Cut-Down de Bruijn Sequences

Ben Cameron
University of Prince Edward Island, 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 L, where $1 \leq L \leq k^n$, such that every substring of length n appears *at most* once. Etzion [*Theor. Comp. Sci* 44 (1986)] introduced an algorithm to construct binary cut-down de Bruijn sequences requiring $o(n)$ simple n-bit operations per symbol generated. In this paper, we simplify the algorithm and improve the running time to $\mathcal{O}(n)$ time per symbol generated using $\mathcal{O}(n)$ space. Additionally, we develop the first successor-rule approach for constructing a binary cut-down de Bruijn sequence by leveraging recent ranking/unranking algorithms for fixed-density Lyndon words. Finally, we develop an algorithm to generate cut-down de Bruijn sequences for $k > 2$ that runs in $\mathcal{O}(n)$ time per symbol using $\mathcal{O}(n)$ space after some initialization.

1 Introduction

A *de Bruijn sequence* (DB sequence) of span n, over an alphabet of size k, 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 DB sequence for $n = 6$ and $k = 2$:

$$[\,0000001111110111100111000110110100110000101110101100101010001001\,]. \qquad (1)$$

The *de Bruijn graph* of span n, over an alphabet of size k, is the directed graph $G(n,k) = (V,E)$ where V is the set of all k-ary strings of length n and there is a directed edge $e = (u,v) \in E$ from $u = u_1u_2\cdots u_n$ to $v = v_1v_2\cdots v_n$ if $u_2\cdots u_n = v_1\cdots v_{n-1}$. Each edge e is labeled by v_n. In this paper, the term *cycle* corresponds to a sequence of edge labels obtained by traversing some cycle/circuit in the de Bruijn graph (or a related edge-labeled graph), and the notation $[\alpha]$ denotes that the sequence α is cyclic. For example, Figure 1 illustrates $G(3,2)$ and the cycles $[01]$ and $[1101100001]$. It is well known that a DB sequence of span n is in one-to-one correspondence with an Euler cycle in $G(n{-}1,k)$.

For some applications it may be more convenient to produce a cycle of arbitrary length such that there are no repeated length-n substrings, i.e., a cycle of arbitrary length in $G(n-1,k)$. For instance, it may be more natural to consider the de Bruijn card trick [7] using 52 cards rather than 32. Also, for applications in robotic vision and location detection [7, 30, 31], instead of forcing a location map to have length k^n, an arbitrary length allows for more flexibility. This gives rise to the notion of a *cut-down de Bruijn sequence*[1] (cut-down DB sequence), which is a cyclic sequence of length L over an alphabet of size k, where $1 \leq L \leq k^n$, such that every substring of length n appears *at most* once.

[1] The term *cutting-down* is perhaps first used in [19] to describe such sequences.

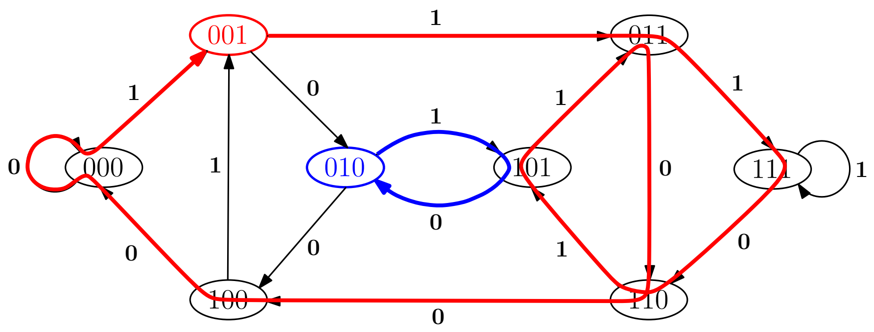


Figure 1 The de Bruijn graph $G(3, 2)$ highlighting cut-down DB sequences $[10]$ (blue) and $[1101100001]$ (red) of length two and ten, respectively.

As an example, the following is a binary cut-down DB sequence of length 52:

$$[\,0000001111001110001101101001100001011101011001010001\,].$$

Note that every substring of length $n = 6$, including in the wraparound, appears at most once. Any cut-down DB sequence of length L with respect to k and n is also a cut-down DB sequence with respect to k and $n + 1$ [9]. Thus, throughout this paper we assume $k^{n-1} < L \leq k^n$.

Cut-down DB sequences are known to exist for all lengths L and any alphabet of size k [23] (for k=2 see [32]). 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 DB sequences and their relatives [2, 26]. A simple algorithm to construct binary cut-down DB sequences based on linear feedback shift registers and primitive polynomials is given by Golomb [15, P. 193]; it runs in $\mathcal{O}(n)$-amortized time per symbol using $\mathcal{O}(n)$ 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 DB sequences where k is a prime power, and subsequently extended to handle arbitrary sized alphabets by applying additional number theoretic results [21]. 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 [17].

A cycle-joining based approach to construct cut-down DB sequences was developed by Etzion [9] for $k = 2$; it requires $o(n)$ simple n-bit operations to generate each symbol. The approach follows two main steps:

- First, an initial cycle is constructed with length $L + s$, where $0 \leq s < n$, using the well-known cycle-joining approach.
- Second, depending on s, up to $\lceil \log n \rceil$ small cycles are detected and removed to obtain a cycle of length L.

The resulting algorithm can construct an exponential number of cut-down DB sequences for any given L; however, their algorithm is not optimized to generate a single cut-down DB sequence. Etzion's construction also has a downside for some applications: It starts with a specific length-n string and the historical context matters to produce successive symbols. This means testing whether or not an arbitrary 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 DB sequence in $\mathcal{O}(n)$ time per symbol using $\mathcal{O}(n)$ space.

2. We develop the first successor-rule approach to construct a binary cut-down DB sequence in $\mathcal{O}(n)$ time per symbol using $\mathcal{O}(n)$ space. The algorithm can start with any string on the cycle and the context does not matter when producing successive symbols. Determining whether or not a length-n string appears as a substring on the sequence can be determined using $\mathcal{O}(n^3)$ operations on n-bit numbers.
3. We develop an algorithm to generate cut-down DB sequences for $k > 2$ that runs in $\mathcal{O}(n)$ time per symbol using $\mathcal{O}(n)$ space. A number of non-trivial adaptations to the binary algorithm are required to generalize to larger alphabets.

All three algorithms require a polynomial time and space initialization step.

Related work. A *generalized de Bruijn sequence*, as defined in [4], is a cut-down DB sequence of length $k^{n-1} < L \leq k^n$ with an additional property: *every k-ary string of length $n-1$ appears as a substring*. Their existence is known for all L and k [11]. For special values of L, these sequences can be generated by considering the base k expansion of $1/L$ [4, 5, 24]. An algorithm based on Lempel's D-morphism [22] has recently been proposed to construct these sequences [25] 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*. We call sequences with this latter property *balanced cut-down de Bruijn sequences*. The proposed algorithm can generate the sequences in $\mathcal{O}(1)$-amortized time per symbol, but it requires exponential space and there is an exponential time delay before outputting the first symbol.

Repeat-free sequences have all of the properties of cut-down DB sequences except the cyclic property; they are prefixes of a DB sequence and correspond to paths in the de Bruijn graph. They were considered from an algorithmic perspective in [3, 8] and discussed from a combinatorial perspective under the name *partial de Bruijn ℓ-sequences* in [6].

Outline of paper. In Section 2, we provide some background on the cycle-joining method and a simple successor rule to construct DB sequences. In Section 3, we review Etzion's approach for constructing binary cut-down DB sequences. In Section 4, we present in detail our simplified algorithm to construct binary cut-down DB sequences; in Section 4.3, we present the first successor-rule algorithm for constructing binary cut-down DB sequences. In Section 5 we extend our binary algorithm to work for $k > 2$. Implementation of our algorithms are available for download at `http://debruijnsequence.org/db/cutdown` [1]; this resource also provides a comprehensive background on DB sequences and their constructions.

