$ denotes the complement of x. When the initial cycles are those induced by an underlying nonsingular FSR, the joining of the cycles can be viewed as a tree. As an example, Figure 1 illustrates the cycles induced by the PCR for $n = 6$ and how they can be joined together to create a de Bruijn sequence. Interestingly, when extending this idea to larger alphabet sizes, the tree visualization no longer applies in general (see [9]).

A *universal cycle* for a set $\mathbf{S}$ of length-n strings is circular string of length $|\mathbf{S}|$ such that every string in $\mathbf{S}$ appears as a substring exactly once. Of course, a de Bruijn sequence is a special case of a universal cycle when $\mathbf{S}$ corresponds to all k-ary strings of length n. A cut-down de Bruijn sequence

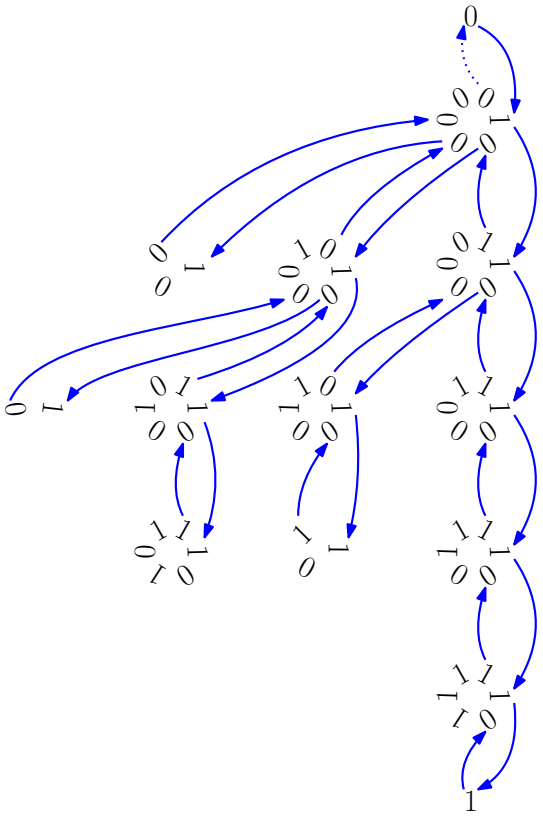


Figure 1 The 14 cycles induced by the PCR joined by applying the de Bruijn successor PCR3.

of length L also corresponds to a universal cycle of length L; however, the corresponding set $\mathbf{S}$ is not necessarily known a priori. A *UC-successor* for $\mathbf{S}$ is a feedback function whose corresponding FSR can be repeatedly applied to construct a universal cycle for $\mathbf{S}$. When $\mathbf{S} = \Sigma^n$ a UC-successor is said to be a *de Bruijn-successor*. A general framework based on the cycle joining approach leads to many simple UC-successors [8]. Application of this framework rediscovers many previously known de Bruijn sequence constructions including one by Jansen [15] that was revisited in [22] with respect to the PCR. This de Bruijn successor, based on the feedback function PCR3 defined in [8] and restated below, is perhaps the simplest of all de Bruijn successors.

PCR3 de Bruijn successor:

$$\mathrm{PCR3}(\alpha) = \begin{cases} \overline{a}_1 & \text{if } a_2a_3\cdots a_n1 \text{ is a necklace;} \\ a_1 & \text{otherwise.} \end{cases}$$

When the FSR with feedback function PCR3 is repeatedly applied to the starting string 000000 for $n = 6$, as illustrated in Figure 1, it produces the following de Bruijn sequence when the first bit is output before each application of the rule:

$$0000001111110111100111000110110100110000101110101100101010001001.$$

It is worth mentioning that the other simple PCR-based successors stated in [8] do not share all the properties we prove later for PCR3 that allow for a simple construction of cut-down de Bruijn

sequences.

3 Etzion's approach

In this section we outline Etzion's approach [5] for constructing a binary cut-down de Bruijn sequence. Recall that L is the length of the cut-down de Bruijn sequence and $2^{n-1} < L \leq 2^n$. The two primary steps in Etzion's construction are as follows:

1. Describe a Main Cycle (MC) that has length $L + s$ where $0 \leq s < n$.
2. Describe $\lceil \log s \rceil$ small cycles that can be removed from the MC to yield a cycle of the desired length L.

