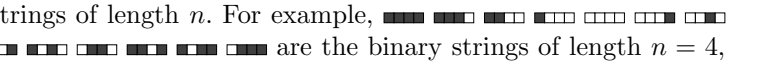
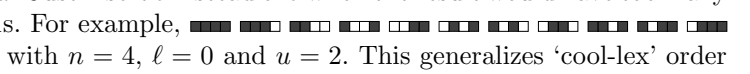
Theory of Computing Systems manuscript No.
(will be inserted by the editor)

The coolest way to generate binary strings

Brett Stevens · Aaron Williams

the date of receipt and acceptance should be inserted later

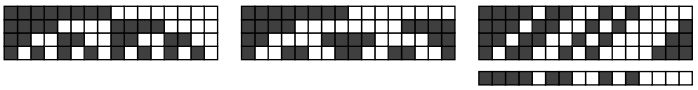
Abstract Pick a binary string of length n and remove its first bit b. Now insert b after the first remaining 10, or insert $\overline{b}$ at the end if there is no remaining 10. Do it again. And again. *Keep going!* Eventually, you will cycle through all 2^n of the binary strings of length n. For example,  are the binary strings of length $n = 4$, where $1 =$ □ and $0 =$ ■. And if you only want strings with weight (number of 1s) between ℓ and u? Just insert b instead of $\overline{b}$ when the result would have too many 1s or too few 1s. For example,  are the strings with $n = 4$, $\ell = 0$ and $u = 2$. This generalizes 'cool-lex' order by Ruskey and Williams (*The coolest way to generate combinations*, Discrete Mathematics) and we present two applications of our 'cooler' order. First, we give a loopless algorithm for generating binary strings with any weight range in which successive strings have Levenshtein distance two. Second, we construct de Bruijn sequences for (i) $\ell = 0$ and any u (maximum specified weight), (ii) any ℓ and $u = n$ (minimum specified weight), and (iii) odd $u - \ell$ (even size weight range). For example, all binary strings with $n = 6$, $\ell = 1$, and $u = 4$ appear once (cyclically) in . We also investigate the recursive structure of our order and show that it shares certain sublist properties with lexicographic order.

Keywords cool-lex order, Gray code, binary strings, combinatorics on words, necklace prefix algorithm, FKM algorithm, de Bruijn sequence, universal cycle, Hamming distance, Levenshtein distance

Research supported in part by NSERC.

Research supported in part by the NSERC Accelerator and Discovery Programmes, and a basic research grant from ONR.

School of Mathematics and Statistics, Carleton University, Canada · Department of Mathematics and Statistics, McGill University, Canada.



(i) `lexicographic` (ii) Gray (iii) de Bruijn

Table 1 The binary strings of length 4 in (i) `lexicographic` order, (ii) Gray order, and (iii) de Bruijn order. In (iii) the binary strings of length 4 (above) are decoded from the de Bruijn sequence of length $2^4 = 16$ (below).

1 Famous Orders of Binary Strings

Let $\mathbf{B}(n)$ be the set of the binary strings of length n. The *weight* of a binary string is its number of 1s, and we let $\mathbf{B}_w(n)$ be the subset of $\mathbf{B}(n)$ containing those strings with weight w. We refer to $\mathbf{B}_w(n)$ as *fixed-weight binary strings*, and we note that some authors use the term *density* to describe weight of a binary string. More generally, *weight-range binary strings* have a weight lower-bound ℓ and a weight upper-bound u, and are denoted by $\mathbf{B}_\ell^u(n) = \mathbf{B}_\ell(n) \cup \mathbf{B}_{\ell+1}(n) \cup \cdots \cup \mathbf{B}_u(n)$. In particular, $\mathbf{B}(n) = \mathbf{B}_0^n(n)$ and $\mathbf{B}_w(n) = \mathbf{B}_w^w(n)$. Throughout this article, we let □ and ■ represent 1 and 0, respectively, and we let exponentiation denote repetition.

We consider two measures for the 'closeness' of two binary strings of length n. Given $\mathbf{b} = b_1 b_2 \cdots b_n \in \mathbf{B}(n)$ and $\mathbf{c} = c_1 c_2 \cdots c_n \in \mathbf{B}(n)$:

- The *Hamming distance* is the minimum number of bit complements that change $\mathbf{b}$ into $\mathbf{c}$. That is, the Hamming distance is $|\{1 \leq i \leq n : b_i \neq c_i\}|$.
- The *Levenshtein distance* is the minimum number of bit insertions, bit deletions, and bit complements that change $\mathbf{b}$ into $\mathbf{c}$. The Levenshtein distance is sometimes referred to as the *edit distance*.

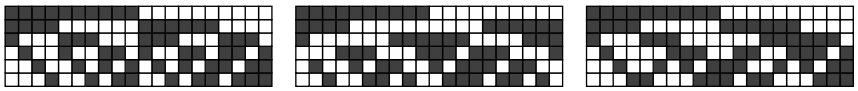
Observe that $\mathbf{b}, \mathbf{c} \in \mathbf{B}(n)$ have Levenshtein distance one if and only they have Hamming distance one. On the other hand, $\mathbf{b}, \mathbf{c} \in \mathbf{B}(n)$ can have Levenshtein distance two and Hamming distance n. In particular, if $n = 2k$, then $(01)^k = 0101 \cdots 01 \in \mathbf{B}(n)$ and $(10)^k = 1010 \cdots 10 \in \mathbf{B}(n)$ differ by n bit complements, but only by one deletion and one insertion.

We discuss famous orders of $\mathbf{B}(n)$ in Section 1.1, and $\mathbf{B}_w(n)$ in Section 1.2. We typeset each order using a distinct font. A preliminary version of this article discusses the reasoning behind each of these choices [26]. Our new results are outlined in Section 1.3.

1.1 Binary Strings of Length n

Table 1 illustrates three orders for $\mathbf{B}(4)$, where individual strings are read top-down, and successive strings are read from left to right.

The `lexicographic` order counts in binary: 0000, 0001, 0010, 0011, ..., 1111. Of the three orders, it is the most organized, since it recursively places



(i) `lexicographic` (ii) Gray (iii) fijnger †

Table 2 The binary strings of $\mathbf{B}_3(6)$ in (i) `lexicographic` order, (ii) Gray order, and (iii) fijnger order. † To facilitate comparison, 0 and 1 are swapped with respect to [6].

all of the strings beginning with 0 before those beginning with 1. In particular, 01^{n-1} is followed by 10^{n-1} in `lexicographic` order, and these strings have Hamming and Levenshtein distance n.

The Gray order cycles through the binary strings while only complementing a single bit at each step: 0000, 0001, 0011, 0010, ..., 1000. See Knuth [12] for a discussion of other orders of $\mathbf{B}(n)$ in which successive strings have Hamming distance one. In this article we focus on the *binary reflected Gray code* patented by Gray [9]. Of the three orders, it is the most versatile, and can be used to gain efficiency in many applications that currently use `lexicographic` order. In fact, the order is so ubiquitous that the term *Gray code* has become synonymous with all minimal-change orders.

The de Bruijn order crams all of $\mathbf{B}(n)$ around a sequence of length 2^n. The sequence is *decoded* by sliding a window of length n along the sequence, so that successive strings differ by deleting the first bit and adding a new last bit: 0000, 0001, 0010, 0100, ..., 1000. The sliding window eventually wraps-around from the end of the sequence to the beginning, so we say the sequence contains all binary strings as *cyclic substrings*. The sliding window mechanism ensures that successive decoded strings have Levenshtein distance at most two, and Hamming distance at most n. In this article we focus on the *lexicographically least de Bruijn sequence* alluded to by Martin [15], formalized by Fredericksen, Kepler, and Maiorana in [8,7], and efficiently generated by Ruskey, Savage and Wang [17], and not the general concept enumerated by de Bruijn [3]. (See Berstel and Perrin [1] for the interesting history dating back to Flye Sainte-Marie [21].) Of the three orders, it is the most compact, using 2^n bits instead of $n \cdot 2^n$ bits.

Despite their differing appearances, the orders are related to one another. The Gray(n) order shares the recursive pattern as `lexicographic`(n), except the Gray$(n-1)$ sublist beginning with 1 is reflected. Also, `lexicographic`(n) and de Bruijn(n) have a deep relationship involving the necklace prefix algorithm, which is discussed in Section 3. For further information on binary string orders refer to Knuth [12], and its updated version in [14].

1.2 Fixed-Weight Binary Strings

Table 2 illustrates three orders for $\mathbf{B}_3(6)$.

The `lexicographic` order of $\mathbf{B}_w(n)$, denoted `lexicographic`$_w(n)$, counts in binary except it skips over the strings that don't have the correct weight:

000111, 001011, 001101,001110, . . ., 111000. In other words, $\texttt{lexicographic}_w(n)$ is the *sublist* of $\texttt{lexicographic}(n)$ that is *induced* by $\mathbf{B}_w(n)$. If $n = 2w$, then 01^w0^{w-1} is followed by 10^w1^{w-1}, so successive strings in $\texttt{lexicographic}_w(n)$ have Hamming distance and Levenshtein distance at most n.

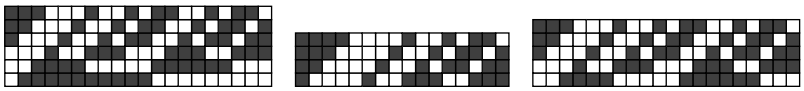
Similarly, the Gray order of $\mathbf{B}_w(n)$, denoted $\mathsf{Gray}_w(n)$, is the sublist of $\mathsf{Gray}(n)$ that is induced by $\mathbf{B}_w(n)$. Successive strings in $\mathsf{Gray}_w(n)$ have Hamming distance two. More specifically, successive strings differ by a *transposition*, meaning that a single 0 is changed to a 1, and a single 1 is changed to a 0. This closeness condition can be easily proven, but is not immediate.

The closeness condition of $\mathsf{Gray}_w(n)$ can be refined as follows. A transposition is *homogeneous* if the bits between the transposed 0 and 1 are all 0s. In other words, a homogeneous-transposition replaces a 10^i substring by 0^i1 or vice versa, where $i > 0$. Eades and McKay [6] were first to construct a homogeneous-transposition Gray code for $\mathbf{B}_w(n)$. Their order is especially useful in situations where the position of the bits set to 1 are stored in an ordered list $p_1 < p_2 < \cdots < p_w$. For example, if a piano student's assignment is to play all w-note chords on a piano with n keys, then they can play consecutive chords without crossing any fingers so long as they follow a homogeneous-transposition Gray code for $\mathbf{B}_w(n)$. For this reason, we refer to the Eades and McKay order as the finger order and denote it by $\mathsf{finger}_w(n)$. Further restrictions of the closeness condition include Chase's Gray code where the only allowed changes are $001 \leftrightarrow 100$ and $01 \leftrightarrow 10$ [2], and Ruskey's Gray code for even n and odd w where only the latter is allowed [16].

A closeness condition that is not possible for fixed-weight binary strings is the one imposed by de Bruijn sequences. More precisely, one cannot create a sequence of length $\binom{n}{w}$ containing each string in $\mathbf{B}_w(n)$ exactly once as a cyclic substring. To see why it's not possible, notice that maintaining a fixed-weight forces successive decoded strings to differ by deleting the first bit and then adding the same bit to the end of the string, thus rotating the string. On the other hand, this does preclude the existence of de Bruijn sequences for weight-range binary strings $\mathbf{B}_\ell^u(n)$ with $\ell < u$. For further information on generating $\mathbf{B}_w(n)$ (also known as *combinations*) refer to Knuth [13] and its update [14].

A last-minute addition to [14] was the *cool-lex order* of $\mathbf{B}_w(n)$ by Ruskey and Williams [20] denoted $\mathsf{cool}_w(n)$. Unlike the other orders in this section, $\mathsf{cool}_w(n)$ is most easily defined iteratively instead of recursively. In the cool order, each successive string is obtained from the previous string by a *successor rule*. The successor rule applies a *prefix-rotation* (or simply *rotation*) to the first i bits, which replaces the prefix $b_1b_2b_3\cdots b_i$ by $b_2b_3\cdots b_ib_1$. The successor rule is *cyclic* in the sense that $\binom{n}{w}$ successive applications of the rule will result in the initial string. The $\mathsf{cool}_w(n)$ order is illustrated for $n = 6$ and $w = 3$ in Table 3 (i).

cool successor rule for $b_1b_2\cdots b_n \in \mathbf{B}_w(n)$
Let i be the minimum value such that $b_ib_{i+1} = 10$ and $i > 1$.
If i exists, then rotate $i + 1$ bits. Otherwise, rotate n bits.