2 Background

Let Σ denote the alphabet $\{0, 1, \ldots, k-1\}$ where $k \geq 2$. 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 $\text{per}(\alpha)$, 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* of a string is the sum of its elements (when $k = 2$, weight is sometimes referred to as *density*). Let $T_k(n, w)$ denote the number of k-ary strings of length n and weight w. Note $T_2(n, w) = \binom{n}{w}$. Let $L_k(n, w)$ denote the number of k-ary Lyndon words of length n and weight w. By partitioning the strings of $T_k(n, w)$ into equivalence classes under rotation and considering the period of the string in each class (see Example 1) observe that $T_k(n, w) = \sum_{d|n} \frac{n}{d} L(\frac{n}{d}, \frac{w}{d})$. By applying Mobiüs inversion we have:

$$L_k(n, w) = \frac{1}{n} \sum_{d|\,\gcd(n,w)} \mu(d)\, T_k\left(\frac{n}{d}, \frac{w}{d}\right),$$

where μ is the Mobiüs function. When $k = 2$, the formula is derived in [14] and applied in [16].

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 $F(\alpha) = a_2a_3 \cdots a_n f(\alpha)$, 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 equivalence classes of strings under rotation. Thus, the cycles induced by the PCR, called *PCR cycles*, are in one-to-one correspondence with the necklaces of order n and also with Lyndon words whose lengths divide n. They also appear as cycles in $G(n-1, k)$. Recall, we use the notation $[\alpha]$ to denote a cycle. When n is understood, we say a cycle *contains* a string α if α has length n and is found as a substring on the cycle (by possibly traversing around the cycle more than once); we say α *belongs* to the cycle.[2] For example, when $n = 6$, the PCR cycle [01] (see the upcoming Example 1) contains the two strings 010101 and 101010; the cycle [0001] (not a PCR cycle) contains the four strings 000100, 001000, 010001, and 100010.

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.

000000	**0**00001	**0**00011	**0**00101	**0**00111	**0**01001	**0**01011
	000010	**0**00110	**0**01010	**0**01110	**0**10010	**0**10110
	000100	**0**01100	**0**10100	**0**11100	**1**00100	**1**01100
	001000	**0**11000	**1**01000	**1**11000		**0**11001
	010000	**1**10000	**0**10001	**1**10001		**1**10010
	100000	**1**00001	**1**00010	**1**00011		**1**00101

001101	**0**01111	**0**10101	**0**10111	**0**11011	**0**11111	**1**11111
011010	**0**11110	**1**01010	**1**01110	**1**10110	**1**11110	
110100	**1**11100		**0**11101	**1**01101	**1**11101	
101001	**1**11001		**1**11010		**1**11011	
010011	**1**10011		**1**10101		**1**10111	
100110	**1**00111		**1**01011		**1**01111	

The following 14 PCR cycles are in one-to-one correspondence with the set of Lyndon words of lengths 1, 2, 3, and 6 (lengths that divide $n = 6$):

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

Note the strings belonging to a PCR cycle all have the same weight. Thus, let the weight of a PCR cycle be the weight of its corresponding length n necklace. For example, when $n = 6$ the weight of $[000001]$ is one and the weight of $[01]$ is three since its corresponding necklace is 010101.

A *universal cycle* for a set $\mathbf{S}$ of length-n strings is a cyclic sequence of length $|\mathbf{S}|$ such that every string in $\mathbf{S}$ appears as a substring exactly once. Two universal cycles for $\mathbf{S_1}$ and $\mathbf{S_2}$ are said to be *disjoint* if the sets $\mathbf{S_1}$ and $\mathbf{S_2}$ are disjoint. Of course, a DB sequence is a special case of a universal cycle when $\mathbf{S}$ corresponds to all k-ary strings of length n. A cut-down DB sequence 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 $\mathbf{S}$ starting from any string in $\mathbf{S}$ (see the

[2] Consider the infinite word α^t; it *contains* the length-n word β if β appears as a substring (factor) of α^t.

upcoming Algorithm 1 for a specific example). When $\mathbf{S} = \Sigma^n$ a UC-successor is said to be a *de Bruijn-successor*.

2.1 Cycle joining

One of the most common ways to construct a DB sequence is by applying the *cycle-joining method* [15], which is akin to Hierholzer's method for finding Euler cycles in graphs [18]. This approach repeatedly joins pairs of disjoint (universal) cycles that share a node $v = a_2 \cdots a_n$ in $G(n-1,k)$. The two cycles are said to be joined via a *conjugate pair* $(xa_2 \cdots a_n, ya_2 \cdots a_n)$, where x and y correspond to the labels of incoming edges to v for each cycle. This leads to the following lemma which is implicitly applied in all cycle-joining constructions and formalized for $k = 2$ in [28].

▶ **Lemma 1.** *Let $\mathbf{S}_1$ and $\mathbf{S}_2$ be disjoint subsets of Σ^n such that $xa_2 \cdots a_n \in \mathbf{S}_1$ and $ya_2 \cdots a_n \in \mathbf{S}_2$; $(xa_2 \cdots a_n, ya_2 \cdots a_n)$ is a conjugate pair. If U_1 is a universal cycle for $\mathbf{S}_1$ and U_2 is a universal cycle for $\mathbf{S}_2$, each with prefix $a_2 \cdots a_n$, then $U = U_1U_2$ is a universal cycle for $\mathbf{S}_1 \cup \mathbf{S}_2$.*

When the initial cycles are those induced by an underlying nonsingular FSR, the joining of the cycles can be viewed as a tree in the binary case. As an example, Figure 2 illustrates the PCR cycles for $n = 6$ and $k = 2$, and one way they can be joined together to create a DB sequence, or equivalently, an Euler cycle in $G(5,2)$. When extending this idea to larger alphabet sizes, the tree visualization no longer applies in general (see [13] and the upcoming Figure 3).

A general framework based on the cycle-joining approach leads to many simple UC-successors [12]. Application of this framework rediscovers many previously known DB sequence constructions including one by Jansen [20] that was revisited in [29] with respect to the PCR. This particular construction starts with $[0]$ as the root cycle and repeatedly joins PCR cycles by increasing weight via conjugate pairs $(1a_2 \cdots a_n, 0a_2 \cdots a_n)$, where $a_2 \cdots a_n1$ is the necklace representative of the new PCR cycle begin joined. This construction is illustrated in Figure 2, where the symbol pointed to by a downward edge is the "last symbol" in the corresponding cycle's necklace representative. Thus, given a necklace $\alpha = a_1 \cdots a_n \neq 0^n$, the parent cycle of $[\alpha]$ in the "cycle-joining tree" rooted at $[0]$ is $[a_1 \cdots a_{n-1}0]$; the conjugate pair that joins them is $(0a_1 \cdots a_{n-1}, 1a_1 \cdots a_{n-1})$. The resulting de Bruijn successor (below), labeled PCR3 in [12], is perhaps the simplest of all de Bruijn successors. Note $\overline{x}$ denotes the complement of the bit x.

PCR3 de Bruijn successor:

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

When the FSR with feedback function PCR3 is repeatedly applied to the starting string 000000 for $n = 6$, as illustrated in Figure 2, it produces the DB sequence in (1), where the first bit of the current string α is output before each application of the rule (see the upcoming Algorithm 1). The arcs between cycles in Figure 2 correspond to the cases when PCR3 returns $\overline{a}_1$.

Of course, the cycle-joining process yielding PCR3 can be applied to any subset of PCR cycles as long as they are "connected" via the defined conjugate pairs, i.e., the PCR cycles form a subtree of the complete cycle joining tree induced by PCR3[3]. Given such a subtree, let $\mathbf{S}$ denote the set of all length-n strings that belong to some PCR cycle in the subtree.

[3] See a related discussion in [27].

