Theoretical Informatics and Applications Will be set by the publisher
Informatique Théorique et Applications

RANDOM GENERATION OF UNIVERSAL CYCLES AND DE BRUIJN SEQUENCES

JOE SAWADA AND DANIEL GABRIĆ

Abstract. We present practical algorithms for generating universal cycles uniformly at random. In particular, we consider universal cycles for shorthand permutations, subsets and multiset permutations, weak orders, and orientable sequences. Additionally, we consider de Bruijn sequences, weight-range de Bruin sequences, and de Bruijn sequences, with forbidden 0^z substring. The algorithms apply a random walk of an underlying Eulerian shift graph to obtain a random arborescence (spanning in-tree). For each object, in order to seed the algorithm, a random edge is selected in the shift graph. Given the random arborescence and the shift graph, a corresponding random universal cycle can be generated in constant time per symbol. We provide experimental results on the average cover time required to compute a random arborescence for each object.

AMS Subject Classification. — Give AMS classification codes —.

1. INTRODUCTION

Let $\mathbf{\Sigma}_k(n)$ denote the set of all strings of length n over the alphabet $\{0, 1, \ldots, k-1\}$. Let $\mathbf{S}$ denote a subset of $\mathbf{\Sigma}_k(n)$. The *shift graph* of $\mathbf{S}$, denoted $G(\mathbf{S})$, is the directed graph where each vertex corresponds to a length-$(n-1)$ prefix or suffix of a string in $\mathbf{S}$, and there is a directed edge from vertex u to vertex v if $u = u_1u_2\cdots u_{n-1}$ and $v = u_2u_2\cdots u_n$. Each edge corresponds to a string $u_1u_2\cdots u_n$ in $\mathbf{S}$ and we label such an edge by u_n. For example, see Figure 1.

A *universal cycle* for $\mathbf{S}$, is a cyclic string of length $|\mathbf{S}|$ that contains each string in $\mathbf{S}$ as a substring exactly once (including the wraparound); they exist when $G(\mathbf{S})$ is Eulerian. In this paper, we are concerned with interesting subsets $\mathbf{S}$ whose underlying shift graph is Eulerian, i.e., $\mathbf{S}$ admits a universal cycle. In particular, we consider:

(1) (shorthand) permutations, subsets, and permutations of a multiset,

Keywords and phrases: random generation, universal cycle, de Bruijn sequence, weak order, subsets, permutations, orientable sequence

$G(\{001, 000, 011, 111, 110, 101\})$ $G(\{001, 010, 101, 011, 110, 100\})$

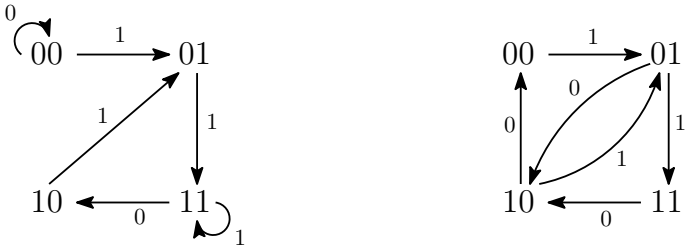

FIGURE 1. Two shift graphs. The one on the right is Eulerian.

(2) weak orders,
(3) k-ary strings of length n (which yield de Bruijn sequences),
(4) k-ary strings of length n that do not contain 0^z (cyclically),
(5) k-ary strings of length n with weight in the range $[a, b]$, and
(6) k-ary strings of length n that produce asymptotically optimal orientable sequences,

where the *weight* of a string is the sum of its symbols, and $[a, b]$ denotes the set of integers $\{a, a+1, \ldots, b\}$.