(i) cool for $\mathbf{B}_3(6)$ (ii) cooler for $\mathbf{B}(4)$ (iii) coolest for $\mathbf{B}_2^3(5)$

Table 3 The cool-lex orders for (i) $\mathbf{B}_3(6)$, (ii) $\mathbf{B}(4)$, and (iii) $\mathbf{B}_2^3(4)$.

Theorem 1 ([20]) *The* cool *rule cyclically generates the* $\mathsf{cool}_w(n)$ *order of* $\mathbf{B}_w(n)$.

Since a prefix-rotation can be accomplished by a deletion followed by an insertion, successive strings in cool order have Levenshtein distance two. It is also easy to show that successive strings in cool have Hamming distance at most four [20].

One reason the order is 'cool' is that it accomplishes Theorem 1 without trying, in the sense that the successor rule does not appear to be related to the goal of generating $\mathbf{B}_w(n)$.

Although Theorem 1 may seem 'lucky', the correctness of the cool successor rule comes from having carefully organized sublists in the resulting $\mathsf{cool}_w(n)$ order (see [20] for more information on the recursive definition of cool-lex order). This structure has led to a number of recent applications using cool-lex order including[1]: the first Gray code for fixed-weight Lyndon words and necklaces in standard representation [18], the first simultaneous Gray code for k-ary Dyck words and k-ary trees [4], and the first constructions of de Bruijn sequences for $\mathbf{B}_\ell^u(n)$ when either $u = \ell + 1$ [19] or $\ell = 0$ [23]. These results are largely based on a careful investigation by Ruskey, Sawada, and Williams [18] which proved that cool-lex order provides a simple Gray code for any binary bubble language. In addition, all of the specific Gray code orders mentioned above have led to either *loopless algorithms* or *constant amortized time algorithms*, which generate each successive possibilities in worst-case O(1)-time and amortized O(1)-time, respectively [24,25]. Given the number of applications involving the sublists of $\mathsf{cool}_w(n)$, it is natural to ask if there is a simple 'superlist' that contains $\mathsf{cool}_w(n)$.

1.3 New Results

In this article, we show that cool is cooler than originally thought! We prove that a modification of the successor rule can generate $\mathbf{B}(n)$, and more generally $\mathbf{B}_\ell^u(n)$. The generalized rule differs from the cool rule since it occasionally complements or *flips* the first bit before performing a rotation. To illustrate the generalized rule, the special case of $\mathbf{B}(n)$ (where $\ell = 0$ and $u = n$) is given below. We call this special case the cooler rule, and reserve the coolest name

[1] When consulting these various applications, it should be noted that they may use different modifications of cool-lex order including reflecting the order or strings, reversing the bits in each string, or complementing the bits in each string.

for the most general rule. An example of the resulting order, $\mathsf{cool}(n)$, is given in Table 3 (ii) for $n = 4$.

cooler successor rule for $b_1 b_2 \cdots b_n \in \mathbf{B}(n)$
Let i be the minimum value such that $b_i b_{i+1} = 10$ and $i > 1$. If i exists, then rotate $i + 1$ bits. Otherwise, flip b_1, and then rotate n bits.

It should be mentioned that generalizing a Gray code from $\mathbf{B}_w(n)$ to $\mathbf{B}_\ell^u(n)$ is not difficult. For example, consider finger order. Given a list of strings $\mathcal{L}$, let first($\mathcal{L}$) and last($\mathcal{L}$) denote its first and last string, respectively. In the Eades and McKay order, first($\mathsf{finger}_w(n)$) $= 1^w 0^{n-w}$ and last($\mathsf{finger}_w(n)$) $= 0^{n-w}1^w$. Therefore, if the piano student was assigned the task of playing all chords with at least ℓ notes and at most u notes, and wanted to do so without ever crossing their fingers, then they could follow $\mathsf{finger}_\ell(n)$ from $1^\ell 0^{n-\ell}$ to $0^{n-\ell}1^\ell$, then add a finger to create $0^{n-\ell-1}1^{\ell+1}$ and follow the reflected version of $\mathsf{finger}_{\ell+1}(n)$ to $1^{\ell+1}0^{n-\ell-1}$, and so on, up to $\mathsf{finger}_u(n)$. We say that the resulting order, $\mathsf{finger}_\ell^u(n)$, is *layered by weight* since all strings of a given weight are consecutive, and we consider these generalizations to be trivial. One drawback of a layered Gray code for $\mathbf{B}_\ell^u(n)$ is that they are not cyclic. In particular, the first and last strings will have Hamming and Levenshtein distance at least $u - \ell$. The general problem of finding cyclic orders of $\mathbf{B}_\ell^u(n)$ with restricted Hamming distance includes difficult special cases. For example, the well known "middle levels conjecture" asks whether $\mathbf{B}_k^{k+1}(2k+1)$ has a cyclic Hamming distance 1 Gray code (see Savage and Winkler [22] and Johnson [11]). Our generalization of cool is not layered by weight and it has the following properties

1. The generalized successor rule is very natural.
2. The resulting order is cyclic with respect to successive strings having Hamming distance at most four and Levenshtein distance at most two.
3. The order can be generated by a simple loopless algorithm.
4. The order provides a simpler definition of the de Bruijn sequence construction for $\mathbf{B}_0^u(n)$ from [23]. This leads to a new common generalization that includes the de Bruijn sequence construction for $\mathbf{B}_{w-1}^w(n)$ from [23].

These properties make us feel that we have found the 'right' generalization of $\mathsf{cool}_w(n)$. Section 2 defines the coolest successor rule and generalized cool-lex order, and gives a recursive formula for the resulting orders. Section 4 presents loopless algorithms for generating our orders. Section 3 discusses the necklace prefix algorithm, and our new de Bruijn sequence result. Section 5 examines sublists of our orders.

2 The Coolest Order of Binary Strings

This section introduces our generalization of cool-lex order. Section 2.1 gives a successor rule that generates the order, and Section 2.2 gives a recursive

formula that describes the overall order. A parity-restricted version of the order is defined Section 2.3.

2.1 The Coolest Successor Rule

The generalized coolest successor rule for generating binary strings in any given weight-range appears below. In the special cases of $\ell = u$, and $\ell = 0$ and $u = n$, the coolest rule is equivalent to the cool rule and cooler rule, respectively.

coolest successor rule for $b_1 b_2 \cdots b_n \in \mathbf{B}_\ell^u(n)$
Let i be the minimum value such that $b_i b_{i+1} = 10$ and $i > 1$. If i exists, then rotate $i+1$ bits. Otherwise, flip b_1 if $\overline{b_1} b_2 b_3 \cdots b_n \in \mathbf{B}_\ell^u(n)$, and then rotate n bits.

Our goal is to prove that the coolest rule cyclically generates $\mathbf{B}_\ell^u(n)$. We will denote this order by $\mathsf{cool}_\ell^u(n)$, with Table 3 (iii) showing $\mathsf{cool}_2^3(5)$. To understand the list of strings that coolest creates, it is helpful to first consider the list of strings that cool creates. More specifically, we need to understand the cool rule in the absence of one special string. Let $\mathbf{B}'_n(w) = \mathbf{B}_w(n) \backslash \{0^{n-1}1^w\}$ be the set of fixed-weight binary strings that is *missing* $0^{n-w}1^w$. Let $\mathsf{cool}'_w(n)$ be a non-cyclic order of fixed-weight strings generated by the cool rule such that

$$\text{first}(\mathsf{cool}'_w(n)) = 0^{n-w-1}1^w 0 \text{ and } \text{last}(\mathsf{cool}'_w(n)) = 10^{n-w}1^{w-1}.$$

This order is well-defined by Theorem 1. Now consider two lemmas.

Lemma 1 *The* $\mathsf{cool}'_w(n)$ *order is a non-cyclic order of* $\mathbf{B}'_n(w)$*. In other words, it includes all strings of* $\mathbf{B}_w(n)$ *except the missing string* $0^{n-w}1^w$*.*

Proof Observe that the cool rule creates the following strings consecutively

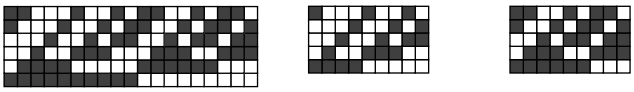
$$\text{last}(\mathsf{cool}'_w(n)) = 10^{n-w}1^{w-1},\ 0^{n-w}1^w,\ 0^{n-w-1}1^w 0 = \text{first}(\mathsf{cool}'_w(n)).$$

Therefore, Theorem 1 implies that $\mathsf{cool}'_w(n)$ contains all strings except for the above string in the middle, $0^{n-1}1^w = 0^{n-w}1^w = \mathbf{B}_w(n) \backslash \mathbf{B}'_n(w)$, as claimed. □

Lemma 2 *The* coolest *rule generates the strings in* $\mathsf{cool}'_w(n)$ *consecutively.*

Proof If $w = 0$ or $w = n$, then the result is vacuously true. Otherwise, observe that the cool and coolest successor rules produce identical successors to strings in $\mathbf{B}_w(n)$ except when the binary string contains no 10 substring after the first bit. There are precisely two such strings: the missing string $0^{n-w}1^w$ and $\text{last}(\mathsf{cool}'_w(n)) = 10^{n-w}1^{w-1}$. Therefore, the cool and coolest rules produce identical successors from $\text{first}(\mathsf{cool}'_w(n))$ to $\text{last}(\mathsf{cool}'_w(n))$. □

Now we can prove our generalized result for the coolest successor rule.



(i) cool for $\mathbf{B}'_6(3)$ (ii) cool for $\mathbf{B}'_5(3)$ (iii) cool for $\mathbf{B}'_5(2)$

Table 4 The cool-lex orders for $\mathbf{B}'_n(w)$ in which $0^{n-w}1^w$ is omitted (i) $\mathsf{cool}'_3(6)$, (ii) $\mathsf{cool}'_3(5)$, and (iii) $\mathsf{cool}'_2(5)$. Removing the left column and bottom row of (i) gives (ii) followed by (iii).

Theorem 2 *The* coolest *rule cyclically generates the following order of* $\mathbf{B}^u_\ell(n)$

$$\mathsf{cool}^u_\ell(n) = 0^{n-\ell}1^\ell,\ 0^{n-\ell-1}1^{\ell+1},\ \ldots,\ 0^{n-u}1^u, \mathsf{cool}'_u(n),\ \mathsf{cool}'_{u-1}(n),\ \ldots,\ \mathsf{cool}'_\ell(n). \quad (1)$$

Proof We prove the result in four steps. First, Lemma 2 implies that the strings in $\mathsf{cool}'_w(n)$ are generated consecutively by the coolest successor rule. Second, observe that the successor rule transforms $\text{last}(\mathsf{cool}'_w(n)) = 10^{n-w}1^{w-1}$ into $\text{first}(\mathsf{cool}'_{w-1}(n)) = 0^{n-w-2}1^{w-1}0$ for all $\ell < w \leq u$. Third, observe that the following strings are consecutively generated by the successor rule

$$10^{n-\ell}1^{\ell-1},\ 0^{n-\ell}1^\ell,\ 0^{n-\ell-1}1^{\ell+1},\ 0^{n-\ell-2}1^{\ell+2},\ \ldots,\ 0^{n-u+1}1^{u-1},\ 0^{n-u}1^u,\ 0^{n-u-1}1^u0.$$

With the exception of the first and last strings, the above list is comprised of the strings that are missing from $\mathsf{cool}'_w(n)$ for all $\ell \leq w \leq u$. Fourth, observe that the first string above is $10^{n-\ell}1^{\ell-1} = \text{last}(\mathsf{cool}'_\ell(n))$ and the last string above is $0^{n-u}1^u = \text{first}(\mathsf{cool}'_u(n))$. Therefore, the strings and lists of (1) are cyclically generated by the coolest rule, which includes all of $\mathbf{B}^u_\ell(n)$ by Lemma 1. □

Since a prefix-rotation can be accomplished by a deletion followed by an insertion, successive strings in $\mathsf{cool}^u_\ell(n)$ have Levenshtein distance two. It is also easy to show that successive strings in $\mathsf{cool}^u_\ell(n)$ have Hamming distance at most four. For example, this is a direct consequence of algorithm Range found in Section 4.3.