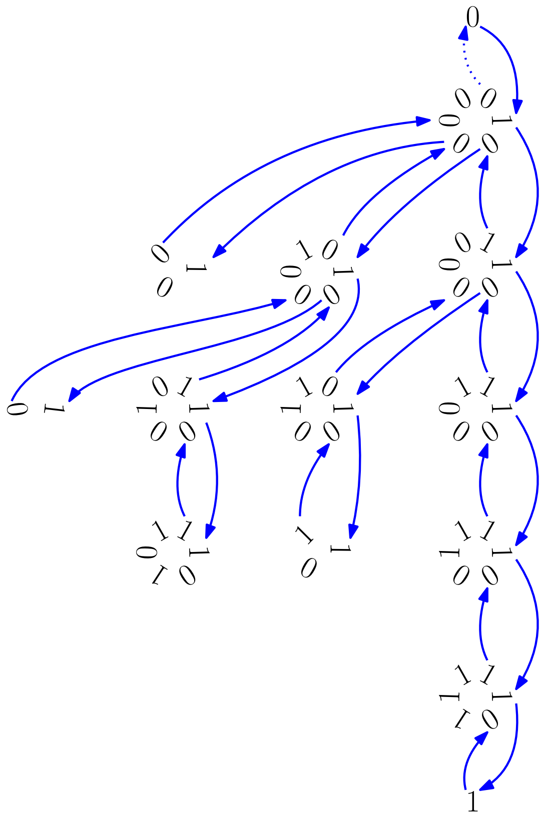


Figure 2 The 14 PCR cycles for $n = 6$ joined by applying the de Bruijn successor PCR3. Starting with the cycle [0] at the top, the cycles are visited in a clockwise fashion until a blue arc transitions to a different PCR cycle. Each blue arc leaves from a string (corresponding to the previous n-symbols) belonging to a conjugate pair used to join two cycles. For instance, the top two cycles [0] and [000001] are joined via the conjugate pair (000000, 100000). Tracing through the cycles and returning to [0] produces a DB sequence for $n = 6$.

Example 2 Consider the cycle-joining tree in Figure 2 based on PCR3 for $n = 6$. Consider the subtree consisting of the cycles [0], [000001], [001]. The set $\mathbf{S}$ corresponding to this subtree is

$$\mathbf{S} = \{000000\} \cup \{000001, 000010, 000100, 001000, 010000, 100000\} \cup \{001001, 010010, 100100\}.$$

Let $\mathcal{S}$ denote the collection of all sets $\mathbf{S}$, ranging over all unique subtrees. In particular, we will be interested in a set $\mathbf{S} \in \mathcal{S}$ obtained from the PCR cycles with weight less than some $m > 0$ together with a subset of PCR cycles with weight m (see Example 3). A UC-successor for any $\mathbf{S} \in \mathcal{S}$ can be obtained from the PCR3 de Bruijn successor by additionally ensuring that PCR3$(a_1 a_2 \cdots a_n)$ maps to $\overline{a}_1$ only if $a_2 \cdots a_n \overline{a}_1$ is in $\mathbf{S}$, i.e., it does not attempt to join a cycle outside the specific subset.

PCR3 UC successor for $\mathbf{S} \in \mathcal{S}$:

$$\text{PCR3}'(\alpha) = \begin{cases} \overline{a}_1 & \text{if } a_2 a_3 \cdots a_n 1 \text{ is a necklace and } a_2 a_3 \cdots a_n \overline{a}_1 \in \mathbf{S}; \\ a_1 & \text{otherwise.} \end{cases}$$

Observe that PCR3 is just a special case of PCR3′ when $\mathbf{S}$ is the set of all binary strings of length n. Note that if $a_1 \cdots a_n$ is in $\mathbf{S}$, then so is $a_2 \cdots a_n a_1$ since they belong to the same PCR cycle. Starting with any string $\alpha \in \mathbf{S}$, Algorithm 1 applies this UC-successor to construct a universal cycle for $\mathbf{S}$, applying the original definition for PCR3. Later, we will specify further implementation details for a specific $\mathbf{S}$.

Algorithm 1 Pseudocode for constructing a universal cycle for $\mathbf{S} \in \mathcal{S}$ assuming $\alpha = a_1a_2\cdots a_n \in \mathbf{S}$.

```
procedure UC($\alpha$)
    for $i \leftarrow 1$ to $|\mathbf{S}|$ do
        PRINT($a_1$)
        $x \leftarrow$ PCR3($\alpha$)
        $\beta \leftarrow a_2\cdots a_n x$
        if $\beta \notin \mathbf{S}$ then $x \leftarrow \overline{x}$
        $\alpha \leftarrow a_2\cdots a_n x$
```

▶ **Lemma 2.** *Algorithm 1 generates a universal cycle for* $\mathbf{S}$*, where* $\mathbf{S} \in \mathcal{S}$.

This DB sequence constructed by PCR3 has an important property not shared by the other simple feedback functions presented in [12]: The strings belonging to $\mathbf{Z}_i$ (defined in Section 4.2) appear contiguously as substrings in the corresponding DB sequence.

3 Etzion's approach

In this section, we outline Etzion's [9] approach for constructing a binary cut-down DB sequence. Recall that L is the length of the cut-down DB sequence and $2^{n-1} < L \leq 2^n$. The two primary steps in Etzion's construction are as follows, where the *surplus* s is an integer in $\{0, 1, \ldots, n{-}1\}$:

1. Construct a Main Cycle (MC) that has length $L + s$.
2. Cut out up to $\lceil \log s \rceil$ small cycles from the MC to yield a cycle of the desired length L.

It is important to note that the small cycles being cut out of the MC, are *not necessarily* PCR cycles. To construct an MC, a subset of the PCR cycles are selected based on their weight and period. Enumeration of strings by weight and period determine which cycles to include. Considering the set of k-ary strings (so we can generalize in later sections) of length n , let

- $A(w)$ denote the number of strings with weight $\leq w$,
- $B(w, p)$ denote the number of strings with weight w, and period p, and
- $C(w, p)$ denote the number of strings with weight w, and period $\leq p$.

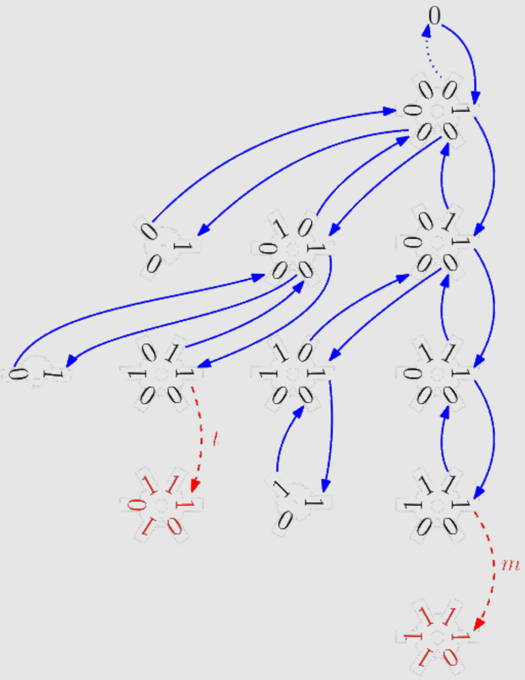
In the binary case when $k = 2$, clearly $A(w) = \sum_{j=0}^{w} \binom{n}{j}$. Recall from the observations in Example 1 that $B(w, p) = pL_k(p, wp/n)$, assuming p divides n. Using these values, let

- m = the smallest weight m such that $A(m) \geq L$,
- h = the smallest period such that $A(m-1) + C(m, h) \geq L$, and
- t = the smallest integer such that $A(m-1) + C(m, h-1) + th \geq L$.

These values can be used to define the surplus s as $A(m-1) + C(m, h-1) + ht - L$.

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. 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 specific t PCR cycles joined are not necessarily known *a priori*. 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.