To construct the MC, a subset of the cycles induced by the PCR based on their weight and period are selected to be part of the MC. Enumeration of strings by weight and period determine which cycles to include. Let

- $\mathbf{A}(n, w)$ denote the set of binary strings of length n and weight less than or equal to w,
- $\mathbf{B}(n, w, p)$ denote the set of binary strings of length n, weight w, and period p,
- $\mathbf{C}(n, w, p)$ denote the set of binary strings of length n, weight w, and period $\leq p$, and
- $\mathbf{L}(n, w)$ denote the set of Lyndon words of length-n and weight w.

Let $A(n, w)$, $B(n, w, p)$, $C(n, w, p)$, and $L(n, w)$ denote the cardinalities of these four sets, respectively. Clearly $A(n, w) = \sum_{j=0}^{w} \binom{n}{j}$, and from previous observations $B(n, w, p) = pL(p, wp/n)$ assuming p divides n. It is well-known that

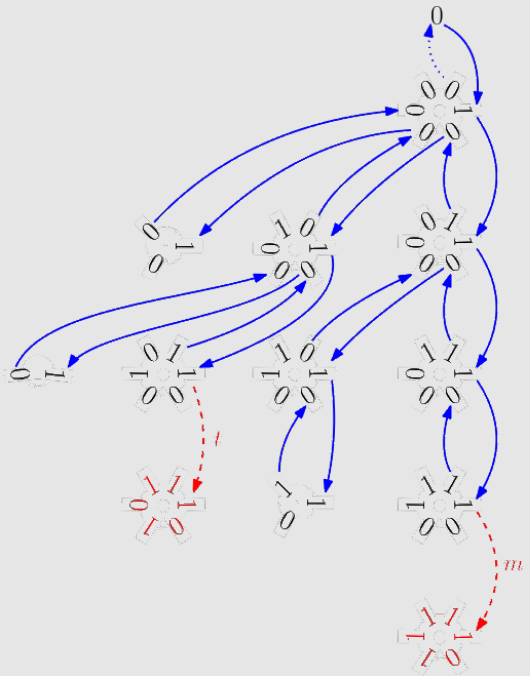
$$L(n, w) = \frac{1}{n} \sum_{i \mid \gcd(n,w)} \mu(i) \binom{n/i}{d/i},$$

where μ is the Mobiüs function [10]. Using the above values, the following four values m, t, h, s are precomputed as follows:

- m denotes the smallest weight m such that $A(n, m) \geq L$,
- h denotes the smallest period such that $A(n, m-1) + C(n, m, h) \geq L$,
- t denotes the smallest integer such that $A(n, m-1) + C(n, m, h-1) + th \geq L$, and
- since $h \leq n$, $A(n, m-1) + C(n, m, h-1) + th = L + s$ for some surplus $0 \leq s < n$.

An MC is the result of joining together all PCR cycles of weight less than m together with all PCR cycles of weight equal to m and period less than h together with exactly t PCR cycles of weight m and period h. Note that Etzion's original presentation adds cycles of weight m starting with the largest period. We made one minor departure from this approach by adding cycles of weight m starting from the smallest period, which handles a special case defined later. Let $\mathbf{T}$ denote a set containing the length-n strings from the latter t PCR cycles of weight m and period h. Thus the MC contains all strings in $\mathbf{S} = \mathbf{A}(n, m-1) \cup \mathbf{B}(n, m, h-1) \cup \mathbf{T}$ as substrings. As cycles are joined, a counter is maintained to keep track of the number of cycles of weight m and period h already joined into the MC. Thus the set $\mathbf{T}$ is not pre-determined. Let $\mathcal{S}_n(m, h, t)$ denote the set of all possible sets $\mathbf{S}$.

Example 2 Consider $L = 46$ and $n = 6$. Since $A(n, 3) = 42$ and $A(n, 4) = 57$, we have $m = 4$. Since $B(6, 4, 1) = B(6, 4, 2) = 0$, $B(6, 4, 3) = 3$, $B(6, 4, 4) = B(6, 4, 5) = 0$, $B(6, 4, 6) = 12$, we have $C(6, 4, 5) = 3$ and $C(6, 4, 6) = 15$. Thus $h = 6$. Since $A(6, 3) + C(6, 4, 5) + h = 42 + 3 + 6 = 51$, we have $t = 1$ and surplus $s = 5$. Applying PCR3 as the underlying method for joining cycles, the following illustrates the construction of the MC starting with 000000.