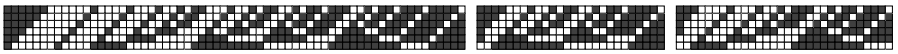
2.2 Recursive Formulae

In this section we recall a recursive formula for the cool order of $\mathbf{B}_w(n)$ and then generalize the formula for the coolest order of $\mathbf{B}^u_\ell(n)$. These formulas are the basis of the recursive algorithms found in Section 4.1 as well as the sublist properties in Section 5. The formula for cool order is illustrated in Table 4.

From Theorem 1 [20], $\mathsf{cool}'_w(n)$ can be expressed recursively as follows

$$\mathsf{cool}'_w(n) = 0^{n-w-1}1^w0,\ \mathsf{cool}'_w(n-1)\cdot 0,\ \mathsf{cool}'_{w-1}(n-1)\cdot 1 \quad (2)$$

where $\cdot$ concatenates x to each string in the given list. The base cases are $\mathsf{cool}'_n(n) = \mathsf{cool}'_0(n) = \epsilon$ (the empty list) for all $n \geq 0$. (The above formula is identical to (5) in [20], except that the order of the strings is reversed and the bits are complemented.)



(i) $\mathsf{cool}_1^4(6)$ (ii) $\mathsf{coool}_1^4(6)$ (iii) $\mathsf{cooool}_1^4(6)$

Table 5 (i) The binary strings in $\mathsf{cool}_1^4(6)$ are restricted to (ii) its odd-weight strings in $\mathsf{coool}_1^4(6) = \mathsf{coool}_1^3(6)$ and (iii) its even-weight strings $\mathsf{cooool}_1^4(6) = \mathsf{cooool}_2^4(6)$.

Given (2) and (1) from Theorem 2, we have recursive descriptions for our cool-lex orders. For convenience, we summarize the formulas for cool and cooler and coolest below

$$\mathsf{cool}_w(n) = 0^{n-w}1^w,\ \mathsf{cool}'_w(n) \tag{3}$$

$$\mathsf{cool}_\ell^u(n) = 0^{n-\ell}1^\ell, 0^{n-\ell-1}1^{\ell+1}, \ldots, 0^{n-u}1^u, \mathsf{cool}'_u(n), \mathsf{cool}'_{u-1}(n), \ldots, \mathsf{cool}'_\ell(n)$$

$$\mathsf{cool}(n) = 0^n,\ 0^{n-1}1,\ \ldots,\ 1^n, \mathsf{cool}'_{n-1}(n),\ \mathsf{cool}'_{n-2}(n),\ \ldots,\ \mathsf{cool}'_1(n) \tag{4}$$

where the last formula uses the fact that $\mathsf{cool}'_n(n)$ and $\mathsf{cool}'_0(n)$ are both empty.

2.3 Coool and Cooool Parity Restrictions

Let $\mathbf{O}(n) = \mathbf{B}_1(n) \cup \mathbf{B}_3(n) \cup ...$ denote the *odd-weight* binary strings of length n, and $\mathbf{E}(n) = \mathbf{B}_0(n) \cup \mathbf{B}_2(n) \cup ...$ denote the *even-weight* binary strings of length n. In our de Bruijn sequence application, we restrict cool-lex order to $\mathbf{O}(n)$ or $\mathbf{E}(n)$. To name the parity-restricted orders, we add o to cool to get the odd coool order, and we add oo to cool to get the even cooool order. More formally, $\mathsf{coool}_\ell^u(n)$ and $\mathsf{cooool}_\ell^u(n)$ are the sublists of $\mathsf{cool}_\ell^u(n)$ containing the odd-weight strings $\mathbf{O}(n)$ and the even-weight strings $\mathbf{E}(n)$, respectively. See Table 5. By Theorem 2 we can express the orders as below. If ℓ and u are both odd, then let

$$\mathsf{coool}_\ell^u(n) = 0^{n-\ell}1^\ell,\ 0^{n-\ell-2}1^{\ell+2},\ ...,\ 0^{n-u}1^u, \mathsf{cool}'_u(n),\ \mathsf{cool}'_{u-2}(n),\ ...,\ \mathsf{cool}'_\ell(n). \tag{5}$$

Similarly, if ℓ and u are both even, then let

$$\mathsf{cooool}_\ell^w(n) = 0^{n-\ell}1^\ell,\ 0^{n-\ell-2}1^{\ell+2},\ ...,\ 0^{n-u}1^u, \mathsf{cool}'_u(n),\ \mathsf{cool}'_{u-2}(n),\ ...,\ \mathsf{cool}'_\ell(n). \tag{6}$$

To make it easier to work with these expressions we also define the following: $\mathsf{coool}_\ell^u(n) = \mathsf{coool}_{\ell+1}^u(n)$ if ℓ is even, $\mathsf{coool}_\ell^u(n) = \mathsf{coool}_\ell^{u-1}(n)$ if u is even, and $\mathsf{cooool}_\ell^u(n) = \mathsf{cooool}_{\ell+1}^u(n)$ if ℓ is odd, $\mathsf{cooool}_\ell^u(n) = \mathsf{cooool}_\ell^{u-1}(n)$ if u is odd.

3 A Family of de Bruijn Sequences

This section describes the necklace prefix algorithm, and how applying it to `lexicographic` order creates a de Bruijn sequence for $\mathbf{B}(n)$. Then we describe related results using cool-lex order for $\mathbf{B}_\ell^u(n)$ when $u = \ell + 1$ or $\ell = 0$.

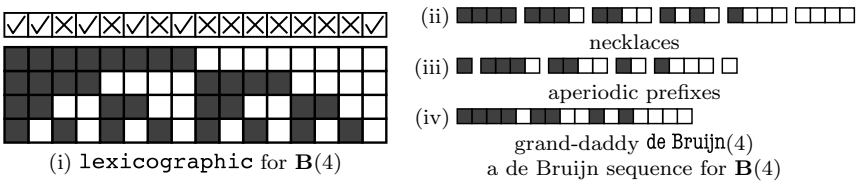


Fig. 1 The necklace prefix algorithm applied to `lexicographic` order creates the de Bruijn sequence for the binary strings of length $n = 4$.

3.1 The Necklace-Prefix Algorithm

A *necklace* is a string in its lexicographically smallest rotation. In other words, $\mathbf{b} = b_1 b_2 \cdots b_n$ is a necklace unless there is an i such that $b_i b_{i+1} \cdots b_n b_1 b_2 \cdots b_{i-1}$ is strictly less than $\mathbf{b}$ in lexicographic order. The *aperiodic prefix* of a string is its shortest prefix that can repeated a whole number of times to create itself. More precisely, the aperiodic prefix of a string $\mathbf{b} = b_1 b_2 \cdots b_n$ is its shortest prefix $\rho(\mathbf{b}) = b_1 b_2 \cdots b_k$ such that $\rho(\mathbf{b})^{n/k} = \mathbf{b}$. The *necklace prefix algorithm* takes a list of strings, removes every non-necklace, reduces the remaining necklaces to their aperiodic prefix, and then glues these prefixes together into a sequence. More formally, if $\mathcal{L}$ is a list of strings, and $\eta_1, \eta_2, \ldots, \eta_m$ is its sublist of necklaces, then the necklace prefix algorithm creates the following sequence

$$\eta\rho(\mathcal{L}) = \rho(\eta_1) \cdot \rho(\eta_2) \cdots \rho(\eta_m) \tag{7}$$

where each $\cdot$ denotes concatenation.

3.2 The Grand-Daddy de Bruijn Sequence

Let us apply the necklace prefix algorithm to the `lexicographic` order of $\mathbf{B}(4)$ in four steps. Figure 1 shows (i) `lexicographic`(4) with ✓ above each necklace, (ii) the necklaces isolated (horizontally), (iii) the necklaces reduced to their aperiodic prefix, and (iv) the prefixes concatenated. Magically, the result is a de Bruijn sequence! In fact, it is the lexicographically least de Bruijn sequence of $\mathbf{B}(4)$.

Theorem 3 ([8,7]) *$\eta\rho($`lexicographic`$(n)) =$ de Bruijn(n) is the lexicographically least de Bruijn sequence for $\mathbf{B}(n)$.*

This method of creating a de Bruijn sequence for $\mathbf{B}(n)$ became known as the *FKM algorithm* for the authors who discovered it. The original proof of Theorem 3 describes de Bruijn(n) as the concatenation of the Lyndon words whose length divides n in lexicographic order; see [19] for a discussion on why the necklace prefix algorithm is a 'better' description. A subsequent analysis by Ruskey, Savage, and Wang [17] showed that this de Bruijn sequence can be constructed efficiently. Due to its impressive definition and efficient construction, Knuth calls de Bruijn(n) the "grand-daddy" of all de Bruijn sequences [12, 14].

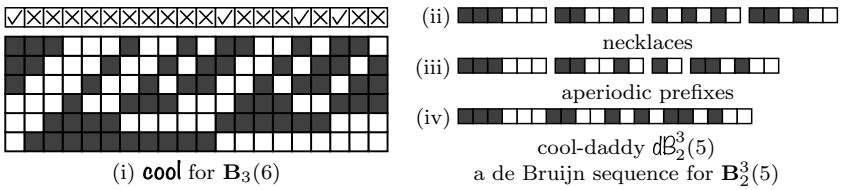


Fig. 2 The necklace prefix algorithm applied to the cool order of $\mathbf{B}_3(6)$ creates a dual-weight de Bruijn sequence for $\mathbf{B}^3_2(5)$.

3.3 The Cool-Daddy de Bruijn Sequence

Recall that de Bruijn sequences do not exist for fixed-weight binary strings $\mathbf{B}_w(n)$. Thus, the tightest possible range of weights for de Bruijn sequences is $\mathbf{B}^w_{w-1}(n)$. These *dual-weight de Bruijn sequences* can be constructed with the following definition,

$$d\mathcal{B}^w_{w-1}(n) = \eta\rho(\mathsf{cool}_w(n{+}1)), \tag{8}$$

where the first string in the cyclic order $\mathsf{cool}_w(n+1)$ is considered to be $0^{n-w+1}1^w$. This definition is illustrated for $n = 5$ and $w = 3$ by Figure 2 using the same four steps as Figure 1. Magically, the result is again a de Bruijn sequence!

Theorem 4 ([19]) *$d\mathcal{B}^w_{w-1}(n) = \eta\rho(\mathsf{cool}_w(n{+}1))$ is a de Bruijn sequence for $\mathbf{B}^w_{w-1}(n)$.*

Theorem 4 doesn't hold when cool is replaced by `lexicographic` or any other order known to the authors. The "cool-daddy" $d\mathcal{B}^w_{w-1}(n)$ can be considered a *fixed-weight de Bruijn sequence* for $\mathbf{B}_w(n{+}1)$, since its $\mathbf{B}^w_{w-1}(n)$ substrings are the unique prefixes of $\mathbf{B}_w(n{+}1)$ that omit the final (redundant) bit. That interpretation is used in [19] with $\mathbb{C}_w(n{+}1) = d\mathcal{B}^w_{w-1}(n)$. Also, $d\mathcal{B}^w_{w-1}(n)$ is denoted $\mathsf{dB}_w(n)$ in [23]. This article uses subscripts/superscripts for lower/upper weights. To conclude this subsection we consider two special cases:

- $d\mathcal{B}^0_{-1}(n) = \eta\rho(\mathsf{cool}_0(n{+}1)) = \rho(0^{n+1}) = 0$ is a de Bruijn sequence for $\mathbf{B}_0(n)$;
- $d\mathcal{B}^{n+1}_n(n) = \eta\rho(\mathsf{cool}_{n+1}(n{+}1)) = \rho(1^{n+1}) = 1$ is a de Bruijn sequence for $\mathbf{B}_n(n)$.

In the rest of the article we let $d\mathcal{B}^0_0(n) = d\mathcal{B}^0_{-1}(n)$ and $d\mathcal{B}^n_n(n) = d\mathcal{B}^{n+1}_n(n)$.

3.4 *La Pecora Nera* de Bruijn Sequence

Now we consider a relative of the grand-daddy and the cool-daddy, whose complicated definition makes it the "black sheep" of the family.

Theorem 4 was extended so that it could create *de Bruijn sequences with a maximum specified weight* by Sawada, Stevens, and Williams [23]. In their construction, they take apart each sequence from Theorem 4 as follows

$$d\mathcal{B}^w_{w-1}(n) = \rho(0^{n+1-w}1^w) \cdot d\mathcal{B}'^w_{w-1}(n) \tag{9}$$

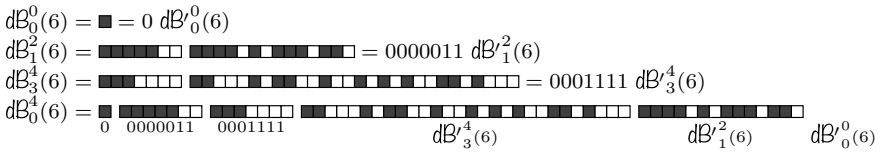


Fig. 3 The "black sheep" construction splits and combines dual-weight de Bruijn sequences to create the de Bruijn sequence $d\mathcal{B}^4_0(6)$ of the binary strings in $\mathbf{B}^4_0(6)$ [23].