Example 3 Consider $L = 46$ and $n = 6$. Since $A(3) = 42$ and $A(4) = 57$, we have $m = 4$. Since $B(4,1) = B(4,2) = 0$, $B(4,3) = 3$, $B(4,4) = B(4,5) = 0$, $B(4,6) = 12$, we have $C(4,5) = 3$ and $C(4,6) = 15$. Thus $h = 6$. Since $A(3) + C(4,5) + h = 42 + 3 + 6 = 51$, 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 with length 51 is

$$[\,000000111100111000110110100110000101100101010001001\,].$$

The second step involves *cutting out* small cycles whose combined length totals the surplus s. When $s = 1$, the cycle $[0]$ is easily removed. However, for arbitrary lengths, finding and removing such small cycles is non-trivial when the cycles are not PCR cycles. The construction of the MC is critical for the ease in which these small cycles can be removed. Etzion's approach requires cutting out up to $\lceil \log s \rceil$ cycles of the form $[0^{i-1}1]$ where $i < s$ is a power of 2. For example, when $s = 14$, 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 Section 4.2. 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 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 earlier, 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 DB sequences

In this section we apply the following two enhancements to simplify and improve Etzion's original approach for constructing a cut-down DB sequence.

1. We apply PCR3 and focus on a single cut-down DB sequence construction.
2. We consider all cycles of the form $[0^{i-1}1]$ for $1 \leq i \leq \lceil n/2 \rceil$ and cut out at most two small cycles.

The latter step involves changing the definition of PCR3 for at most two strings. As noted in the previous section, when defining an MC, the cycles of weight m will be added by increasing (instead of decreasing) period, thus handling **Special case #1**. To account for **Special case #2**, we assume that $\mathbf{S}$ contains all strings from the cycle $[(01)^{m-1}1]$ when $n = 2m-1$. Details for ensuring this are provided in the upcoming Algorithm 3.

We conclude this section by applying a ranking algorithm for fixed-weight Lyndon words to produce the first successor-rule construction of cut-down DB sequences.

4.1 Constructing a Main Cycle with PCR3′

Recall that the substrings of length n from an MC of length $L+s$ correspond to the set $\mathbf{S}$ of

- all length-n binary strings belonging to PCR cycles with weight less than m,
- all length-n binary strings belonging to PCR cycles with weight m and period less than h, and
- all length-n binary strings from t PCR cycles with weight m and period h,

where m, h, t, s are the precomputed variables described in the previous section. Thus, Algorithm 1 can be applied to construct a universal cycle for $\mathbf{S}$.

For the remainder of this section, let $MC_{\mathbf{S}}$ denote the universal cycle for $\mathbf{S}$ obtained from Algorithm 1.

4.2 Cutting out small cycles

In order to cut down $MC_{\mathbf{S}}$ to a cycle of length L, ideally we 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 totals s.

When $s = 1$, it is straightforward to cut out the cycle $Z_1 = [0]$ by cutting the unique substring $z(1) = 0^n$ down to 0^{n-1}. Otherwise, consider cycles of the form $Z_i = [0^{i-1}1]$ for $1 < i \leq \lceil \frac{n}{2} \rceil$. Etzion [9] considers similar cycles, but only those with length that is a power of two. Let $\mathbf{Z}_i$ denote the set of i length-n strings belonging to Z_i. When i divides n, the strings in $\mathbf{Z}_i$ belong to a single PCR cycle. However, if i does not divide n, then the strings in $\mathbf{Z}_i$ belong to two PCR cycles with different weights. We cannot cut out a cycle $[0^{s-1}1]$ when $\lceil n/2 \rceil < s < n$ because the length-n strings belonging to Z_i do not appear contiguously on $MC_{\mathbf{S}}$.

▶ **Lemma 3.** *Let $\lceil n/2 \rceil < s < n$. The length-$n$ strings belonging to $[0^{s-1}1]$ do not appear contiguously on $MC_{\mathbf{S}}$.*

Proof. Let $F(a_1a_2\cdots a_n) = a_2\cdots a_nx$ where $x = \text{PCR3}'(a_1a_2\cdots a_n)$. Consider two length-$n$ strings $\alpha = 0^{s-1}10^{n-s}$ and $\beta = 0^{n-s}10^{s-1}$ belonging to $[0^{s-1}1]$. By the definition of s, $s{-}1 > n{-}s$. Thus, $0^{s-2}10^{n-s}1$ is a necklace, but $0^{n-s-1}10^{s-1}1$ is not. Therefore, neither $F(\alpha) = 0^{s-2}10^{n-s}1$, nor $F(\beta) = 0^{n-s-1}10^{s}$ belongs to $[0^{s-1}1]$. This implies the length-n strings belonging to $[0^{s-1}1]$ do not appear contiguously on $MC_{\mathbf{S}}$. ◀

For $1 < i \leq \lceil \frac{n}{2} \rceil$, let $w(i) = 0^b1(0^{i-1}1)^{a-1}0^{i-1} = w_1w_2\cdots w_{n-i+1}$, where $a = \lceil \frac{n}{i} \rceil$ and $b = n + i - 1 - ai$. Observe, that $w(i)$ contains each string in $\mathbf{Z}_i$ as a substring.

Example 4 Let $n = 10$ and consider $\mathbf{Z}_3 = \{1001001001, 0010010010, 0100100100\}$. Observe that $w(3) = 100100100100$ (noting $a = 4$ and $b = 0$) has length 12; it contains each string in $\mathbf{Z}_3$ as a substring.

The upcoming lemma implies that the i strings in $\mathbf{Z}_i$ appear contiguously as substrings on $MC_{\mathbf{S}}$, with the string $z(i) = 0^b1(0^{i-1}1)^{a-1}$ appearing first as the length-n prefix of $w(i)$.

▶ Remark 4. A string $0^x1u0^{x+1}v$ is **not** a necklace for any integer x and binary strings u, v.

▶ **Lemma 5.** *$w(i)$ is a substring of $MC_{\mathbf{S}}$ (considered cyclically), for $1 < i \leq \lceil \frac{n}{2} \rceil$.*

Proof. For $n \geq 2$, the strings in $\mathbf{Z}_i$ have weight at most $\lceil \frac{n}{2} \rceil$. This upper limit is obtained only when $i = 2$ for $Z_2 = [01]$. Thus, the strings in $\mathbf{Z}_i$ always appear as substrings in $MC_{\mathbf{S}}$ since we already accounted for the two special cases defined at the end of Section 3. Let $w_j(i)$ denote the length-n substring of $w(i)$ starting at index j, for $1 \leq j \leq i$. We show that $\text{PCR3}'(w_j(i)) = w_{j+n} = 0$, for $1 \leq j < i$. If i divides n, then $b = i{-}1$ and by Remark 4, $\text{PCR3}'(w_j(i)) = w_j = 0$. Suppose i does not divide n. If $j \neq b{+}1$, then $w_j = 0$ and by Remark 4, $\text{PCR3}'(w_j(i)) = w_j = 0$. If $j = b+1$, then $w_j = 1$ and $w_{j+1}\cdots w_{j+n-1}1 = (0^{i-1}1)^{a-1}0^b1$ is a necklace. Thus, $\text{PCR3}'(w_j(i)) = \overline{w_j} = 0$. ◀

Observe that the prefix $w_1\cdots w_{n-1}$ is the same as length-$(n{-}1)$ suffix of $w(i)$. Let w_0 denote the bit that appears before the prefix $w_1\cdots w_{n-1}$ in $MC_{\mathbf{S}}$; $\text{PCR3}'(w_0w_1\cdots w_{n-1}) = w_n$. Thus, applying the above lemma, we can modify Algorithm 1 to cut out the cycle Z_i by checking if β is $z(i)$ after Line 6, and if it is, complement the bit x, which effectively continues the cycle as after $w(i)$. When considering the de Bruijn graph $G(n, 2)$, this corresponds to cutting out the cycle Z_i that starts and ends at the vertex with label $w_1\cdots w_{n-1}$. The resulting Algorithm 2 is obtained by making the following three modifications to Algorithm 1 for some $1 \leq i \leq \lceil \frac{n}{2} \rceil$:

1. the initial string α is in $\mathbf{S} \setminus \mathbf{Z}_i$ (i.e, it is not one of the strings being cut out),
2. the value for x is complemented after Line 6 if $\beta = z(i)$ (cutting out the cycle Z_i), and
3. the **for** loop iterates i less times (to account for the cycle being cut out).