The MC joins all cycles with weight less than $m = 4$, all cycles with weight $m = 4$ and period less than $h = 6$, and exactly $t = 1$ cycles with weight $m = 4$ and period $h = 6$. Note that the cycle pointed to by the dashed arc labeled m is not added since it has weight greater than $m = 4$. The cycle pointed to by the dashed arc labeled t is not added since there was already $t = 1$ cycles added with weight $m = 4$ and period $h = 6$. The resulting MC is

$$000000111100111000110110100110000101100101010001001.$$

The second step involves *cutting out* small cycles whose combined length totals s. For instance, when $s = 1$, the cycle $[0]$ is easily removed. However, for arbitrary lengths, finding and removing such small cycles is non-trivial and the construction of the MC become critical for the ease in which these small cycles can be removed. Etzion's approach requires cutting out up to $\lceil \log n \rceil$ cycles of the form $[0^i 1]$ where $i + 1 < n$ is a power of 2. For example, when $n = 14$, the possible cycles to remove would be of the form $[01]$, $[0001]$, $[00000001]$. Depending on s, the cycle $[0]$ may also need to be cut-out (the removal of a single 0). Details on how to cut such cycles out are discussed in the next section. There are two special cases that Etzion addresses to ensure that the aforementioned small cycles are indeed on the MC.

- **Special case #1:** When $n = 2m$, the the cycle $[01]$ with weight m must be included.
- **Special case #2:** When $n = 2m - 1$ the cycle $[(01)^{m-1}1]$ with weight m must be included.

As noted above, by considering the cycles of weight m in increasing order by period, **Special case #1** is handled since the cycle $[01]$ is the unique cycle with period 2 and will always be added in that case. **Special case #2** requires extra care when adding cycles of weight m and period n, noting that there are clearly no cycles of weight m and period less than n when $n = 2m - 1$.

4 Efficiently constructing binary cut-down de Bruijn sequences

The key steps to improving and simplifying Etzion's original approach are as follows.

1. We apply PCR3 and focus on a single cut-down de Bruijn sequence construction.
2. We consider all cycles of the form $[0^i1]$ for $1 \leq i \leq \lceil n/2 \rceil$ and cut out at most two small cycles. This requires changing the definition of PCR3 for at most two strings.
3. When defining the MC, we add the cycles of weight m by increasing (instead of decreasing) period, as already outlined in the previous section. This handles **Special case #1**.

4.1 Constructing the Main Cycle with PCR3

Based on the precomputed variables m, h, t described in the previous section, recall each set $\mathbf{S} \in \mathcal{S}_n(m, h, t)$ contains a unique subset $\mathbf{T}$ containing ht strings of weight m and period h. Given any set $\mathbf{S} \in \mathcal{S}_n(m, h, t)$, Algorithm 1 applies the feedback function PCR3 to construct a universal cycle for $\mathbf{S}$ starting from any string in $\mathbf{S}$.

Algorithm 1 Pseudocode for constructing a universal cycle for $\mathbf{S} \in \mathcal{S}_n(m, h, t)$ assuming pre-computed values m, h, t, s and the set $\mathbf{S}$. It will have length $L + s$ and assumes $\alpha \in \mathbf{S}$.

```
1: procedure MC(α = a_1 a_2 ··· a_n)
2:     for i ← 1 to L + s do
3:         PRINT(a_1)
4:         x ← PCR3(α)
5:         β = a_2 ··· a_n x
6:         if β ∉ S then x ← x̄
7:         α ← a_2 ··· a_n x
```

▶ **Lemma 1.** *Algorithm 1 generates a universal cycle for any* $\mathbf{S} \in \mathcal{S}_n(m, h, t)$.

Proof. Based on the proof of Theorem 4.3 in [8], PCR3 induces a spanning tree of the PCR cycles rooted at $[0]$ where the parent of a cycle with weight w is a cycle of weight $w - 1$ (see Figure 1). Based on the definition of $\mathbf{S}$, this means the cycles from strings in $\mathbf{S}$ are also connected under PCR3. Since line 6 complements the output of PCR3 when the resulting string would land outside $\mathbf{S}$, the Algorithm 1 cuts out exactly those cycles containing strings that are not in $\mathbf{S}$. ◀

Given $\mathbf{S} \in \mathcal{S}_n(m, h, t)$, let $MC_{\mathbf{S}}$ denote the universal cycle for $\mathbf{S}$ obtained from Algorithm 1. We now demonstrate how and when small cycles can be cut out of $MC_{\mathbf{S}}$ to obtain a universal cycle of the desired length L.