This equation splits $d\mathcal{B}^w_{w-1}(n)$ into the bits from its first necklace, $\rho(\eta_1) = \rho(0^{n+1-w}1^w)$, and its remaining bits $d\mathcal{B}'^w_{w-1}(n)$. De Bruijn sequences for $\mathbf{B}^u_0(n)$ are created in [23] by gluing the pieces of (9) together as follows

$$d\mathcal{B}^u_0(n) = \begin{cases} 0\ 0^{n-1}1^2\ 0^{n-3}1^4 \cdots 0^{n-u+1}1^u\ d\mathcal{B}'^u_{u-1}(n) \cdots d\mathcal{B}'^4_3(n)\ d\mathcal{B}'^2_1(n) & \text{if } u \text{ even} \\ 0^n1\ 0^{n-2}1^3\ 0^{n-4}1^5 \cdots 0^{n-u+1}1^u\ d\mathcal{B}'^u_{u-1}(n) \cdots d\mathcal{B}'^5_4(n)\ d\mathcal{B}'^3_2(n) & \text{if } u \text{ odd.} \end{cases} \tag{10}$$

Notice that the $0^{n-w+1}1^w$ necklaces are concatenated by increasing w, followed by the $d\mathcal{B}'^w_{w-1}(n)$ subsequences by decreasing w. (The published versions of Table 1 and 2 in [23] incorrectly order the $d\mathcal{B}'^w_{w-1}(n)$ subsequences by increasing w.)

Theorem 5 ([23]) *$d\mathcal{B}^u_0(n)$ is a de Bruijn sequence for $\mathbf{B}^u_0(n)$.*

Figure 3 illustrates Theorem 5 for $n = 6$ and $u = 4$. The theorem is proven by equating the substrings of $d\mathcal{B}^u_0(n)$ to the substrings in the $d\mathcal{B}^w_{w-1}(n)$ sequences that are spliced together to construct it. A nice corollary of Theorem 5 is that $d\mathcal{B}^u_0(2u+1)$ is a "complement-free" de Bruijn sequence for $\mathbf{B}(2u+1)$ [23]. Lemma 3 helps us redefine $d\mathcal{B}^u_0(n)$ in Section 3.5 and is also illustrated by Figure 3.

Lemma 3 *If $d\mathcal{B}'^w_{w-1}(n)$ is non-empty, then it has the following prefix and suffix*

$$d\mathcal{B}'^w_{w-1}(n) = 0^{n-w}\ \cdots\ 1^{w-1}.$$

Proof If $\mathbf{B}_w(n+1)$ contains one necklace, then $d\mathcal{B}'^w_{w-1}(n)$ is empty. If $\mathbf{B}_w(n+1)$ contains two necklaces, then either (i) $n = 3$ and $w = 2$, (ii) $n = 4$ and $w = 2$, or (iii) $n = 4$ and $w = 3$. In these three cases, (i) $d\mathcal{B}'^w_{w-1}(n) = 01$, (ii) $d\mathcal{B}'^w_{w-1}(n) = 00101$, and (iii) $d\mathcal{B}'^w_{w-1}(n) = 01011$ and the claim is easily verified. Otherwise, if there are at least three necklaces in $\mathbf{B}_w(n+1)$, then Lemma 1 of [23] proves that the following necklaces are consecutive in cool order

$$0^x10^y1^{w-1},\ 0^{n-w+1}1^w,\ 0^{n-w}1^{w-1}01$$

where $x = \lceil (n+1-w)/2 \rceil$ and $y = \lfloor (n+1-w)/2 \rfloor$. Furthermore, these necklaces are aperiodic. This proves the result since $0^{n-w+1}1^w$ is excluded from $d\mathcal{B}'^u_{u-1}(n)$, and so $d\mathcal{B}'^u_{u-1}(n)$ begins with $0^{n-w}1^{w-1}01$ and ends with $0^x10^y1^{w-1}$. □

3.5 The Coolest de Bruijn Sequences

This section gives a common generalization of our de Bruijn sequence constructions for binary strings with dual-weight or maximum specified weight. We begin by re-expressing the "black sheep" and cool-daddy constructions.

Lemma 4 *The de Bruijn sequence* $\mathfrak{dB}^u_0(n)$ *for* $\mathbf{B}^u_0(n)$ *and the de Bruijn sequence* $\mathfrak{dB}^w_{w-1}(n)$ *for* $\mathbf{B}^w_{w-1}(n)$ *can be created from the necklace prefix algorithm and the parity versions of cool-lex order. More specifically,*

$$\mathfrak{dB}^w_{w-1}(n) = \begin{cases} \eta\rho(\mathfrak{coool}^{w+1}_{w-1}(n+1)) & \text{if } w \text{ odd} \\ \eta\rho(\mathfrak{cooool}^{w+1}_{w-1}(n+1)) & \text{if } w \text{ even} \end{cases} \qquad \mathfrak{dB}^u_0(n) = \begin{cases} \eta\rho(\mathfrak{coool}^{u+1}_0(n+1)) & \text{if } u \text{ odd} \\ \eta\rho(\mathfrak{cooool}^{u+1}_0(n+1)) & \text{if } u \text{ even.} \end{cases}$$

(The subscript and superscript values are chosen to accommodate Theorem 6.)

Proof Theorem 4 suffices for $\mathfrak{dB}^w_{w-1}(n)$ since $\mathfrak{coool}^{w+1}_{w-1}(n{+}1) = \mathfrak{cool}_w(n{+}1)$ for odd w, and $\mathfrak{cooool}^{w+1}_{w-1}(n{+}1) = \mathfrak{cool}_w(n{+}1)$ for even w. For $\mathfrak{dB}^u_0(n)$ and even u,

$$\begin{aligned} \mathfrak{dB}^u_0(n) &= 0 \cdot 0^{n-1}1^2 \cdots 0^{n-u+1}1^u \cdot \mathfrak{dB'}^u_{u-1}(n) \cdots \mathfrak{dB'}^4_3(n) \cdot \mathfrak{dB'}^2_1(n) \\ &= \eta\rho(0^{n+1}, 0^{n-1}1^2, \ldots, 0^{n-u+1}1^u, \mathfrak{cool}'_u(n+1), \ldots, \mathfrak{cool}'_4(n+1), \mathfrak{cool}'_2(n+1)) \\ &= \eta\rho(\mathfrak{cooool}^u_0(n+1)) = \eta\rho(\mathfrak{cooool}^{u+1}_0(n+1)) \end{aligned}$$

with (10) and (1) explaining the first and last equalities. Similarly, for odd u,

$$\begin{aligned} \mathfrak{dB}^u_0(n) &= 0^n1 \cdot 0^{n-2}1^3 \cdots 0^{n-u+1}1^u \cdot \mathfrak{dB'}^u_{u-1}(n) \cdots \mathfrak{dB'}^3_2(n) \cdot \mathfrak{dB'}^1_0(n) \\ &= \eta\rho(0^n1, 0^{n-2}1^3, ..., 0^{n-u+1}1^u, \mathfrak{cool}'_u(n+1), ..., \mathfrak{cool}'_3(n+1), \mathfrak{cool}'_1(n+1)) \\ &= \eta\rho(\mathfrak{cooool}^u_1(n+1)) = \eta\rho(\mathfrak{coool}^{u+1}_0(n+1)). \qquad \square \end{aligned}$$

$\square$

Lemma 4 hints at a common generalization. To develop the 'right' generalization, let us step back and reconsider the two constructions:

- $\mathfrak{dB}^w_{w-1}(n)$ is a de Bruijn sequence for two consecutive weights;
- $\mathfrak{dB}^u_0(n)$ is a de Bruijn sequence for consecutive weights beginning with $\ell = 0$.

When u is odd, $\mathfrak{dB}^u_0(n)$ 'includes' $\mathfrak{dB}^w_{w-1}(n)$ for $w = 1, 3, \ldots, u$. This suggests the construction of *even-range de Bruijn sequence* where $\{\ell, \ell+1, \ldots, u\}$ contains an even number of values. When u is even, $\mathfrak{dB}^u_0(n)$ 'includes' $\mathfrak{dB}^w_{w-1}(n)$ for $w = 0, 2, \ldots, u$. In this case, $\mathfrak{dB}^0_0(n) = 0$ contributes the single string of weight $w = 0$, thereby resulting in an odd-range de Bruijn sequence starting from $\ell = 0$. This suggests the construction of *de Bruijn sequences with a minimum specified weight* by using $\mathfrak{dB}^n_n(n) = 1$ to (hopefully) contribute the single string of weight $w = n$. The generalization in Theorem 6 accounts for these two ideas and is illustrated in Figure 4.

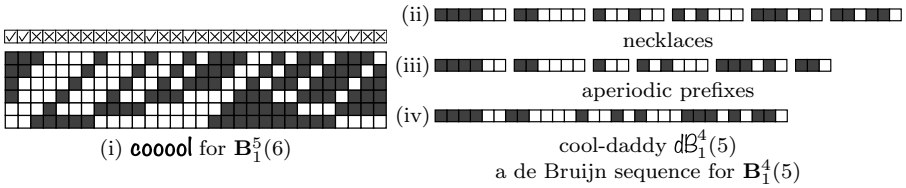


Fig. 4 The necklace prefix algorithm applied to the cooool order of $\mathbf{B}_1^5(6)$ creates a weight-range de Bruijn sequence for $\mathbf{B}_1^4(5)$.

Theorem 6 *De Bruijn sequences for $\mathbf{B}_\ell^u(n)$ can be constructed by the necklace prefix algorithm and the parity versions of cool-lex order whenever (i) $\ell = 0$, or (ii) $u = n$, or (iii) $u - \ell$ is odd. More specifically, the de Bruijn sequences are*

$$\mathrm{d}\mathbb{B}_\ell^u(n) = \begin{cases} \eta\rho(\texttt{cooool}_\ell^{u+1}(n+1)) & \text{if } (\ell \text{ is even or } \ell = 0) \text{ and } (u \text{ is odd or } u = n) \\ \eta\rho(\texttt{coooool}_\ell^{u+1}(n+1)) & \text{if } (\ell \text{ is odd or } \ell = 0) \text{ and } (u \text{ is even or } u = n). \end{cases}$$

When $\ell = 0$ and $u = n$, $\mathrm{d}\mathbb{B}_\ell^u(n)$ gives two definitions for de Bruijn sequences of all binary strings $\mathbf{B}_0^n(n) = \mathbf{B}(n)$, which we call the cooool *and* coooool *constructions. (Note: "ℓ is even" is stated as "ℓ is even or $\ell = 0$" for case symmetry.)*

Proof First we consider several special cases that follow from Lemma 4:

- $\mathrm{d}\mathbb{B}_\ell^u(n)$ is valid when $\ell = u - 1$.
- $\mathrm{d}\mathbb{B}_0^u(n)$ is valid when $\ell = 0$ and $u < n$.
- The cooool construction of $\mathrm{d}\mathbb{B}_0^n(n)$ is valid when $\ell = 0$ and $u = n$ is odd.
- The coooool construction of $\mathrm{d}\mathbb{B}_0^n(n)$ is valid when $\ell = 0$ and $u = n$ is even.