▶ **Lemma 6.** *Algorithm 2 constructs a cut-down DB sequence of length $L + s - i$.*

Let $j = \lceil n/2 \rceil$. If $s \leq j$, then let $\mathbf{R} = \{z(s)\}$; otherwise let $\mathbf{R} = \{z(j), z(s{-}j)\}$. Since the strings in each $\mathbf{Z}_i$ are distinct, we can modify Line 7 of Algorithm 2 to test if $\beta \in \mathbf{R}$ to remove either one or two small cycles; it follows from Lemma 6 that the resulting algorithm will generate a cut-down DB sequence of length L. Algorithm 3 applies this modification along with implementation details required to efficiently test if a string belongs to $\mathbf{S}$. It starts with $\alpha = 0^{n-1}1$, which is a string that does not belong to any $\mathbf{Z}_i$. Computing the weight and period of the current length-n string α leads to an $\mathcal{O}(n)$-time membership tester for $\mathbf{S}$. Since the length-n strings belonging to the t PCR cycles with weight m and period h are not known *a priori*, 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

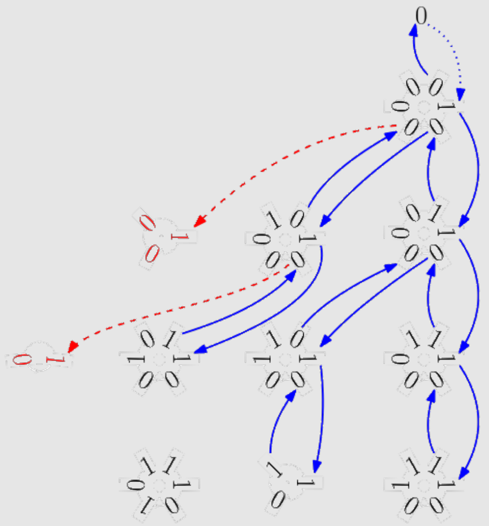
Algorithm 2 Pseudocode for constructing a universal cycle for $\mathbf{S} \setminus \mathbf{Z}_i$, where $\alpha \in \mathbf{S} \setminus \mathbf{Z}_i$. Assume $\mathbf{S}$ includes the strings from the cycle $[(01)^{m-1}1]$ when $n = 2m - 1$.

```
procedure UC2($\alpha$)
    for $i \leftarrow 1$ to $L + s - i$ do
        PRINT($a_1$)
        $x \leftarrow$ PCR3($\alpha$)
        $\beta \leftarrow a_2 \cdots a_n x$
        if $\beta \notin \mathbf{S}$ then  $x \leftarrow \overline{x}$
        if $\beta = z(i)$ then  $x \leftarrow \overline{x}$
        $\alpha \leftarrow a_2 \cdots a_n x$
```

the counter t'. In the special case when $n = 2m - 1$, a flag is set to make sure the cycle $[(01)^{m-1}1]$ is included; the first string visited on this cycle is $(01)^{m-1}1$.

▶ **Theorem 7.** *Algorithm 3 generates a binary cut-down DB sequence of length L in $\mathcal{O}(n)$ time per symbol using $\mathcal{O}(n)$ space.*

Example 5 Recall the MC from Example 3 of length 51 where $L = 46$ and $s = 5$. Setting $\mathbf{R} = \{001001, 010101\}$, the cycles $[001]$ and $[01]$ are cut out to obtain a cut-down DB sequence of length L. This is illustrated below where the dashed red arcs are not followed.



The resulting cut-down DB sequence of length $L = 46$ starting from $\alpha = 000001$ is

$$[\,0000011110011100011011010011000010110010100010\,].$$

Note: Cutting out cycles Z_i where i does not divide n involves cutting chunks out of two PCR cycles.

4.3 A successor-rule construction

In this section we define a successor rule that can be used to construct a binary cut-down DB sequence of length L. Unlike the algorithm in the previous section, no context is required when iterating through the successor rule; furthermore, determining whether or not a length-n string is found on the sequence can be computed efficiently.

Algorithm 3 Pseudocode for constructing a binary cut-down DB sequence of length L assuming precomputed values m, h, t and the set $\mathbf{R}$

```
procedure CUT-DOWN
    $\alpha = a_1 a_2 \cdots a_n \leftarrow 0^{n-1}1$
    $t' \leftarrow 0$
    if $n = 2m - 1$ then $flag \leftarrow 1$
    else $flag \leftarrow 0$

    for $i \leftarrow 1$ to $L$ do
        PRINT($a_1$)

        ▷ UC-successor for the Main Cycle
        $w \leftarrow$ weight of $\alpha$
        $x \leftarrow$ PCR3($\alpha$)
        $\beta \leftarrow a_2 \cdots a_n x$
        if $w = m$ and $w - a_1 + x = m + 1$ then $x \leftarrow \overline{x}$
        if $w = m - 1$ and $w - a_1 + x = m$ then
            if $\mathrm{per}(\beta) > h$ then $x \leftarrow \overline{x}$
            if $\mathrm{per}(\beta) = h$ then
                if $\beta = (01)^{m-1}1$ then $flag \leftarrow 0$
                ▷ Cut out excess cycles of weight $m$ and period $h$
                if $t' = t$ or ($t' + 1 = t$ and $flag = 1$) then $x \leftarrow \overline{x}$
                else $t' \leftarrow t' + 1$

        if $\beta \in \mathbf{R}$ then $x \leftarrow \overline{x}$    ▷ Cut out small cycle(s)
        $\alpha \leftarrow a_2 \cdots a_n x$
```

Recall that the set $\mathbf{S}$, which contains the length-n substrings of an MC, depends on t PCR cycles of weight m and period h that are not known *a priori*. The key to our upcoming successor rule is to define a specific set of PCR cycles with weight m and period h. Let $\mathbf{T}$ denote the set of length-n strings belonging to PCR cycles corresponding to the t **lexicographically largest** Lyndon words of length h and weight mh/n. Considering the t largest Lyndon words instead of the t smallest 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. Let γ be the Lyndon word with rank $L_2(h, mh/n) - t + 1$ in the lexicographic ordering of Lyndon words with length h, and weight mh/n; there are t Lyndon words with length h and weight mh/n that are lexicographically greater than or equal to γ. Applying a recent unranking algorithm for fixed-weight Lyndon words in lexicographic order, γ can be computed in $\mathcal{O}(h^4)$ operations on h-bit numbers [16]. Thus, given γ, we can determine whether or not β is in $\mathbf{T}$ by applying a simple string comparison (see Line 12, noting β is a necklace, in the upcoming Algorithm 4).

Let $\mathbf{S}'$ denote the set $\mathbf{S}$ that includes the specific subset $\mathbf{T}$. Let $\mathbf{C}$ denote the set of strings cut out from the main cycle. Specifically,

$$\mathbf{C} = \begin{cases} \mathbf{Z}_s & \text{if } s \leq \lceil \frac{n}{2} \rceil; \\ \mathbf{Z}_{\lceil \frac{n}{2} \rceil} \cup \mathbf{Z}_{s - \lceil \frac{n}{2} \rceil} & \text{otherwise.} \end{cases}$$

By precomputing γ, we adapt Algorithm 3 to obtain the procedure CUTDOWNSUCCESSOR(α), given in Algorithm 4, that constructs a cut-down DB sequence of length L; it is a universal cycle for $\mathbf{S}' \setminus \mathbf{C}$. The algorithm can be initialized with any string α in $\mathbf{S}' \setminus \mathbf{C}$.