4.2 Cutting out small cycles

In order to cut-down $MC_{\mathbf{S}}$ to a cycle of length L, ideally we could just cut out a single substring of length s. However, cutting out such a substring without introducing duplicate length-n substrings is a challenge. When $s \leq \lceil \frac{n}{2} \rceil$ we will demonstrate that finding such a substring is possible; otherwise we cut out two substrings whose combined length total s.

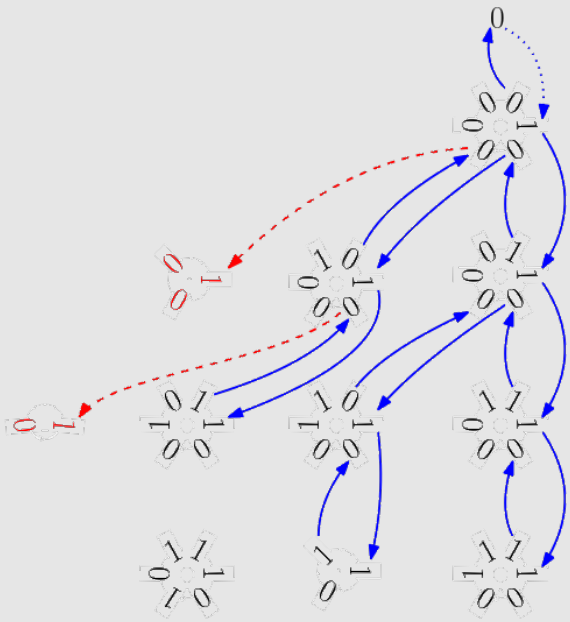
When $s = 1$, it is straightforward to cut-down the unique substring 0^n down to 0^{n-1}. Otherwise we consider cycles of the form $Z_i = [0^{i-1}1]$ for $1 < i \leq \lceil \frac{n}{2} \rceil$. Etzion [5] considers similar cycles, but only those with length that are a power of two. Let $\mathbf{Z}_i$ denote the set of length-n strings that are found on the cycle Z_i. When i divides n, Z_i corresponds to a PCR cycle. For example if $n = 6$ then $\mathbf{Z}_2 = \{010101, 101010\}$ and $\mathbf{Z}_3 = \{001001, 010010, 100100\}$. However, if i does not divide

n, then the strings in $\mathbf{Z}_i$ will appear in two PCR cycles with different weights. For example when $n = 6$, $\mathbf{Z}_5 = \{000010, 000100, 001000, 010000, 100001\}$. These strings appear on the PCR cycles $[000001]$ and $[000011]$. Observe that such cycles will have weight at most $\lceil \frac{n}{2} \rceil$, which is observed when $i = 2$ for $Z_2 = [01]$, and thus they always appear on our MC as long as we account for the two special cases defined at the end of Section 3. To cut out the strings from a given cycle Z_i, we demonstrate it suffices to change the successor for one string γ_i by complementing its output from PCR3. These strings are defined as follows:

$$\gamma_i = \begin{cases} 10^{n-1} & \text{if } i = 1; \\ 0(0^{i-1}1)^{a-1}0^{i-1} & \text{if } i \text{ divides } n; \\ 10^b(10^{i-1})^a & \text{otherwise,} \end{cases}$$

where $a = \lfloor \frac{n}{i} \rfloor$ and $b - 1 = n \bmod i$. When i divides n, complementing the output of PCR3 amounts to cutting out the cycle Z_i entirely; otherwise a more careful analysis is required.

Example 3 Recall the MC from Example 2 where $L = 46$ and $s = 5$. By cutting out the cycles $[001]$ and $[01]$, we obtain a cut-down de Bruijn sequence of length L. This is done by complementing the PCR3 output for the strings $\gamma_3 = 000100$ and $\gamma_2 = 001010$ in Algorithm 1 as illustrated below.



The dashed red edges indicate the output from PCR3 if γ_3 and γ_2 are not complemented. The resulting cut-down de Bruijn sequence of length $L = 46$ starting from $\alpha = 000001$ is

$$0000011110011100011011010011000010110010100010.$$

▶ **Theorem 2.** *Let* $\mathbf{S}$ *be a set in* $\mathcal{S}_n(m, h, t)$ *such that if* $n = 2m - 1$ *then it includes the cycle* $[(01)^{m-1}1]$. *Let* $j = \lceil n/2 \rceil$. *If* $s \leq j$, *then complementing the output of* $\mathrm{PCR3}(\gamma_s)$ *in Algorithm 1 produces a cut-down de Bruijn sequence of length* L. *Otherwise, complementing the output of both* $\mathrm{PCR3}(\gamma_j)$ *and* $\mathrm{PCR3}(\gamma_{s-j})$ *in Algorithm 1 produces a cut-down de Bruijn sequence of length* L.