As another special case, we claim the validity of $\mathrm{d}\mathbb{B}_\ell^n(n)$ where $u = n$ and $u - \ell$ is even, reduces to the validity of $\mathrm{d}\mathbb{B}_\ell^{n-1}(n)$. We proceed based on the parity of $u = n$ and ℓ. If $u = n$ and ℓ are odd then consider the following subsequences

$$\mathrm{d}\mathbb{B}_\ell^{n-1}(n) = \eta\rho(\texttt{coooool}_\ell^n(n+1)) = \cdots\rho(0^2 1^{n-1})\cdots = \cdots 001^{n-1}\cdots$$
$$\mathrm{d}\mathbb{B}_\ell^{n}(n) = \eta\rho(\texttt{coooool}_\ell^{n+1}(n+1)) = \cdots\rho(0^2 1^{n-1})\rho(1^{n+1})\cdots = \cdots 001^{n-1}1\cdots .$$

The length n substrings of $\mathrm{d}\mathbb{B}_\ell^{n-1}(n)$ and $\mathrm{d}\mathbb{B}_\ell^n(n)$ are identical, except that the additional 1 in $\mathrm{d}\mathbb{B}_\ell^n(n)$ contributes the unique string in $\mathbf{B}_\ell^n(n)\backslash\mathbf{B}_\ell^{n-1}(n) = \{1^n\}$. Similarly, if $u = n$ and ℓ are even then the same argument applies since

$$\mathrm{d}\mathbb{B}_\ell^{n-1}(n) = \eta\rho(\texttt{cooool}_\ell^n(n+1)) = \cdots\rho(0^2 1^{n-1})\cdots = \cdots 001^{n-1}\cdots$$
$$\mathrm{d}\mathbb{B}_\ell^{n}(n) = \eta\rho(\texttt{cooool}_\ell^{n+1}(n+1)) = \cdots\rho(0^2 1^{n-1})\rho(1^{n+1})\cdots = \cdots 001^{n-1}1\cdots .$$

In this case the argument also covers the validity of the cooool construction of $\mathrm{d}\mathbb{B}_0^n(n)$ where $\ell = 0$ and $u = n$ is even.

A final special case reduces the validity of the **cooool** construction of $d\mathcal{B}_0^n(n)$ when $\ell = 0$ and $u = n$ is odd, to the validity of $d\mathcal{B}_1^{n-1}(n)$ by these subsequences

$$\begin{aligned} d\mathcal{B}_1^{n-1}(n) &= \eta\rho(\textbf{cooool}_1^n(n+1)) = \cdots \rho(0^{n-1}1^2)\cdots\rho(0^2 1^{n-1})\cdots \\ &= \cdots 0^{n-1}11\cdots 001^{n-1}\cdots \\ d\mathcal{B}_0^n(n) &= \eta\rho(\textbf{cooool}_0^{n+1}(n+1)) = \cdots \rho(0^{n+1})\rho(0^{n-1}1^2)\cdots\rho(0^2 1^{n-1})\rho(1^{n+1})\cdots \\ &= \cdots 00^{n-1}11\cdots 001^{n-1}1\cdots . \end{aligned}$$

The length n substrings of $d\mathcal{B}_1^{n-1}(n)$ and $d\mathcal{B}_0^n(n)$ are identical, except the additional bits in $d\mathcal{B}_0^n(n)$ contribute the unique strings in $\mathbf{B}_\ell^n(n)\backslash\mathbf{B}_\ell^{n-1}(n) = \{0^n, 1^n\}$.

This myriad of special cases has reduced the theorem to the cases where $\ell > 0$, $u < n$, $u - \ell$ is odd, and $u - \ell > 1$. For the remainder of the proof we assume u is even, since the proof for odd u is similar. We begin with an expression for our de Bruijn sequence of $\mathbf{B}_0^u(n)$

$$\begin{aligned} d\mathcal{B}_0^u(n) &= \eta\rho(\textbf{cooool}_0^{u+1}(n+1)) \\ &= 0\cdot 0^{n-1}11\cdots 0^{n+1-u}1^u \cdot d\mathcal{B}'^u_{u-1}(n+1)\cdot d\mathcal{B}'^{u-2}_{u-3}(n+1)\cdots d\mathcal{B}'^2_1(n+1) \\ &= 0\cdot 0^{n-1}11\cdots 0^{n-\ell+2}1^{\ell-1}\cdot d\mathcal{B}_\ell^u(n)\cdot d\mathcal{B}'^{\ell-1}_{\ell-2}(n+1)\cdots d\mathcal{B}'^2_1(n+1) \end{aligned}$$

This shows $d\mathcal{B}_\ell^u(n) = 0^{n-\ell}1^{\ell+1}\cdots 0^{n+1-u}1^u\cdot d\mathcal{B}'^u_{u-1}(n+1)\cdots d\mathcal{B}'^{\ell+1}_\ell(n+1)$ is a subsequence of $d\mathcal{B}_0^u(n)$. When $d\mathcal{B}_\ell^u(n)$ is deleted from $d\mathcal{B}_0^u(n)$, the remainder is

$$d\mathcal{B}_0^{\ell-1}(n) = 0\cdot 0^{n-1}11\cdots 0^{n-\ell+2}1^{\ell-1}\cdot d\mathcal{B}'^{\ell-1}_{\ell-2}(n+1)\cdots d\mathcal{B}'^2_1(n+1)$$

where $d\mathcal{B}_0^{\ell-1}(n)$ is our de Bruijn sequence for $\mathbf{B}_0^{\ell-1}(n)$. Since $\mathbf{B}_0^u(n) = \mathbf{B}_0^{\ell-1}(n)\cup \mathbf{B}_\ell^u(n)$, we can now make a conclusion about the substrings of $d\mathcal{B}_0^u(n)$: Each $\mathbf{b}\in\mathbf{B}_\ell^u(n)$ appears as a substring of $d\mathcal{B}_0^u(n)$ that must either be completely inside of the $d\mathcal{B}_\ell^u(n)$ subsequence, or at least overlap with it. In other words, we can conclude that each $\mathbf{b}\in\mathbf{B}_\ell^u(n)$ appears non-cyclically as a substring below

$$0^{n-\ell+2}1^{\ell-1}\cdot d\mathcal{B}_\ell^u(n)\cdot 0^{n-\ell}$$

where the substring on the right is a prefix of $d\mathcal{B}'^{\ell-1}_{\ell-2}(n+1)$ by Lemma 3. (Lemma 3 implies $0^{n-\ell+1}$ is a prefix of $d\mathcal{B}'^{\ell-1}_{\ell-2}(n+1)$, but we trim this prefix since strings in $\mathbf{B}_\ell^u(n)$ have at most $n-\ell$ copies of 0.) By Lemma 3 we can conclude that each $\mathbf{b}\in\mathbf{B}_\ell^u(n)$ appears non-cyclically as a substring below

$$0^{n-\ell+2}1^{\ell-1}\ \cdot\ \overbrace{0^{n-\ell}\ \cdots\ 1^{\ell}}^{d\mathcal{B}_\ell^u(n)}\ \cdot 0^{n-\ell}.$$

Since strings in $\mathbf{B}_\ell^u(n)$ have at most $n-\ell$ copies of 0, we trim the subsequence to

$$1^{\ell-1}\ \cdot\ \overbrace{0^{n-\ell}\ \cdots\ 1^{\ell}}^{d\mathcal{B}_\ell^u(n)}\ \cdot 0^{n-\ell}.$$

The string to the left of $\mathfrak{dB}^u_\ell(n)$ is a suffix of $\mathfrak{dB}^u_\ell(n)$, and the string to the right of $\mathfrak{dB}^u_\ell(n)$ is a prefix of $\mathfrak{dB}^u_\ell(n)$. Therefore, the non-cyclic substrings in the above expression are all cyclic substrings of $\mathfrak{dB}^u_\ell(n)$. Thus, each $\mathbf{b} \in \mathbf{B}^u_\ell(n)$ appears non-cyclically as a substring in $\mathfrak{dB}^u_\ell(n) = 0^{n-\ell} \cdots 1^\ell$. To complete the proof that $\mathfrak{dB}^u_\ell(n)$ is a de Bruijn sequence, note that $\mathfrak{dB}^u_\ell(n)$ has exactly $|\mathbf{B}^u_\ell(n)|$ substrings of length n since $|\mathbf{B}^u_0(n)| = |\mathbf{B}^{\ell-1}_0(n)| + |\mathbf{B}^u_\ell(n)|$. □

4 Algorithms

In this section we consider the efficient generation of the **cool** and **cool**er and **cool**est orders from Section 2, as well as the de Bruijn sequences from Section 3.5. In each algorithm, the current binary string b is stored in an array of length n which is repeatedly modified and *visited*. The array uses 1-based indexing, so the binary string is stored in $b[1]b[2]\cdots b[n]$.

Each algorithm has been implemented in C and is available from the authors. We would also like to note that implementing these algorithms in other languages can be fun. In particular, the authors implemented **cool** order in the PostScript programming language, where prefix-rotations are performed by pushing and popping the stack. This implementation was then used to automatically draw Figures 5 and 6 in Section 5.

4.1 Recursive Algorithms

We begin with recursive algorithms for generating the various versions of cool-lex order, as presented in Algorithms 1 and 2. The key is the $\mathsf{Weight}'(n, w)$ routine for generating $\mathbf{cool}'_w(n)$. (Recall that $\mathbf{cool}'_w(n)$ is the **cool** order of fixed-weight strings with one string missing, $\mathbf{B}'_n(w) = \mathbf{B}_w(n)\backslash\{0^{n-1}1^w\}$.) We prove that each routine runs in *constant amortized time*, meaning that successive strings are visited in amortized O(1)-time.

Theorem 7 *The routines in Algorithms 1 and 2 generate the various versions of cool-lex order in constant amortized time.*

Proof The $\mathsf{Weight}'(n, w)$ routine has the following precondition: The first n bits of binary string b initially holds $0^{n-w}1^w$. The routine begins by swapping the nth and $(n-w)$th bits on lines 2–3. This creates $0^{n-w-1}1^w0$ in b, which is the first string in $\mathbf{cool}'_w(n)$. This string is visited and then the routine recursively calls $\mathsf{Weight}'(n-1, w)$. Observe that the precondition holds during this recursive call since the first $n-1$ bits of b are $0^{n-w-1}1^w$. After the recursive call, the routine swaps the bits back on lines 6–7 so that b holds $0^{n-w}1^w$. Finally, the routine recursively calls $\mathsf{Weight}'(n-1, w-1)$. Observe that the precondition again holds during this recursive call since the first $n-1$ bits of b are $0^{n-w}1^{w-1}$. Since at most two recursive calls can be made before a visit call, the algorithm visits each successive binary string in amortized O(1)-time.

The correctness of $\mathsf{Weight}'(n,w)$ follows from its recursive formula (2). Similarly, the correctness of the remaining routines follow from (3), (4), (1), (5), and (6). Furthermore, each routine runs in constant amortized time due to the fact that $\mathsf{Weight}'(n,w)$ does. □

Algorithm 1 Recursive algorithms for generating $\mathsf{cool}'_w(n)$ (left) and $\mathsf{cool}_w(n)$ (right).

```
Routine Weight'(n, w)
 1: if w > 0 and n − w > 0
 2:     b[n] ← 0
 3:     b[n − w] ← 1
 4:     visit(b)
 5:     Weight'(n − 1, w)
 6:     b[n − w] ← 0
 7:     b[n] ← 1
 8:     Weight'(n − 1, w − 1)
 9: end if
```

```
Routine Weight(n, w)
 1: b ← array(0^{n−w}1^w)
 2: visit(b)
 3: Weight'(n, w)
```

Algorithm 2 Recursive algorithms for generating $\mathsf{cool}(n)$ (left) and $\mathsf{cool}^u_\ell(n)$ (middle) and the parity restricted versions $\mathsf{coooool}^u_\ell(n)$ and $\mathsf{coool}^u_\ell(n)$ (right).

```
Routine Binary(n)
 1: b ← array(0^n)
 2: for w ← 0, 1, ..., n−1
 3:     visit(b)
 4:     b[n − w] ← 1
 5: end for
 6: visit(b)
 7: for w ← n−1, n−2, ..., 1
 8:     b[n − w] ← 0
 9:     Weight'(n, w)
10: end for
```

```
Routine Range(n, l, u)
 1: b ← array(0^{n−l}1^l)
 2: for w ← l, l+1, ..., u−1
 3:     visit(b)
 4:     b[n − w] ← 1
 5: end for
 6: visit(b)
 7: Weight'(n, u)
 8: for w ← u, u−1, ..., l
 9:     b[n − w] ← 0
10:     Weight'(n, w)
11: end for
```

```
Routine Parity(n, l, u, p)
 1: if l mod 2 ≠ p
 2:     l = l + 1
 3: end if
 4: if u mod 2 ≠ p
 5:     u = u − 1
 6: end if
 7: b ← array(0^{n−l}1^l)
 8: for w ← l, l+2, ..., u−2
 9:     visit(b)
10:     b[n − w] ← 1
11:     b[n − w − 1] ← 1
12: end for
13: visit(b)
14: Weight'(n, u)
15: for w ← u−2, u−4, ..., l
16:     b[n − w] ← 0
17:     b[n − w − 1] ← 0
18:     Weight'(n, w)
19: end for
```

We mention that our new algorithms for generating $\mathsf{cool}'_w(n)$ and $\mathsf{cool}_w(n)$ are simpler than the corresponding recursive algorithms that first appeared in Section 3.1 of [20]. The improvement is due to the altered precondition for $\mathsf{Weight}'(n,w)$.