Algorithm 4 A successor rule based construction of a cut-down DB sequence of length L based on the precomputed values m, h, t, the Lyndon word γ, and the set $\mathbf{R}$.

```
procedure CutDownSuccessor($\alpha = a_1a_2\cdots a_n$)
    for $i \leftarrow 1$ to $L$ do
        Print($a_1$)

        ▷ Context-free UC-successor for the Main Cycle
        $w \leftarrow$ weight of $\alpha$
        $x \leftarrow$ PCR3($\alpha$)
        $\beta \leftarrow a_2\cdots a_n x$
        if $w > m$ or ($w = m$ and $\text{per}(\beta) > h$) then $x \leftarrow \overline{x}$
        if $w = m - 1$ and $w - a_1 + x = m$ then
            if $\text{per}(\beta) > h$ then $x \leftarrow \overline{x}$
            if $\text{per}(\beta) = h$ and $a_2a_3\cdots a_{h+1} < \gamma$ then $x \leftarrow \overline{x}$

        if $\beta \in \mathbf{R}$ then $x \leftarrow \overline{x}$    ▷ Cut out small cycle(s)
        $\alpha \leftarrow a_2\cdots a_n x$
```

▶ **Theorem 8.** CutDownSuccessor(α) *generates a cut-down DB sequence of length L starting from any $\alpha \in \mathbf{S}' \setminus \mathbf{C}$ in $\mathcal{O}(n)$ time per symbol using $\mathcal{O}(n)$ space.*

Proof. Every operation in the **for** loop can be computed in linear time. Thus, each symbol can be computed in $\mathcal{O}(n)$ time. Clearly, the algorithm requires $\mathcal{O}(n)$ space to store a constant number of length-n strings. ◀

To determine whether or not a string is in $\mathbf{S}' \setminus \mathbf{C}$, we can compute γ via an unranking algorithm and apply the logic from Algorithm 4; the result is a membership tester that requires $\mathcal{O}(n^4)$ n-bit operations. However, by applying a related ranking algorithm for fixed-weight Lyndon words [16], we can perform this test using $\mathcal{O}(n^3)$ operations on n-bit numbers.

▶ **Lemma 9.** *Determining whether or nor a length-n string β belongs to $\mathbf{S}' \setminus \mathbf{C}$ can be computed in $\mathcal{O}(n^3)$ operations on n-bit numbers.*

Proof. Consider a length-n string β. Since there are less than n strings in $\mathbf{C}$, testing whether or not β is in $\mathbf{C}$ can be determined in $\mathcal{O}(n^2)$ time. If β has weight less than m, or if it has weight m and period less than h, then it belongs to $\mathbf{S}$. If β has weight greater than m, or weight equal to m and period greater than h, then it does not belong to $\mathbf{S}$. These cases can be handled in $\mathcal{O}(n)$ time. If β has weight m and period h, let $\ell_1\ell_2\cdots\ell_n$ denote the lexicographically smallest rotation of β. If the rank of $\ell_1\cdots\ell_h$ is greater than or equal to $L_2(h, mh/n) - t + 1$, then β is in $\mathbf{T}$ and hence in $\mathbf{S}'$; otherwise, β is not in $\mathbf{S}'$. As noted earlier, the rank can be computed using $\mathcal{O}(n^3)$ operations on n-bit numbers. Thus, since $L_k(n, w)$ and the variables m, h, t, s can be easily be computed in $O(n^3)$ time, testing whether or not β is in $\mathbf{S}' \setminus \mathbf{C}$ can be computed using $\mathcal{O}(n^3)$ operations on n-bit numbers. ◀

5 Cut-down DB sequences for $k > 2$

In this section we extend the strategy for constructing binary cut-down DB sequences to alphabets of arbitrary size.[4] When generalizing the binary approach, the selection of an underlying de Bruijn successor is critical to a simple construction for a cut-down DB sequence when $k > 2$. The PCR3 successor applied in the binary case has two natural generalizations for $k \geq 2$. These generalizations have been previously defined as g_3 and g_3' in [13]. The key to selecting an underlying de Bruijn successor is to allow for the simplest possible method to cut out small cycles; for our purposes, g_3' is the one that allows for this. It can be computed in $\mathcal{O}(n)$ time using $\mathcal{O}(n)$ space [13]; we relabel this de Bruijn successor to PCR3_k below[5], noting $\alpha = a_1a_2\cdots a_n$.

PCR3_k **de Bruijn successor**:

Let y be the smallest symbol in $\{1, 2, \ldots, k{-}1\}$ such that $a_2a_3\cdots a_ny$ is a necklace, or $y = 0$ if no such symbol exists. Then:

$$\text{PCR3}_k(\alpha) = \begin{cases} k{-}1 & \text{if } y > 0 \text{ and } a_1 = y{-}1; \\ a_1{-}1 & \text{if } y > 0 \text{ and } a_1 > y{-}1; \\ a_1 & \text{otherwise.} \end{cases}$$

Interestingly, the original presentation of PCR3_k is described as a UC-successor for k-ary strings with weight less than or equal to some fixed w. However, we choose to apply the restriction separately from the successor as we additionally must consider the periods of the cycles.

The challenge when extending to larger alphabets is that the cycle-joining approach may no longer apply disjoint conjugate pairs. Instead, several cycles which have common substrings of length $n{-}1$ can be joined in a cyclic fashion; the same string can belong to more than one conjugate pair used during the cycle-joining process. As an example, see Figure 3 which illustrates how PCR3_k joins PCR cycles for $n = 3$ and $k = 4$ to obtain the DB sequence

$$[\ 0003303203103002302202102001301201133132131123122333232221211101\ ].$$

As a specific example of how PCR3_k joins cycles, consider the cycles [003], [031], [032], and [033]. Consider a UC that has joined [003] but none of the other listed cycles. Then [031] is joined via the conjugate pair (003, 103). Subsequently, [032] is joined via the conjugate pair (003, 203), and finally [033] is joined via the conjugate pair (003, 303). Observe that 003 is used in all three conjugate pairs. This leads to the substring 003303203103 highlighted in the above DB sequence obtained via repeated application of Lemma 1. Starting from 003, the cycles are visited in the order [003], [033], [032], [031] as illustrated in Figure 3.

5.1 The MC for $k > 2$

As in the binary case, let $\mathbf{S}$ denote the set of strings belonging to an MC with a corresponding subset $\mathbf{T}$ of ht strings belonging to t cycles of weight m and period h. Recall $|\mathbf{S}| = L + s$ and $k^{n-1} < L \leq k^n$. The choice of the underlying de Bruijn successor PCR3_k allows for a simple construction of the MC when $k > 2$.

[4] In [9], Etzion concludes by stating that his binary construction of cut-down DB sequences can be generalized to alphabets of arbitrary size; however, no details are provided.

[5] PCR3_k is labeled PCR3 (alt) in [1].