Proof. A complete proof is provided in the Appendix. The basic idea is to trace the substring following a given γ_i in $MC_{\mathbf{S}}$ based on the output of PCR3. Then by complementing the output of

PCR3 for γ_i, it is easy to see that a substring of length i is cut out leaving a cut-down de Bruijn sequence of length $L - i$. The cutting out of two such substrings is possible since they appear on disjoint PCR cycles. ◀

The reason why we cannot cut out a single cycle $[0^{s-1}1]$ where $s > \lceil n/2 \rceil$ is because there exists at least one string $\alpha = a_1 \cdots a_n$ in $\mathbf{Z}_s$ such that $a_2 \cdots a_n 1$ is a necklace and $a_1 = 0$. This means such a string is used to "bridge" a PCR cycle into the MC and thus cutting out this string would also cut the PCR cycle and any of its children from the original MC. As an example, consider $n = 9$ and $s = 6$. The string $\alpha = 000001000 \in \mathbf{Z}_6$ and $\beta = 000010001$ is a necklace and hence PCR3(α) = 1 and thus β belongs to a new PCR cycle with larger weight. It may be possible to re-attach such a cycle to another cycle on the MC, but that involves extra theory and complexity to the algorithm.

Let $\mathbf{R}$ denote the set containing the one or two strings of the form γ_i whose output from PCR3 must be complemented as described in Theorem 2. Then Algorithm 2 will generate a cut-down de Bruijn sequence of length L. It starts with $\alpha = 0^{n-1}1$ in case the cycle $[0]$ needs to be cut out. Computing the weight and period of the current length-n string α leads to an $O(n)$-time membership tester for $\mathbf{S}$. In this implementation, the subset $\mathbf{T}$ of $\mathbf{S}$ is not known a priori. Thus we keep track of how many cycles of weight m and period h we have seen so far, adding them if we do not exceed t. This is maintained by the counter t'. In the special case when $n = 2m - 1$, a flag is set to make sure the special cycle $[(01)^{m-1}1]$ is included; the first string visited on this cycle is $(01)^{m-1}1$. The result is a construction that takes only $O(n)$ time per symbol using only $O(n)$ space.

Algorithm 2 Pseudocode for constructing a binary cut-down de Bruijn sequence assuming pre-computed values m, h, t and the set $\mathbf{R}$

```
1:  procedure CUT-DOWN
2:      α = a_1 a_2 ··· a_n ← 0^{n-1}1
3:      t' ← 0
4:      if n = 2m − 1 then flag ← 1
5:      else flag ← 0

6:      for i ← 1 to L do
7:          PRINT(a_1)

8:          ▷ UC-successor for the Main Cycle
9:          w ← weight of α
10:         x ← PCR3(α)
11:         β = a_2 ··· a_n x
12:         if w = m and w − a_1 + x = m + 1 then x ← x̄
13:         if w = m − 1 and w − a_1 + x = m then
14:             if per(β) > h then x ← x̄
15:             if per(β) = h then
16:                 if β = (01)^{m-1}1 then flag ← 0
17:                 ▷ Cut out excess cycles of weight m and period h
18:                 if t' = t or (t' + 1 = t and flag = 1) then x ← x̄
19:                 else t' ← t' + 1

20:         if β ∈ R then x ← x̄    ▷ Cut out small cycle(s)
21:         α ← a_2 ··· a_n x
```

▶ **Theorem 3.** *A binary cut-down de Bruijn sequence of length $2^{n-1} < L \leq 2^n$ can be generated in $O(n)$ time per symbol using $O(n)$ space.*

5 A successor rule construction of cut-down de Bruijn sequences

In this section we define a successor rule that can be used to construct a cut-down de Bruijn sequence of length L. Unlike the algorithm in the previous section, no context is required when iterating through the successor rule.