The primary objective of this paper is to describe time-efficient algorithms to generate universal cycles for these objects *uniformly at random*, while using exponential space to store their underlying shift graph. We describe a generic algorithm that generates a random Euler cycle in any directed Eulerian graph by first generating a random arborescence (spanning in-tree). For each set $\mathbf{S}$, we must be able to generate a random string in $\mathbf{S}$ in order to seed the algorithm by selecting a root for the arborescence. The generic algorithm is presented in Section 2. Then, in Section 3 we consider each of the aforementioned sets $\mathbf{S}$ and provide (i) a discussion of how to generate a random element from $\mathbf{S}$ and (ii) experimental evidence for the average cover time to compute a random arborescence in $G(\mathbf{S})$.

Despite the vast literature on universal cycle constructions, and in particular, de Bruijn sequences, we have found no discussion or resource regarding the generation of these sequences uniformly at random. This paper attempts to fill this void, even though it is well-known that a random arborescence in an Eulerian graph can be used to generate a random Euler cycle [14]. A recent result in [15] describes how to construct random de Bruijn sequences, although not uniformly at random.

2. ALGORITHM

A de Bruijn sequence of order n is in one-to-one correspondence with an Euler cycle in the de Bruijn graph $G(\mathbf{\Sigma}_k(n))$. Each Euler cycle corresponds to a unique rooted spanning in-tree (arborescence). By generating all such arborescences and considering all permutations for the remaining outgoing edges from each vertex, we can generate all Euler cycles in $G(\mathbf{\Sigma}_k(n))$, and hence all de Bruijn sequences; see, for instance, Chapter

7 in [16]. Furthermore, a de Bruijn sequence can be generated *uniformly at random* by generating a random arborescence and randomly ordering the remaining outgoing edges from each vertex. Similarly, we can generate universal cycles for $\mathbf{S}$ uniformly at random given the corresponding (Eulerian) shift graph $G(\mathbf{S})$ as follows:

Algorithm R

Generate a universal cycle for set $\mathbf{S}$ uniformly at random given the underlying (Eulerian) shift graph $G(\mathbf{S})$:

(1) Generate a random edge (r, v) in $G(\mathbf{S})$, i.e., a string in $\mathbf{S}$, to obtain a random root vertex r
(2) Generate a random arborescence T directed to root r
(3) Make each edge of T (the bridges) the last edge on the adjacency list of the corresponding outgoing vertex (the root does not have such a bridge), then randomly assign the order of the remaining outgoing edges
(4) Starting from r, traverse the edges in $G(\mathbf{S})$ (for $|\mathbf{S}|$ steps) by visiting the first unused edge in the current vertex's adjacency list

Since $G(\mathbf{S})$ is not necessarily regular, if we want T to be generated uniformly at random, then it is important to initialize the algorithm by selecting a random edge rather than a vertex. For Eulerian graphs, an arborescence can be generated uniformly at random by performing a random *backwards walk* until every vertex is visited. The first time a vertex is visited, the edge is recorded as a *tree edge* in the random arborescence [14]. The running time of this step depends on the *cover time* of the random walk, which is the number of steps it takes to visit every vertex. Thus, Algorithm R has exponential time delay due to steps (2) and (3) and requires exponential space to store $G(\mathbf{S})$. The final step (4) generates a random universal cycle for $\mathbf{S}$ in constant time per symbol.

Example 1 Consider the set $\mathbf{S}$ consisting of all binary strings of length $n = 6$ with weight in the range [1,2]. The shift graph $G(\mathbf{S})$ shown in Figure 2 illustrates a spanning in-tree (arborescence) rooted at 00010. Based on this spanning in-tree, the ordering of the adjacency lists for the vertices following Algorithm R, where the adjacency list of the root 00010 is randomly selected as 0,1, is as follows:

00000 →	1	01000 →	1,0	00110 →	0	10001 →	0
00001 →	1,0	10000 →	1,0	01001 →	0	10010 →	0
00010 →	0,1	00011 →	0	01010 →	0	10100 →	0
00100 →	1,0	00101 →	0	01100 →	0	11000 →	0.

The final step of Algorithm R produces the following random universal cycle for $\mathbf{S}$:

01000101000011000010.