4.2 de Bruijn Sequence Algorithms

In this subsection we create the de Bruijn sequences from Section 3.5 by 'filtering' the visited strings from the previous subsection. Recall from Theorem 6 that $\mathrm{d}\mathbb{B}_\ell^u(n)$ can be created by selecting the aperiodic prefixes of every necklace in $\texttt{cooool}_\ell^{u+1}(n+1)$ or $\texttt{coooool}_\ell^{u+1}(n+1)$. Thus, a simple approach to creating $\mathrm{d}\mathbb{B}_\ell^u(n)$ is to call $\mathsf{Parity}(n+1, l, u, p)$ and only record the desired portion of each visited string. More specifically, if b is the visited binary string, then the number of bits we wish to record is given by the following function

$$D(b) = \begin{cases} 0 & \text{if } b \text{ is not a necklace} \\ k & \text{if } b \text{ is a necklace and its aperiodic prefix has length } k. \end{cases} \tag{11}$$

This function can be computed in O(n)-time for strings of length n by the work of Duval [5]. (When calling $\mathsf{Parity}(n+1, l, u+1, p)$ the binary strings to be tested will have length $n+1$.) We thank Joe Sawada for providing the succinct implementation of routine Duval in Algorithm 3.

Lemma 5 ([5]) *If b stores a binary string of length n, then* $\mathsf{Duval}(b)$ *computes $D(b)$ in worst-case O(n)-time.*

Algorithm 3 Routine for computing the function $D(b)$ from (11).

```
Routine Duval(b)
 1: n ← |b|
 2: k ← 1
 3: for i ← 2, 3, ..., n
 4:     if b[i − k] > b[i]
 5:         return 0
 6:     end if
 7:     if b[i − k] < b[i]
 8:         k ← i
 9:     end if
10: end for
11: if n mod k ≠ 0
12:     return 0
13: end if
14: return k
```

Theorem 8 *Successive bits of the de Bruijn sequences $\mathrm{d}\mathbb{B}_\ell^u(n)$ can be generated in amortized O(n)-time.*

Proof Implement the visit routine as follows

```
Routine visit(b)
  k ← Duval(b)
  Add b[1]b[2] ··· b[k] to the de Bruijn sequence.
```

The routine $\mathsf{Parity}(n+1, \ell, u+1, p)$ for the correct choice of the parity flag $p \in \{0,1\}$ will create the de Bruijn sequence by Theorem 6. When $0 < u < \ell < n$ and $u - \ell$ is odd, the $\mathsf{Parity}(n+1, \ell, u+1, p)$ routine generates a total of

$$\binom{\ell+1}{n+1}+\binom{\ell+3}{n+1}+\cdots+\binom{u}{n+1} = \binom{\ell}{n}+\binom{\ell+1}{n}+\cdots+\binom{u-1}{n}+\binom{u}{n} \quad (12)$$

binary strings. Each string takes amortized $\mathrm{O}(n)$-time to generate due to the fact that Parity runs in constant amortized time and Duval takes linear time. Finally, observe that the number of bits in $\mathfrak{dB}^u_\ell(n)$ is equal to the number of strings generated by $\mathsf{Parity}(n+1, \ell, u+1, p)$ due to the equality in (12). The analysis is similar for the $\ell = 0$ and $u = n$ cases is similar. □

The run-time in Theorem 8 can be improved. Necklaces in $\mathsf{cool}_w(n)$ can be directly generated in constant amortized time [25]. This was used in Theorem 3 of [23] to prove that successive blocks of n bits of $\mathfrak{dB}^u_0(n)$ can be generated in constant amortized time. This approach can be taken for generating $\mathfrak{dB}^u_\ell(n)$ in general, although the authors enjoy the succinct approach described here.

4.3 Iterative Algorithms

We conclude our algorithmic section with iterative routines for generating the cool and cooler and coolest order in Algorithm 4. The routines are *loopless*, meaning that they visit each successive binary string in worst-case $\mathrm{O}(1)$-time, where the hidden constant in $\mathrm{O}(1)$ is independent of the length of the binary strings. For efficiency reasons the routines generate reflected versions of cool-lex order. (See Section 3 of [4] for a discussion of this efficiency issue in the context of a similar routine.)

The first routine $\mathsf{Weight}(n, w)$ generates $\mathsf{cool}_w(n)$ and forms the basis of the other two routines. It is nearly identical to the corresponding loopless algorithm in Section 3.2.2 of [20], and we refer the reader to that article for an understanding of $\mathsf{Weight}(n, w)$ and the simple extensions presented here. The routines each have one main loop and no other loops. Since the main loop visits one binary strings during each iteration, it is clear that the routines are loopless.

Theorem 9 ([20]) *The orders $\mathsf{cool}_w(n)$, $\mathsf{cool}(n)$, and $\mathsf{cool}^u_l(n)$ are generated in reflected order by the loopless routines in Algorithm 4.*

5 Cool Sublists

In this section we show that cool order shares several sublist properties with `lexicographic` order. There are several motivations for this investigation. First, the authors are interested in general conditions under which the necklace-prefix algorithm produces de Bruijn sequences, and the shared sublist properties could contribute to such a result. Second, the results help explain why

Algorithm 4 Iterative algorithms for generating $\texttt{cool}_w(n)$ (left), $\texttt{cool}(n)$ (middle), and $\texttt{cool}^u_\ell(n)$ (right) with spaces inserted to make identical commands appear on the same line.

```
Routine Weight(n, w)                Routine Binary(n)                   Routine Range(n, l, u)
                                                                          m ← min(u, n − 1)
b ← array(01^w 0^{n−w−1})           b ← array(01^{n−1})                 b ← array(01^m 0^{n−m−1})
x ← 2                               x ← 2                               x ← 2
y ← 1                               y ← 1                               y ← 1
visit(b)                            visit(b)                            visit(b)
while x ≤ n                         while y ≤ n                         while x ≠ y or y ≤ u
   b[x] ← 0                            b[x] ← 0                            b[x] ← 0
   b[y] ← 1                            b[y] ← 1                            b[y] ← 1
   x ← x + 1                           x ← x + 1                           x ← x + 1
   y ← y + 1                           y ← y + 1                           y ← y + 1
                                       if x=n+1 and y≠n+1                  if x=n+1 and y≠n+1
                                                                              if l = 0 and b[2] = 0
                                                                                 b[1] ← 0
                                                                              end if
                                                                              if b[l + 1] = 0
                                                                                 y ← l + 1
                                                                                 x ← l + 1
                                                                              else
                                          b[1] ← 0                               b[1] ← 0
                                          y ← 1                                  y ← 1
                                          x ← 1 + b[2]                           x ← 2
                                                                              end if
   if x ≤ n and b[x] = 0            elif x ≠ y and b[x] = 0             elif x ≠ y and b[x] = 0
      b[x] ← 1                         b[x] ← 1                            b[x] ← 1
      b[1] ← 0                         b[1] ← 0                            b[1] ← 0
      if y > 2                         if y > 2                            if y > 2
         x ← 2                            x ← 2                               x ← 2
      end if                           end if                              end if
      y ← 1                            y ← 1                               y ← 1
   end if                           end if                              end if
   visit(b)                         visit(b)                            visit(b)
end while                        end while                           end while
```

sublists of **cool** order yield Gray codes for so many combinatorial objects [18]. Third, it explains the visual similarity between $\texttt{cool}_w(n)$ and $\texttt{lexicographic}_w(n)$ when they are drawn "inside-out" with respect to each other, as illustrated by Figures 5 and 6. Finally, it deepens the connections between **cool** and `lexicographic` that are discussed in [20].

Given a list of strings $\mathcal{L}$, let $\text{prefix}_x(\mathcal{L})$ be the sublist of strings beginning with $x \in \{0, 1\}$, except that the leading x is removed. Similarly, $\text{suffix}_x(\mathcal{L})$ is the sublist of strings beginning with x, with the trailing x removed. In both cases, the relative order of the remaining strings is unchanged. We say that $\text{prefix}_x(\mathcal{L})$ and $\text{suffix}_x(\mathcal{L})$ are *prefix sublists* and *suffix sublists* of $\mathcal{L}$. Informally, we say that the prefix and suffix sublists are *recursive* if they provide smaller versions of a given order.

The ultimate order for recursive sublists is `lexicographic` order, as illustrated by Table 6. Recursive formulae for the `lexicographic` order of $\mathbf{B}(n)$

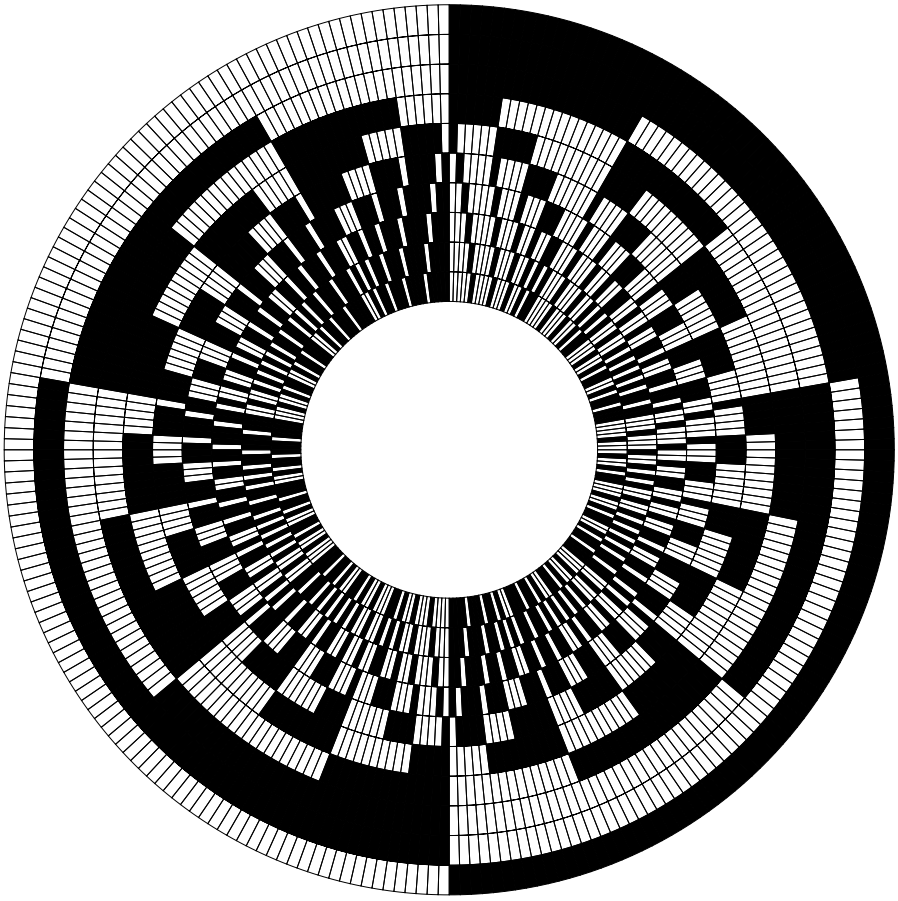


Fig. 5 An illustration of `lexicographic` order for $\mathbf{B}_5(10)$. Each string radiates inwards to the center, and the order proceeds clockwise starting just after 12 o'clock.