Recall that $\mathcal{S}_n(m, h, t)$ contains sets with exactly $L + s$ binary strings of length n based on the pre-computed values m, h, t, s as described in Section 3. Also recall the set $\mathbf{R}$ containing one or two strings. Each set in $\mathcal{S}_n(m, h, t)$ contains a subset $\mathbf{T}$ of exactly ht binary strings of length n, weight m, and period h. The key to our successor rule is to identify these ht strings precisely. In particular, let $\mathbf{T}$ denote the length-n strings found on cycles of the PCR corresponding to the t **largest** Lyndon words of length h and weight mh/n. Choosing the t largest instead of the t smallest cycles is important since it ensures the cycles required in the special cases are always included; each special cycle corresponds to the lexicographically largest Lyndon word for a given weight.

Determining whether or not a string belongs to $\mathbf{T}$ can be determined by applying a recent algorithm to rank fixed-weight Lyndon words as they appear in lexicographic order [12]. Let $\alpha = a_1 a_2 \cdots a_n$ denote a binary string with period n with with weight m. Let σ be the lexicographically smallest rotation of α, which implies σ is a Lyndon word with weight m. Let RL$(a_1 a_2 \cdots a_n)$ denote the rank of σ in a lexicographic listing of length-n Lyndon words with weight m. Computing σ can be done in $O(n)$ time [3] and computing the rank of σ can be done in $O(n^3)$ time based on the unit-cost RAM model [12]. Applying this function, the procedure CUTDOWNSUCCESSOR(α) given in Algorithm 3 will construct a cut-down de Bruijn sequence of length L, which corresponds to a universal cycle for $\mathbf{S} - \mathbf{R}$, starting from any length-n substring α in $\mathbf{S} - \mathbf{R}$.

▶ **Theorem 4.** CUTDOWNSUCCESSOR*(α) generates a cut-down de Bruijn sequence of length L starting from any $\alpha \in \mathbf{S} - \mathbf{R}$ where each symbol is generated in amortized $O(n^{1.5})$-time, based on the unit-cost RAM model, using $O(n)$ space.*

Proof. By pre-computing $L(h, mh/n)$, each iteration of the **for** loop requires $O(n)$ time not including the time required by a possible call to RL. A call to RL is made exactly once for each cycle of weight m and period h; it is made when α has weight $m - 1$ and β has weight m and period h. These cycles correspond to Lyndon words of length h and weight mh/n and thus the call to RL is made exactly $L(h, mh/n)$ times. Clearly, $L(n, w) \leq \binom{n}{w}/n$, and it is well-known that $\binom{n}{w}$ is $O(2^n/\sqrt{n})$ [19, p. 35]. Since the running time of RL is $O(n^3)$, the work done by all calls to RL amortized over the $\theta(2^n)$ iterations of the **for** loop is $O(\frac{n^3}{n\sqrt{n}}) = O(n^{1.5})$. ◀

6 Summary and Future work

Etzion [6] provided the first algorithm to construct a cut-down de Bruijn sequence. We simplified the algorithm and improved the running time by a factor of n, so it requires only $O(n)$ time per symbol using $O(n)$ space. Each construction has a minor downside; they require the context of the sequence generated so far in order to produce successive symbols. By taking advantage of a recent ranking algorithm for fixed weight Lyndon words, we are able to adapt the algorithm so it generates a cut-down de Bruijn sequence via a successor rule; no context is required. The trade-off is that it requires $O(n^{1.5})$-amortized time. It assumes a unit-cost RAM model, and it should be noted that there are mathematical operations on integers.

Algorithm 3 A successor rule based construction of a cut-down de Bruijn sequence of length L based on the precomputed values m, h, t and the set $\mathbf{R}$

```
1:  procedure CutDownSuccessor(α = a_1 a_2 ··· a_n)
2:      for i ← 1 to L do
3:          Print(a_1)
4:          ▷ Context-free UC-successor for the Main Cycle
5:          w ← weight of α
6:          x ← PCR3(α)
7:          β = a_2 ··· a_n x
8:          if w > m or (w = m and per(β) > h) then x ← x̄
9:          if w = m − 1 and w − a_1 + x = m then
10:             if per(β) > h then x ← x̄
11:             if per(β) = h and L(h, mh/n)−RL(a_1 a_2 ··· a_h) +1 > t then x ← x̄
12:         if β ∈ R then x ← x̄    ▷ Cut out small cycle(s)
13:         α ← a_2 ··· a_n x
```