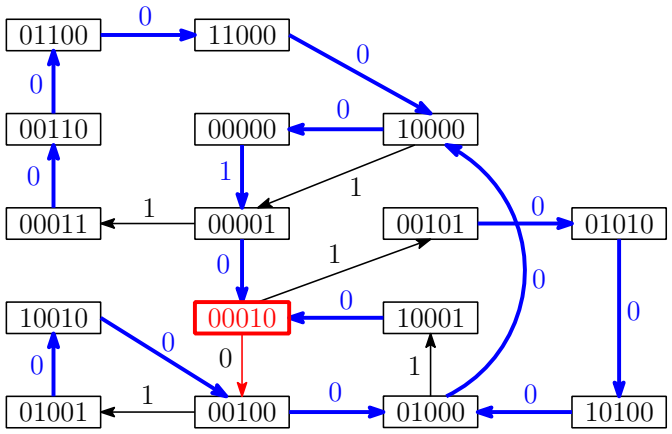

FIGURE 2. The shift graph $G(\mathbf{S})$ for the set $\mathbf{S}$ consisting of all binary strings of length $n = 6$ with weight in the range $[1, 2]$. Steps (1) and (2) of Algorithm R are illustrated where 000101 is selected as a random edge, and the bold (blue) edges highlight a spanning in-tree rooted at 00010.

Observe this is also a universal cycle for the 2-subsets from a ground set of size $n = 7$; for each length 6 substring add a 0 or 1 so the resulting length 7 string has exactly two 1s [17].

Arbitrary graphs have an $O(n^3)$ [8] and $\Omega(n \log n)$ [7] expected cover time, and these bounds are tight, as witnessed by the lollipop graph with n nodes and the complete graph with n nodes respectively. One class of graphs that are relevant to this paper are de Bruijn graphs, which are themselves regular graphs. A graph is regular if every node has the same degree and if the graph is directed, then the in-degrees and out-degrees of every node are equal. Regular graphs have an $O(n^2)$ expected cover time [9], as witnessed by the cycle graph with n nodes. However, the expected cover time for random regular graphs is $\Theta(n \log n)$ with high probability [5]. To the best of the authors' knowledge, the expected cover time of de Bruijn graphs remains unknown.

3. APPLICATIONS AND EXPERIMENTAL RESULTS

In this section we apply Algorithm R to generate universal cycles uniformly at random for shorthand permutations, subsets and multiset permutations, k-ary strings (de Bruijn sequences), generalizations of de Bruijn sequences including those with no 0^z substring and those with bounded weight, and orientable sequences.

For each object, we discuss how to generate a random edge in the underlying shift graph to seed the algorithm, and present experimental evidence for the cover time required to compute the random arborescence. Implementations of our algorithms are available at

http://debruijnsequence.org/db/random. In our implementations to compute the random arborescences, we did not pre-compute the shift-graphs, but instead used a mapping of each vertex to an integer (using a ranking algorithm or similar) to store whether a vertex had been visited. We applied a similar strategy to generate the random universal cycle in step (4) of Algorithm R.

3.1. PERMUTATIONS, SUBSETS, AND MULTISET PERMUTATIONS

Universal cycles do not exist, in general, for permutations and subsets. For permutations, however, observe that the final symbol is redundant. If $p_1p_2\cdots p_n$ is a permutation, we say that $p_1p_2\cdots p_{n-1}$ is a *shorthand permutation* of order n, where the last symbol is implied. Let $\mathbf{SP}(n)$ be the set of all shorthand permutations of order n. Similarly, if $b_1b_2\cdots b_n$ is a binary string with k ones (representing a k-subset of an n-set), we say that $b_1b_2\cdots b_{n-1}$ is a *shorthand k-subset* of order n, where the last bit is implied. Let $\mathbf{S}(n,k)$ be the set of all shorthand k-subsets of order n. Multiset permutations (strings with fixed content) generalize both permutations and subsets. If $m_1m_2\cdots m_n$ is a permutation of the multiset $\{s_1,s_2,\ldots,s_n\}$, then we say $m_1m_2\cdots m_{n-1}$ is a *shorthand multiset permutation*. When each $s_i = i$, a multiset permutation is simply a permutation, and when the multiset contains k ones and $(n-k)$ zeros, it represents a binary string with weight k representing a k-subset of an n-set.