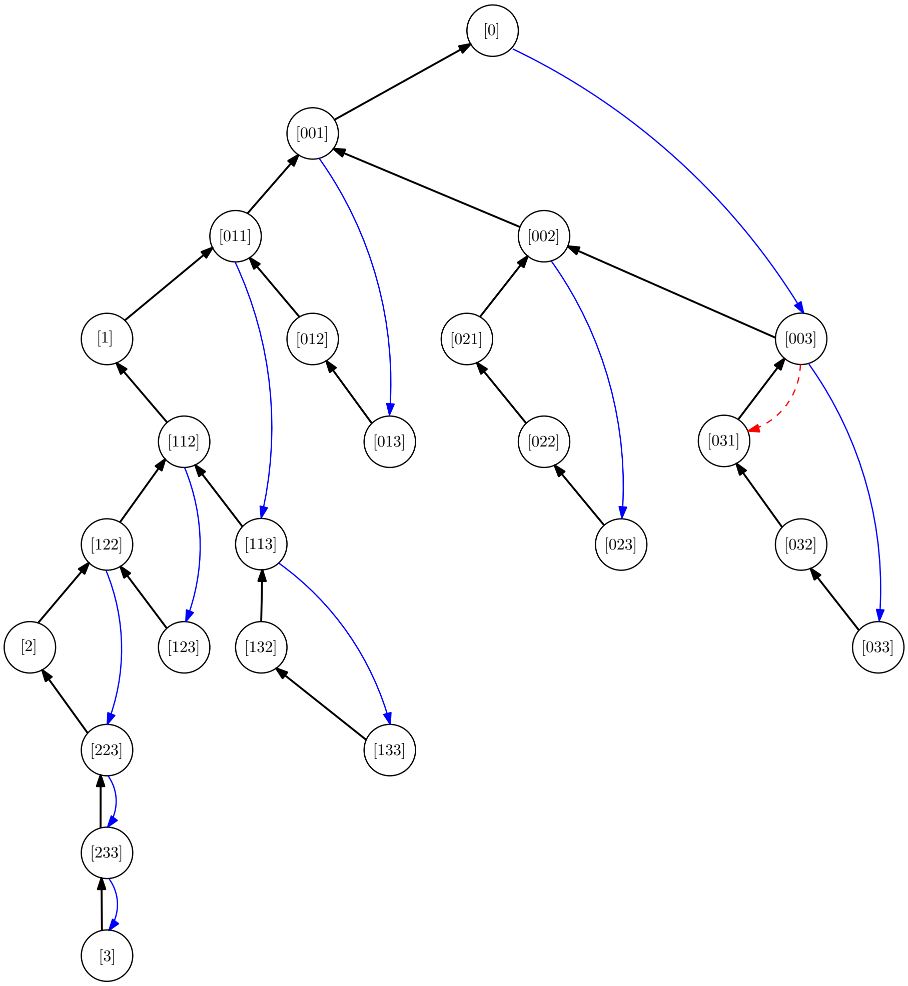


Figure 3 Joining PCR cycles for $n = 3$, $k = 4$ by applying PCR3$_k$. The cycles drawn at the same level have the same weight. When constructing an MC with maximum weight $m = 4$, the dashed red edge illustrates how the cycles [032] and [033] can be cut out while still including [031]. The directions of the arrows indicate the order the cycles are visited when starting from [0].

Given $\alpha = a_1a_2\cdots a_n$, let $F_k(\alpha) = a_2\cdots a_n\mathrm{PCR3}_k(\alpha)$. Like in the binary case, we need only make a minor modification to the de Bruijn successor when attempting to branch to a PCR cycle with larger weight that does not belong to the MC. In particular, if $F_k(\alpha)$ is not in $\mathbf{S}$, then it must be that $y = a_1 + 1$ is the smallest value such that $a_2\cdots a_n y$ is a necklace. This is the only case where $F_k(\alpha)$ has weight greater than α, noting $\mathrm{PCR3}_k(\alpha) = k-1$. Instead, the next symbol in the universal cycle should be the **largest symbol so the resulting string belongs to S**. This ensures we only cut out PCR cycles containing strings that do not belong to $\mathbf{S}$. The resulting algorithm is detailed in Algorithm 5. It differs from Algorithm 1 only at Line 6 to handle when attempting to branch to a PCR cycle not in the MC.

▶ **Lemma 10.** *Algorithm 5 generates a universal cycle for* $\mathbf{S}$.

Example 6 Consider Figure 3 where $m = 4$ and $h = 3$. Consider $\alpha = 003$, which belongs to the cycle [003]. Observe that $F_k(\alpha) = 033$. This string belongs to [033] with weight 6 and

Algorithm 5 Pseudocode for constructing a universal cycle for $\mathbf{S}$ assuming the input $\alpha = a_1a_2\cdots a_n \in \mathbf{S}$ and $k > 2$.

```
procedure $k$-MC($\alpha$)
    for $i \leftarrow 1$ to $|\mathbf{S}|$ do
        PRINT($a_1$)
        $x \leftarrow$PCR3$_k(\alpha)$
        $\beta \leftarrow a_2\cdots a_nx$
        if $\beta \notin \mathbf{S}$ then  $x \leftarrow$ the largest symbol such that $a_2\cdots a_nx \in \mathbf{S}$
        $\alpha \leftarrow a_2\cdots a_nx$
```

hence is not on the MC. If we stay on the cycle 003 by setting $x = 0$, then we also *cut out* cycles [032] and [031] which may contain strings on the MC. However, if we have not already visited t cycles with weight $m = 4$, then we still want to join the cycle [031], as illustrated by the red dashed edge in Figure 3; i.e., x should be assigned 1, which is the largest symbol such that $03x \in \mathbf{S}$.

Using the same notation as in the binary case, let $MC_{\mathbf{S}}$ denote the universal cycle for $\mathbf{S}$ obtained from Algorithm 5. Next, we demonstrate how small cycles Z_i can be cut out of $MC_{\mathbf{S}}$ to obtain a universal cycle of the desired length L.

5.2 Cutting out small cycles for $k > 2$

Like in the binary case, Algorithm 5 can be applied to construct a specific MC by counting the number of cycles added to the MC of weight m and period h. By the choice of the successor PCR3$_k$, we can cut out the same small cycles Z_i using the same set $\mathbf{R}$ from the binary case. The resulting Algorithm 6 will construct a cut-down DB sequence of length L for $k > 2$. The key differences from the binary algorithm are as follows:

- The initial string is generalized to $0^{n-1}(k-1)$, as it is the substring following 0^n in the DB sequence generated by PCR3$_k$.
- The two special cases no longer need to be considered since m will always be greater than $\lceil n/2 \rceil$ with $k > 2$. Thus, the variable *flag* is no longer required.
- Lines 9-15 apply simple operations to cut out appropriate cycles based on the definition of PCR3$_k$ to obtain the desired MC.
- Line 17 assigns x to 0, which is the same as complementing x when $x = 1$ in the binary case. The strings in $\mathbf{R}$ remain the same.

▶ **Theorem 11.** *Algorithm 6 generates a cut-down DB sequence of length $k^{n-1} < L \leq k^n$ where $k > 2$ requiring $\mathcal{O}(n)$ time per symbol and using $\mathcal{O}(n)$ space.*

Proof. The modifications from Algorithm 5 to include only PCR cycles in $MC_{\mathbf{S}}$ are straightforward (Lines 9-15). Thus, we focus on Line 17, which cuts out one or two cycles of the form Z_i, depending on $\mathbf{R}$. Since the strings in these cycles are disjoint, we focus on a single case of $z(i) \in \mathbf{R}$, where $1 < i \leq \lceil \frac{n}{2} \rceil$. Since $k > 2$, each string in $\mathbf{Z}_i$ is clearly a substring of $MC_{\mathbf{S}}$; they have small weight and L is large. The arguments provided in Lemma 5 for $k = 2$ also apply for $k > 2$, that is, $w(i)$ is a substring of $MC_{\mathbf{S}}$. The arguments for cutting out the cycle are also the same as the binary case. Since the symbol following $w(i)$ in $MC_{\mathbf{S}}$ is PCR3$_k(w_i\cdots w_{n+i-1}) = 0$, x is assigned 0 at Line 17; we do not need to consider the case when $i = 1$, since the algorithm is initialized to $0^{n-1}(k-1)$

Algorithm 6 Pseudocode for constructing a cut-down DB sequence (for $k > 2$) of length L assuming precomputed values m, h, t and the set $\mathbf{R}$.

```
procedure $k$-CUT-DOWN
    $\alpha = a_1 a_2 \cdots a_n \leftarrow 0^{n-1}(k-1)$
    $t' \leftarrow 0$
    for $i \leftarrow 1$ to $L$ do
        PRINT($a_1$)

        ▷ UC-successor for the Main Cycle
        $w \leftarrow$ weight of $\alpha$
        $x \leftarrow \mathrm{PCR3}_k(\alpha)$
        if $w - a_1 + x \geq m$ then
            $x \leftarrow m - w + a_1$
            $\beta \leftarrow a_2 \cdots a_n x$
            if $\mathrm{per}(\beta) > h$ then $x \leftarrow x - 1$    ▷ Cut out cycles of weight $m$ and period $> h$
            if $\mathrm{per}(\beta) = h$ then
                if $t' = t$ then $x \leftarrow x - 1$    ▷ Cut out excess cycles of weight $m$ and period $h$
                else $t' \leftarrow t' + 1$

        $\beta \leftarrow a_2 \cdots a_n x$
        if $\beta \in \mathbf{R}$ then $x \leftarrow 0$    ▷ Cut out small cycle(s)
        $\alpha \leftarrow a_2 \cdots a_n x$
```

and the string $\alpha = 10^{n-1}$ is visited in the final iteration of the **for** loop. By limiting the number of iterations of the **for** loop to L to account for cutting out the one or two small cycles, the resulting Algorithm 6 generates a cut-down DB sequence of length L. ◀