In Etzion's [5] original construction of cut-down de Bruijn sequences, he states that his binary algorithm can be generalized alphabets of arbitrary size. While it is true that the high level ideas follow through to larger alphabets, the details are far from trivial. The primary challenge is that the cycle-joining approach no longer corresponds to a spanning tree of smaller PCR cycles. Furthermore, the simple counting formulae used to compute the variables m, h, t are much more complex. When generalizing the binary approach, the selection of an underlying successor-rule is again critical to a simple construction for a k-ary cut-down de Bruijn sequence. Even the PCR3-successor we applied in the binary case has two natural generalizations for $k \geq 2$; they have been previously labeled PCR3 and PCR3-alt [9] and both correspond to the binary PCR3 described in this paper when $k = 2$. The key in selecting a de Bruijn successor is to allow for simplest possible way to cut out small cycles. Without going into a deep analysis, the PCR3-alt turns out to be the one that allows for this. An algorithm to construct a k-ary cut-down de Bruijn sequence of length $k^{n-1} < L \leq k^n$ is available at `http://debruijnsequence.org/cutdown` [1]. It generates each successive symbol in $O(n)$-time using $O(n)$ space. In future work, we will provide the algorithm details and analysis.

When generalizing to $k > 2$, there is one benefit: the handling of **Special case #2** is longer required. However, since there is no known ranking algorithm for k-ary Lyndon words with fixed-content, the approach used in Section 5 for finding a successor rule for the resulting de Bruijn sequence can not currently be applied efficiently.

References

1 DB. De Bruijn sequence and universal cycle constructions (2021). http://debruijnsequence.org.

2 AGUIRRE, G., MATTAR, M., AND WEINBERG, L. De Bruijn cycles for neural decoding. *NeuroImage 56* (02 2011), 1293–300.

3 BOOTH, K. S. Lexicographically least circular substrings. *Inform. Process. Lett. 10*, 4/5 (1980), 240–242.

4 DIACONIS, P., AND GRAHAM, R. *Magical Mathematics: The Mathematical Ideas That Animate Great Magic Tricks*. Princeton University Press, 2011.

5 ETZION, T. An algorithm for generating shift-register cycles. *Theor. Comput. Sci. 44* (1986), 209–224.

6 ETZION, T. Self-dual sequences. *J. Comb. Theory Ser. A 44*, 2 (Mar. 1987), 288–298.

7 GABRIC, D., HOLUB, S., AND SHALLIT, J. O. Generalized de bruijn words and the state complexity of conjugate sets. *CoRR abs/1903.05442* (2019).

8 GABRIC, D., SAWADA, J., WILLIAMS, A., AND WONG, D. A framework for constructing de Bruijn sequences via simple successor rules. *Discrete Mathematics 241*, 11 (2018), 2977–2987.

9 GABRIC, D., SAWADA, J., WILLIAMS, A., AND WONG, D. A successor rule framework for constructing k-ary de Bruijn sequences and universal cycles. *IEEE Transactions on Information Theory 66*, 1 (2020), 679–687.

10 GILBERT, E. N., AND RIORDAN, J. Symmetry types of periodic sequences. *Illinois J. Math. 5*, 4 (1961), 657 – 665.

11 GOLOMB, S. W. *Shift Register Sequences*. World Scientific, Singapore, 2017.

12 HARTMAN, P., AND SAWADA, J. Ranking and unranking fixed-density necklaces and Lyndon words. *Theoretical Computer Science 791* (2019), 36–47.

13 HEMMATI, F., AND COSTELLO, D. J. An algebraic construction for q-ary shift register sequences. *IEEE Transactions on Computers C-27*, 12 (1978), 1192–1195.

14 HIERHOLZER, C. Ueber die Möglichkeit, einen Linienzug ohne Wiederholung und ohne Unterbrechung zu umfahren. *Mathematische Annalen 6* (1873), 30–32.

15 JANSEN, C. J. A., FRANX, W. G., AND BOEKEE, D. E. An efficient algorithm for the generation of DeBruijn cycles. *IEEE Transactions on Information Theory 37*, 5 (Sep 1991), 1475–1478.

16 LANDSBERG, M. Feedback functions for generating cycles over a finite alphabet. *Discrete Mathematics 219*, 1 (2000), 187–194.

17 LEMPEL, A. On a homomorphism of the de Bruijn graph and its applications to the design of feedback shift registers. *IEEE Transactions on Computers C-19*, 12 (Dec 1970), 1204–1209.