and $\mathbf{B}_w(n)$ are below

$$\begin{aligned}\texttt{lexicographic}(n) &= 0{\cdot}\texttt{lexicographic}(n{-}1),\ 1{\cdot}\texttt{lexicographic}(n{-}1)\\ \texttt{lexicographic}_w(n) &= 0{\cdot}\texttt{lexicographic}_w(n{-}1),\ 1{\cdot}\texttt{lexicographic}_{w-1}(n{-}1)\end{aligned}$$

with base cases of $\texttt{lexicographic}(1) = 0,\ 1$ and $\texttt{lexicographic}_0(n) = 0^n$ and $\texttt{lexicographic}_n(n) = 1^n$ for all $n \geq 1$. From these definitions it is obvious that `lexicographic` has recursive prefix sublists. That is,

$$\text{prefix}_x(\texttt{lexicographic}(n)) = \texttt{lexicographic}(n{-}1).$$

It can be proven by a simple induction that its suffix sublists are also recursive,

$$\text{suffix}_x(\texttt{lexicographic}(n)) = \texttt{lexicographic}(n{-}1).$$

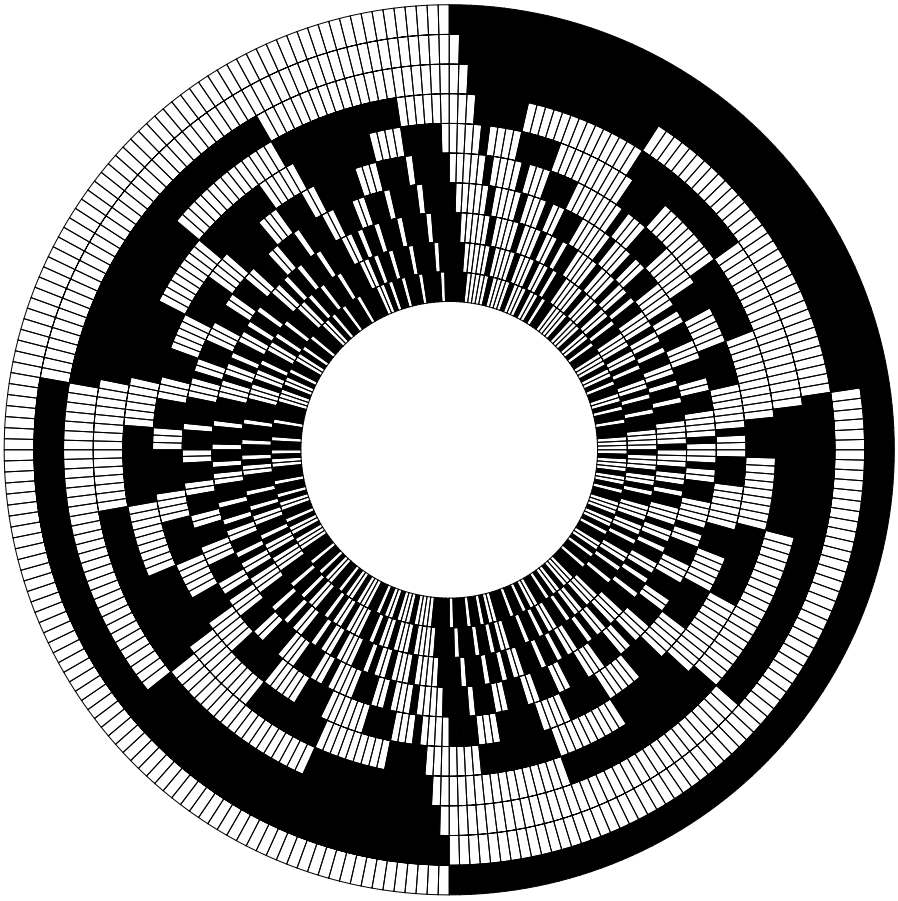


Fig. 6 An illustration of **cool** order for $\mathbf{B}_5(10)$. Each string radiates outwards from the center, and the order proceeds clockwise starting just before 12 o'clock.

Similarly, the `lexicographic` prefix and suffix sublists are recursive for $\mathbf{B}_w(n)$,

$$\begin{aligned}\text{prefix}_x(\texttt{lexicographic}_w(n)) &= \texttt{lexicographic}_{w-x}(n-1)\\ \text{suffix}_x(\texttt{lexicographic}_w(n)) &= \texttt{lexicographic}_{w-x}(n-1).\end{aligned}$$

Note: The cases (i) $x = 0$ and $w = n$ and (ii) $x = 1$ and $w = 0$ are excluded above, since $\mathbf{B}_n(n-1)$ and $\mathbf{B}_{-1}(n-1)$ are undefined, respectively.

Now we explore the prefix and suffix sublists of **cool** order, which are illustrated by Table 6. In each of the following results we implicitly use (2) and (3), and we ignore cases (i) and (ii) mentioned above. Our first result shows that **cool** order has recursive suffix sublists.

Theorem 10 *The following suffix sublist equality holds for* $x \in \{0, 1\}$

$$\text{suffix}_x(\textbf{cool}_w(n)) = \textbf{cool}_{w-x}(n-1).$$

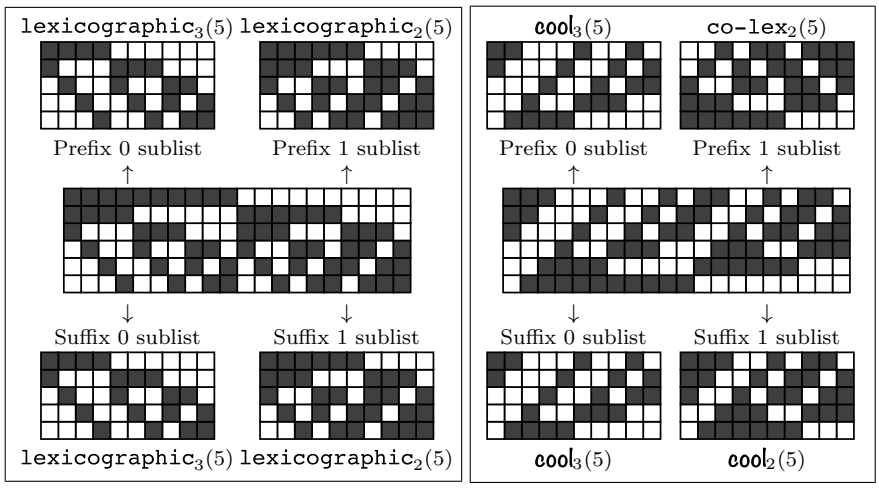


Table 6 The sublists of $\texttt{lexicographic}_3(6)$ (left) and $\textsf{cool}_3(6)$ (right). Each sublist is recursive, except one. The sublist of **cool** order with prefix 1 is the co-lexicographic order denoted as `co-lex` order, which is the same as `lexicographic` order except the individual strings are read right-to-left (which is bottom-up using the above diagram).

Proof First consider **cool** order's suffix sublist for $x = 0$. There are two base cases. First, if $w = 0$, then $\textsf{cool}_0(n) = 0^n$ for all $n > 0$, and so $\text{suffix}_0(\textsf{cool}_0(n)) = 0^{n-1} = \textsf{cool}_0(n-1)$ as desired. Second, if $w = n - 1$, then

$$\textsf{cool}_{n-1}(n) = 01^{n-1}, 1^{n-1}0, 1^{n-2}01, \ldots, 101^{n-2},$$

and so $\text{suffix}_0(\textsf{cool}_{n-1}(n)) = 1^{n-1} = \textsf{cool}_{n-1}(n-1)$ as desired. Otherwise, we can assume $0 < w < n$ and the following derivation proves the result by induction

$$\begin{aligned}
&\text{suffix}_0(\textsf{cool}_w(n)) \\
&= \text{suffix}_0(0^{n-w}1^w,\ \textsf{cool}'_w(n)) \\
&= \text{suffix}_0(\textsf{cool}'_w(n)) \\
&= \text{suffix}_0(0^{n-w-1}1^w0,\ \textsf{cool}'_w(n-1)\cdot 0,\ \textsf{cool}'_{w-1}(n-1)\cdot 1) \\
&= 0^{n-w-1}1^w,\ \text{suffix}_0(\textsf{cool}'_w(n-1)\cdot 0),\ \text{suffix}_0(\textsf{cool}'_{w-1}(n-1)\cdot 1) \\
&= 0^{n-w-1}1^w,\ \textsf{cool}'_w(n-1) \\
&= \textsf{cool}_w(n-1).
\end{aligned}$$

Next consider **cool** order's suffix sublist for $x = 1$. We initially prove that **cool**' order's suffix sublist for $x = 1$ is recursive. The base case of $w = n$ follows from the fact that $\textsf{cool}'_n(n) = \epsilon$. Otherwise, $0 < w < n$ and the following

derivation completes our initial proof

$$\begin{aligned}
&\mathrm{suffix}_1(\mathsf{cool}'_w(n)) \\
&= \mathrm{suffix}_1(0^{n-w-1}1^w0,\ \mathsf{cool}'_w(n-1)\cdot 0,\ \mathsf{cool}'_{w-1}(n-1)\cdot 1) \\
&= \mathrm{suffix}_1(\mathsf{cool}'_{w-1}(n-1)\cdot 1) \\
&= \mathsf{cool}'_{w-1}(n-1)
\end{aligned}$$

Now we prove the desired for cool as follows

$$\begin{aligned}
\mathrm{suffix}_1(\mathsf{cool}_w(n)) &= \mathrm{suffix}_1(0^{n-w}1^w,\ \mathsf{cool}'_w(n)) \\
&= \mathrm{suffix}_1(0^{n-w}1^w),\ \mathrm{suffix}_1(\mathsf{cool}'_w(n)) \\
&= 0^{n-w}1^{w-1},\ \mathsf{cool}'_{w-1}(n-1) \\
&= \mathsf{cool}_{w-1}(n-1). \qquad \square
\end{aligned}$$

Next we show that cool order has recursive prefix sublists for the symbol 0.

Theorem 11 *The following prefix sublist equality holds*

$$\mathrm{prefix}_0(\mathsf{cool}_w(n)) = \mathsf{cool}_w(n-1).$$

Proof Consider cool order's prefix sublist for $x = 0$. We initially prove that cool' order's prefix sublist for $x = 0$ is recursive. There are two base cases. First, the base case of $w = 0$ follows from the fact that $\mathsf{cool}'_0(n) = \epsilon$. Second, the base case of $w = n-1$ follows from

$$\mathsf{cool}'_{n-1}(n) = 1^{n-1}0, 1^{n-2}01, \ldots, 101^{n-2},$$

and so $\mathrm{prefix}_0(\mathsf{cool}'_{n-1}(n)) = \epsilon = \mathsf{cool}'_{n-1}(n-1)$ as desired. Otherwise, we assume $0 < w < n-1$ and the following derivation completes our initial proof

$$\begin{aligned}
&\mathrm{prefix}_0(\mathsf{cool}'_w(n)) \\
&= \mathrm{prefix}_0(0^{n-w-1}1^w0,\ \mathsf{cool}'_w(n-1)\cdot 0,\ \mathsf{cool}'_{w-1}(n-1)\cdot 1) \\
&= \mathrm{prefix}_0(0^{n-w-1}1^w0),\ \mathrm{prefix}_0(\mathsf{cool}'_w(n-1)\cdot 0),\ \mathrm{prefix}_0(\mathsf{cool}'_{w-1}(n-1)\cdot 1) \\
&= 0^{n-w-2}1^w0,\ \mathrm{prefix}_0(\mathsf{cool}'_w(n-1))\cdot 0,\ \mathrm{prefix}_0(\mathsf{cool}'_{w-1}(n-1))\cdot 1 \\
&= 0^{n-w-2}1^w0,\ \mathsf{cool}'_w(n-1)\cdot 0,\ \mathsf{cool}'_{w-1}(n-1)\cdot 1 \\
&= \mathsf{cool}'_w(n-1).
\end{aligned}$$

Now we prove the desired for cool as follows

$$\begin{aligned}
\mathrm{prefix}_0(\mathsf{cool}_w(n)) &= \mathrm{prefix}_0(0^{n-w}1^w,\ \mathsf{cool}'_w(n)) \\
&= \mathrm{prefix}_0(0^{n-w}1^w),\ \mathrm{prefix}_0(\mathsf{cool}'_w(n)) \\
&= 0^{n-w-1}1^w,\ \mathsf{cool}'_w(n-1) \\
&= \mathsf{cool}_w(n-1). \qquad \square
\end{aligned}$$

Thus far, we have shown that cool has recursive sublists, except possibly for the prefix sublist with $x = 1$. Interestingly, this sublist turns out to be a simple variant of $\texttt{lexicographic}_w(n)$ known as *co-lexicographic* order. The order is identical to `lexicographic` order except that the individual strings are read from right-to-left instead of left-to-right. We denote the order as `co-lex` order, and it is defined recursively as follows

$$\texttt{co-lex}_w(n) = \texttt{co-lex}_w(n-1) \cdot 0,\ \texttt{co-lex}_{w-1}(n-1) \cdot 1$$

with base case $\texttt{co-lex}_0(n) = 0^n$ and $\texttt{co-lex}_n(n) = 1^n$ for all $n \geq 1$.