5.3 Precomputing m, h, t, s for $k > 2$

The time to precompute the values m, h, t, s for $k > 2$ depends on the time to enumerate $T_k(n', w)$ for all $0 \leq n' \leq n$ and $0 \leq w \leq (k-1)n$. These values can be computed in $\mathcal{O}(k^2 n^2)$ time by applying dynamic programming techniques to the following recurrence for $n \geq 0$:

$$T_k(n', w) = \begin{cases} 0 & \text{if } w < 0 \text{ or } (n' = 0 \text{ and } w > 0); \\ 1 & \text{if } n' = 0 \text{ and } w = 0; \\ \sum_{j=0}^{k-1} T_k(n'-1, w-j) & \text{otherwise.} \end{cases}$$

Example 7 We illustrate the computations of m, h, t, s for $n = 6$, $k = 3$ and $L = 617$. First, we compute the following table of values for $T(n', w)$:

$n' \setminus w$	0	1	2	3	4	5	6	7	8
1	1	1	1	0	0	0	0	0	0
2	1	2	3	2	1	0	0	0	0
3	1	3	6	7	6	3	1	0	0
4	1	4	10	16	19	16	10	4	1
5	1	5	15	30	45	51	45	30	15
6	1	6	21	50	90	126	141	126	90

Recall the definitions of $A(w)$, $B(w,p)$, and $C(w,p)$ defined in Section 3, for $n = 6$ and $k = 3$.

Since $A(7) = 561$ and $A(8) = 651$, we have $m = 8$. Since $B(8,1) = B(8,2) = 0$, $B(8,3) = 6$, $B(8,4) = B(8,5) = 0$, and $B(8,6) = 84$, we have $C(8,5) = 6$ and $C(8,6) = 90$. Thus $h = 6$. Since $A(8) + C(8,5) + 9h = 621$, we have $t = 9$ and surplus $s = 4$.

6 Summary and future work

In this paper, we have enhanced Etzion's algorithm [10] to construct binary cut-down DB sequences. Moreover, we generalize the algorithm to alphabets of arbitrary size by selecting an appropriate underlying feedback function. The resulting algorithms run in $\mathcal{O}(n)$-time per symbol using $\mathcal{O}(n)$ space after some initialization requiring polynomial time and space; they are available for download at http://debruijnsequence.org/db/cutdown [1]. By utilizing an efficient algorithm to rank/unrank fixed-weight Lyndon words, we developed the first successor-rule construction for binary cut-down DB sequences that only requires the current length-n substring to determine the next bit. It also requires $\mathcal{O}(n)$-time per symbol using $\mathcal{O}(n)$ space after an initialization phase. However, it is important to note that the efficient ranking algorithm only applies to the binary case.

It is not difficult to observe that the cut-down DB sequences produced by our algorithms are not *balanced*. Thus, avenues for future research include:

1. Develop an efficient (cycle-joining) construction for generalized DB sequences.
2. Develop an efficient (cycle-joining) construction for balanced cut-down DB sequences.
3. Develop an efficient ranking algorithm for fixed-weight Lyndon words and necklaces for $k > 2$.

7 Acknowledgment

We would like to thank the anonymous reviewers. Their suggestions have improved the quality of our presentation. In particular, the suggestion to precompute γ in Section 4.3 using unranking (instead of a membership tester which applied ranking) resulted in the $O(n)$-time successor-rule.

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*, 3 (2011), 1293–1300.

3 Bar-Lev, D., Kobovich, A., Leitersdorf, O., and Yaakobi, E. Universal framework for parametric constrained coding, arXiv:2304.01317, 2023.

4 Blum, L., Blum, M., and Shub, M. Comparison of two pseudo-random number generators. In *Advances in Cryptology* (Boston, MA, 1983), D. Chaum, R. L. Rivest, and A. T. Sherman, Eds., Springer US, pp. 61–78.

5 Blum, L., Blum, M., and Shub, M. A simple unpredictable pseudo-random number generator. *SIAM Journal on Computing 15*, 2 (1986), 364–383.

6 Chang, Z., Chrisnata, J., Ezerman, M. F., and Kiah, H. M. Rates of DNA sequence profiles for practical values of read lengths. *IEEE Transactions on Information Theory 63*, 11 (2017), 7166–7177.

7 Diaconis, P., and Graham, R. *Magical Mathematics: The Mathematical Ideas That Animate Great Magic Tricks*. Princeton University Press, 2011.

8 Elishco, O., Gabrys, R., Yaakobi, E., and Médard, M. Repeat-free codes. *IEEE Transactions on Information Theory 67*, 9 (2021), 5749–5764.

9 Etzion, T. An algorithm for generating shift-register cycles. *Theoretical Computer Science 44* (1986), 209–224.

10 Etzion, T. Self-dual sequences. *Journal of Combinatorial Theory, Series A 44*, 2 (1987), 288–298.

11 Gabric, D., Holub, S., and Shallit, J. Maximal state complexity and generalized de Bruijn words. *Information and Computation 284* (2022), 104689. Selected Papers from DCFS 2019, the 21st International Conference on Descriptional Complexity of Formal Systems.

12 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.

13 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.

14 Gilbert, E. N., and Riordan, J. Symmetry types of periodic sequences. *Illinois Journal of Mathematics 5*, 4 (1961), 657 – 665.

15 Golomb, S. W. *Shift Register Sequences*, 3rd revised ed. World Scientific, 2017.

16 Hartman, P., and Sawada, J. Ranking and unranking fixed-density necklaces and Lyndon words. *Theoretical Computer Science 791* (2019), 36–47.

17 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.

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

19 Jackson, B., Stevens, B., and Hurlbert, G. Research problems on Gray codes and universal cycles. *Discrete Mathematics 309*, 17 (2009), 5341–5348.

20 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.

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

22 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 (1970), 1204–1209.

23 Lempel, A. m-ary closed sequences. *Journal of Combinatorial Theory, Series A 10*, 3 (1971), 253–258.

24 Matthews, R. Maximally periodic reciprocals. *Bulletin of The Institute of Mathematics and its Applications 28* (01 1992).

25 Nellore, A., and Ward, R. Arbitrary-length analogs to de Bruijn sequences. In *33rd Annual Symposium on Combinatorial Pattern Matching (CPM 2022)* (2022), pp. 9:1–9:20.

26 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.

27 Sawada, J., Sears, J., Trautim, A., and Williams, A. Demystifying our grandparent's de Bruijn sequences with concatenation trees. *arXiv preprint arXiv:2308.12405* (2023).

28 Sawada, J., Williams, A., and Wong, D. Universal cycles for weight-range binary strings. In *Combinatorial Algorithms - 24th International Workshop, IWOCA 2013, Rouen, France, July 10-12, 2013, LNCS 8288* (2013), pp. 388–401.

29 Sawada, J., Williams, A., and Wong, D. A surprisingly simple de Bruijn sequence construction. *Discrete Mathematics 339*, 1 (2016), 127–131.

30 Scheinerman, E. R. Determining planar location via complement-free de Brujin sequences using discrete optical sensors. *IEEE Transactions on Robotics and Automation 17* (2001), 883–889.

31 Sinden, F. W. Sliding window codes. *AT&T Bell Labs Technical Memorandum* (1985).

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