18 LEMPEL, A. m-ary closed sequences. *J. Combin. Theory Ser. A 10*, 3 (1971), 253–258.

19 LUKE, Y. L. *The special functions and their approximations, Vol. I.* Mathematics in Science and Engineering, Vol. 53. Academic Press, New York-London, 1969.

20 NELLORE, A., AND WARD, R. Arbitrary-length analogs to de Bruijn sequences, arXiv:2108.07759, 2021.

21 PEVZNER, P. A., TANG, H., AND WATERMAN, M. S. An Eulerian path approach to DNA fragment assembly. *Proceedings of the National Academy of Sciences 98*, 17 (2001), 9748–9753.

22 SAWADA, J., WILLIAMS, A., AND WONG, D. A surprisingly simple de Bruijn sequence construction. *Discrete Math. 339*, 1 (2016), 127–131.

23 SCHEINERMAN, E. R. Determining planar location via complement-free de Brujin sequences using discrete optical sensors. *IEEE Trans. Robotics Autom. 17* (2001), 883–889.

24 SINDEN, F. W. Sliding window codes. *AT&T Bell Labs Tech. Memorandum* (1985).

25 YOELI, M. Binary ring sequences. *The American Mathematical Monthly 69*, 9 (1962), 852–855.

A Proof of Theorem 2

By the restriction on $\mathbf{S}$, the PCR cycles containing the strings in each $\mathbf{Z}_i$ are included in $MC_{\mathbf{S}}$. The case for cutting out the $[0]$ cycle by complementing the PCR3 output for γ_1 is straightforward. For $2 \leq i \leq \lceil n/2 \rceil$, consider the string γ_i in $MC_{\mathbf{S}}$ and the i symbols following it by tracing Algorithm 2 and observing the output for PCR3. We consider two cases depending on whether or not i divides n. When i divides n, the next i symbols are precisely 10^{i-1}. This is easily determined by the definition of a necklace and PCR3. For example when $n = 8$ and $i = 4$ we have $\gamma_i = 00001000$. The $i + 1$ symbols following γ_i in $MC_{\mathbf{S}}$ are given in the final column.

$a_1 \cdots a_8$	$\beta = a_2 \cdots a_8 1$	PCR3(β)
00001000	00010001	**1**
00010001	00100011	0
00100010	01000101	0
01000100	10001001	0
10001000	00010001	**0**

Note the last four strings in the first column are precisely the strings in $\mathbf{Z}_4$ belonging to the PCR cycle Z_4. This demonstrates that such cycles Z_i are not the parents of other PCR cycles in the joining of the cycles to create the MC. Also, the first and last strings in the table share a length $n - 1$ suffix; they are conjugate pairs. Thus complementing the output for γ_i effectively cuts out a single cycle of length i from $MC_{\mathbf{S}}$.

When i does not divide d, the next i symbols again are precisely 10^{i-1}. However, now the strings of $\mathbf{Z}_i$ belong to two different PCR cycles that are joined together in the MC. By tracing the i symbols following γ_i in $MC_{\mathbf{S}}$ as we did in the previous case, it is easy to observe based on the definition of PCR3 that the strings in $\mathbf{Z}_i$ appear consecutively as substrings. For example when $n = 11$ and $i = 4$ we have $\gamma_i = 10010001000$.

$a_1 \cdots a_{11}$	$\beta = a_2 \cdots a_{11} 1$	PCR3(β)
10010001000	00100010001	**1**
00100010001	01000100011	0
01000100010	10001000101	0
10001000100	00010001001	0
00010001000	00100010001	**0**

Note the last four strings in the first column are precisely the strings in $\mathbf{Z}_4$. This demonstrates that the strings in $\mathbf{Z}_i$ are not the parents of other PCR cycles in the joining of the cycles to create the MC. Also, the first and last strings share a length $n - 1$ suffix; they are conjugate pairs. Thus, complementing the output for γ_i effectively cuts out a single cycle of length i from $MC_{\mathbf{S}}$.

Though the observations made above are for specific examples, they easily adapt for general i and n, and we leave it as an exercise for the reader. As a result of these observations, if $s \leq j$, we obtain the theorem result directly. Otherwise, observe that the strings of $\mathbf{Z}_j$ and $\mathbf{Z}_{s-j}$ belong to disjoint PCR cycles. Thus complementing the output of both PCR3(γ_j) and PCR3(γ_{s-j}) in Algorithm 1 produces a cut-down de Bruijn sequence of length L.