Theorem 12 *The following prefix sublist equality holds*

$$\text{prefix}_1(\textsf{cool}_w(n)) = \texttt{co-lex}_{w-1}(n-1).$$

Proof We initially prove that cool' order's prefix sublist for $x = 1$ gives co-lexicographic order. There are two base cases. First, the base case of $w = 0$ follows from the fact that $\textsf{cool}'_0(n) = \epsilon$. Second, the base case of $w = n - 1$ follows from

$$\textsf{cool}'_{n-1}(n) = 1^{n-1}0, 1^{n-2}01, \ldots, 101^{n-2},$$

so $\text{prefix}_1(\textsf{cool}'_{n-1}(n)) = 1^{n-2}0, 1^{n-3}01, \ldots, 01^{n-2} = \texttt{co-lex}_{n-2}(n-1)$ as desired. Otherwise, we assume $0 < w < n - 1$ and the following derivation completes our initial proof

$$\begin{aligned}
&\text{prefix}_1(\textsf{cool}'_w(n)) \\
&= \text{prefix}_1(0^{n-w-1}1^w0,\ \textsf{cool}'_w(n-1) \cdot 0,\ \textsf{cool}'_{w-1}(n-1) \cdot 1) \\
&= \text{prefix}_1(\textsf{cool}'_w(n-1)) \cdot 0,\ \text{prefix}_1(\textsf{cool}'_{w-1}(n-1)) \cdot 1 \\
&= \texttt{co-lex}_{w-1}(n-2) \cdot 0,\ \texttt{co-lex}_{w-2}(n-2) \cdot 1 \\
&= \texttt{co-lex}_{w-1}(n-1).
\end{aligned}$$

Now we prove the desired for cool as follows

$$\begin{aligned}
\text{prefix}_1(\textsf{cool}_w(n)) &= \text{prefix}_1(0^{n-w}1^w,\ \textsf{cool}'_w(n)) \\
&= \text{prefix}_1(\textsf{cool}'_w(n)) \\
&= \texttt{co-lex}_{w-1}(n-1). \qquad \square
\end{aligned}$$

We complete this section by considering the prefix and suffix sublists of cooler order. We show that the suffix sublists are recursive for $x = 1$ and the prefix sublists are recursive for $x = 0$. In the proof of the theorem we implicitly use (4). The recursive sublist properties for `lexicographic` and cooler order are illustrated in Table 7.

Theorem 13 *The following prefix sublist equalities hold*

$$\text{suffix}_1(\textsf{cool}(n)) = \textsf{cool}(n-1) \text{ and } \text{prefix}_0(\textsf{cool}(n)) = \textsf{cool}(n-1).$$

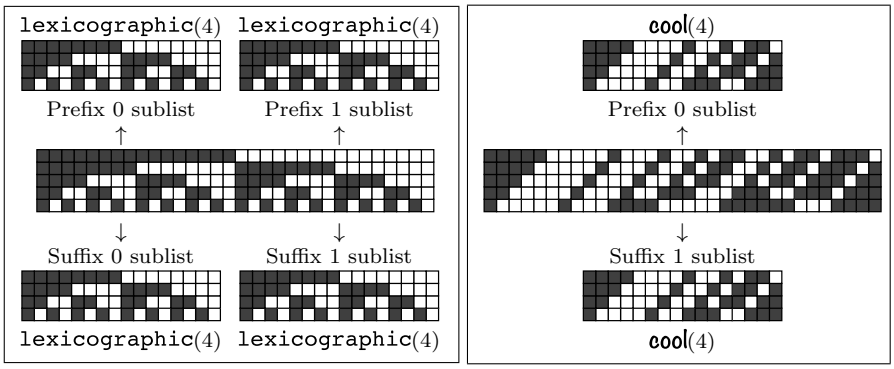


Table 7 The sublists of `lexicographic`(5) (left) and **cool**(5) (right). Each sublist is recursive in `lexicographic` order, while two of the sublists are recursive in **cool**er order.

Proof Consider **cool**er order's suffix sublist for $x = 1$. Using Theorem 13, we prove the sublist is recursive by the following derivation

$$\begin{aligned}
&\text{suffix}_1(\textbf{cool}(n)) \\
&= \text{suffix}_1(0^n,\ 0^{n-1}1,\ \ldots,\ 1^n, \textbf{cool}'_{n-1}(n),\ \ldots,\ \textbf{cool}'_1(n)) \\
&= 0^{n-1}, 0^{n-2}1,\ \ldots,\ 1^{n-1}, \text{suffix}_1(\textbf{cool}'_{n-1}(n)),\ \ldots,\ \text{suffix}_1(\textbf{cool}'_1(n)) \\
&= 0^{n-1}, 0^{n-2}1,\ \ldots,\ 1^{n-1}, \textbf{cool}'_{n-2}(n-1),\ \textbf{cool}'_{n-3}(n-1),\ \ldots,\ \textbf{cool}'_1(n-1) \\
&= \textbf{cool}(n-1).
\end{aligned}$$

Next consider **cool**er order's prefix sublist for $x = 0$. Using Theorem 12, we prove the sublist is recursive by the following derivation

$$\begin{aligned}
&\text{prefix}_0(\textbf{cool}(n)) \\
&= \text{prefix}_0(0^n,\ 0^{n-1}1,\ \ldots,\ 1^n, \textbf{cool}'_{n-1}(n),\ \ldots,\ \textbf{cool}'_1(n)) \\
&= 0^{n-1}, 0^{n-2}1,\ \ldots,\ 1^{n-1},\ \text{prefix}_0(\textbf{cool}'_{n-1}(n)),\ \ldots,\ \text{prefix}_0(\textbf{cool}'_1(n)) \\
&= 0^{n-1}, 0^{n-2}1,\ \ldots,\ 1^{n-1}, \textbf{cool}'_{n-2}(n-1),\ \ldots,\ \textbf{cool}'_1(n-1) \\
&= \textbf{cool}(n-1). \qquad \square
\end{aligned}$$

6 Concluding Remarks

We have presented a fun and cool new order for the binary strings of length n with weight at least ℓ and weight at most u, and have used this order to construct de Bruijn sequences for various weight-ranges. Algorithmically, we have given loopless algorithms for generating the weight-range binary strings, and simple O(n)-time algorithms for generating successive bits in the de Bruijn sequences. We have also investigated sublist properties of **cool** and **cool**est order, showing that they are similar to those found in `lexicographic` order.

There are various aspects of **cool**er and **cool**est order that can be further investigated. For example, Knuth [14] created a computer word algorithm

for generating the cool order of $\mathbf{B}_w(n)$, which is now distributed with his 64-bit architecture MMIX (also see [20]). Is there a simple modification that will generate the cooler order of $\mathbf{B}(n)$? As another example, *ranking* algorithms for $\mathsf{cool}_w(n)$ were given in [20] and could be generalized to $\mathsf{cool}^u_\ell(n)$. Finally, several of the Gray codes for binary bubble languages [18] could have natural generalizations using our cooler order.

There are at least two avenues for generalizing the results of this article. First, one could increase the number of distinct symbols in each string beyond $\{0, 1\}$. There is a natural generalization of cool order from $\mathbf{B}_w(n)$ to multiset permutations by Williams [27] and this order also yields a generalization of dual-weight de Bruijn sequences [28]. However, the authors are not sure how to generalize this order from multiset permutations to tuples in the 'coolest' way. Second, one could increase the number of dimensions of the strings. For example, one could create all binary x by y rectangular grids by coiling the strings of $\mathsf{cool}(n)$ with $n = xy$ around the rectangle. However, this Gray code is not particularly 'cool'. Ideally, there would be a Gray code that would yield de Bruijn tori using a two-dimensional version of the necklace-prefix algorithm (see [10] for a discussion of the de Bruijn torus problem).

The authors are also interested in the existence of a mechanical game or puzzle using $\mathsf{cool}(n)$. *Spin-Out* by Thinkfun is an example of a fun puzzle that uses $\mathsf{Gray}(n)$.

The authors thank Joe Sawada for his contribution to Section 4.2.

References

1. J. Berstel and D. Perrin. The origins of combinatorics on words. *European Journal of Combinatorics*, 28:996–1022, 2007.
2. P.J. Chase. Combination generation and graylex ordering. *Congressus Numerantium*, 69(19):215–242, 1989.
3. N.G. de Bruijn. A combinatorial problem. *Koninkl. Nederl. Acad. Wetensch. Proc. Ser A*, 49:758–764, 1946.
4. S. Durocher, P. C. Li, D. Mondal, F. Ruskey, and A. Williams. Cool-lex order and k-ary Catalan structures. *Journal of Discrete Algorithms*, 16:287–307, 2012.
5. J.P. Duval. Génération d'une section de classes de conjugasion et arbre des mots de Lyndon de lonueur bornée. *Theoretical Computer Science*, 60:255–283, 1988.
6. P. Eades and B. McKay. An algorithm for generating subsets of fixed size with a strong minimal change property. *Information Processing Letters*, 19:131–133, 1984.
7. H. Fredericksen and I. J. Kessler. An algorithm for generating necklaces of beads in two colors. *Discrete Mathematics*, 61:181–188, 1986.
8. H. Fredericksen and J. Maiorana. Necklaces of beads in k colors and $k$ary de Bruijn sequences. *Discrete Mathematics*, 23(3):207–210, 1978.
9. F. Gray. Pulse code communication. *U.S. Patent 2,632,058*, 1947.
10. G. Hurlbert and G. Isaak. On the de Bruijn torus problem. *J. Comb. Theory A*, 61(1):50–62, 1995.
11. R. J. Johnson. Long cycles in the middle two layers of the discrete cube. *J. Combin. Theory Ser. A*, 105(2):255–271, 2004.
12. D. E. Knuth. *The Art of Computer Programming*, volume 4 fascicle 2: Generating All Tuples and Permutations. Addison-Wesley, errata (updated 10/02/2008) edition, 2005. ISBN 0-201-85393-0.

13. D. E. Knuth. *The Art of Computer Programming*, volume 4 fascicle 3: Generating All Combinations and Partitions. Addison-Wesley, errata (updated 10/02/2008) edition, 2005. ISBN 0-201-85394-9.
14. D. E. Knuth. *The Art of Computer Programming*, volume 4: Combinatorial Algorithms, Part 1. Addison-Wesley, 2010.
15. M. H. Martin. A problem in arrangements. *Bull. Amer. Math. Soc.*, 40:859–864, 1934.
16. F. Ruskey. Adjacent interchange generation of combinations. *Journal of Algorithms*, 9(2):162–180, June 1988.
17. F. Ruskey, C. Savage, and T. Wang. Generating necklaces. *Journal of Algorithms*, 13:414–430, 1992.
18. F. Ruskey, J. Sawada, and A. Williams. Binary bubble languages and cool-lex Gray codes. *Journal of Combinatorial Theory, Series A*, 119(1):155–169, 2012.
19. F. Ruskey, J. Sawada, and A. Williams. De Bruijn sequences for fixed-weight binary strings. *SIAM Discrete Math*, 26(2):605–617, 2012.
20. F. Ruskey and A. Williams. The coolest way to generate combinations. *Discrete Mathematics*, 309(17):5305–5320, September 2009.
21. C. Flye Sainte-Marie. Solution to question nr. 48. *L'intermédiaire des Mathématiciens*, 1:107–110, 1894.
22. C. Savage and P. Winkler. Monotone Gray codes and the middle levels problem. *J. Combin. Theory Ser. A*, 70(2):230–248, 1995.
23. J. Sawada, B. Stevens, and A. Williams. De Bruijn sequences for the binary strings with a maximum density. In *WALCOM 2011: The 5th International Workshop on Algorithms and Computation*, volume 6552 of *Lecture Notes in Computer Science*, pages 182–190, New Dehli, India, 2011. Springer-Verlag.
24. J. Sawada and A. Williams. Efficient oracles for generating binary bubble languages. *Electronic Journal of Combinatorics*, 19:Paper 42, 20 pages, 2012.
25. J. Sawada and A. Williams. A Gray code for fixed-density necklaces and Lyndon words in constant amortized time. *Theoretical Computer Science*, DOI: 10.1016/j.tcs.2012.01.013, 2012, in press.
26. B. Stevens and A. Williams. The coolest order of binary strings. In E. Kranakis, D. Krizanc, and F. Luccio, editors, *FUN '12: Proceedings of the sixth international conference on Fun with Algorithms*, volume 7288 of *Lecture Notes in Computer Science*, pages 322–333, San Servolo Island, Venice, Italy, 2012. Springer.
27. A. Williams. Loopless generation of multiset permutations using a constant number of variables by prefix shifts. In *SODA '09: The Twentieth Annual ACM-SIAM Symposium on Discrete Algorithms*, New York, New York, USA, 2009.
28. A. Williams. *Shift Gray codes*. PhD thesis in Computer Science, University of Victoria, 2009.