For shorthand permutations, the underlying shift graph has $n!$ edges. Each vertex has in-degree = out-degree = 2 and thus there are $n!/2$ vertices. For shorthand k-subsets, where $k \geq 2$, the underlying shift graph has $\binom{n}{k}$ edges and each vertex is a binary string of length $n-2$ with weight $k-2$, $k-1$, or k; there are $\binom{n-2}{k-2} + \binom{n-2}{k-1} + \binom{n-2}{k}$ vertices.

A random multiset permutation $m_1m_2\cdots m_n$ can be generated in $O(n)$ time by applying the *Fisher-Yates shuffle*, as illustrated in Algorithm 1. Thus, $m_1m_2\cdots m_{n-1}$ is a random shorthand multiset permutation that can be used to seed Algorithm R for shorthand permutations, shorthand subsets, and more generally, shorthand multiset permutations.

Algorithm 1 Random generation of a permutation $m_1m_2\cdots m_n$ of the multiset $\{s_1,s_2,\ldots,s_n\}$ applying the Fisher-Yates shuffle

```
m_1 m_2 ··· m_n ← s_1 s_2 ··· s_n
for i from n down to 2 do
    j ← random integer in [1, i]
    SWAP(m_i, m_j)
```

Table 1 shows the minimum, maximum, and average ratios of the cover time to total edges in $G(\mathbf{SP}(n))$ and $G(\mathbf{S}(n,n/2))$ by running Algorithm R for 10,000 iterations.

Universal cycles for shorthand permutations can be constructed in $O(1)$-amortized time per symbol using $O(n)$ space [13, 18]. An $O(n)$-time successor rule is presented in [12], and an $O(1)$-amortized time per symbol algorithm applying concatenation trees is presented in [19] that uses $O(n^2)$ space. Universal cycles for shorthand subsets can be constructed in $O(1)$-amortized time per symbol using $O(n)$ space [17]. Universal cycles for shorthand multiset permutations (strings with fixed content) can be constructed in $O(1)$-amortized time per symbol using $O(n)$ space [20].

n	**Ratio**: cover time / $n!$ **Min**	**Max**	**Avg**
3	0.3	0.3	0.3
4	0.5	3.7	1.1
5	0.8	5.3	2.0
6	1.7	6.8	3.0
7	2.7	6.5	3.9
8	3.9	8.0	4.9
9	5.0	7.7	5.9
10	6.4	8.1	7.0

n	**Ratio**: cover time / $\binom{n}{k}$ **Min**	**Max**	**Avg**
10	1.8	18.9	5.0
12	3.2	18.0	6.3
14	4.6	16.4	7.6
16	5.9	18.8	8.8
18	7.2	20.3	10.1
20	8.5	21.8	11.4
22	9.7	21.8	12.7
24	11.2	22.6	14.1
26	12.4	24.0	15.4

TABLE 1. The minimum, maximum, and average ratios of the cover time to total edges in the shift graphs $G(\mathbf{SP}(n))$ (left) and $G(\mathbf{S}(n, n/2))$ (right) by running Algorithm R for 10,000 iterations.

n	**Ratio**: cover time / W_n **Min**	**Max**	**Avg**
3	0.5	9.3	1.8
4	1.1	12.7	3.7
5	2.5	12.9	5.4
6	4.1	16.7	7.2
7	6.4	21.3	9.3
8	8.7	18.5	11.5
9	12.0	15.4	13.8

TABLE 2. The minimum, maximum, and average ratios of the cover time to total edges in $G(\mathbf{W}(n))$ by running Algorithm R for 10,000 iterations.

3.2. WEAK ORDERS

A *weak order* is the number of ways n competitors can finish in a race if ties are allowed. Let $\mathbf{W}(n)$ denote the set of weak orders with n competitors, and let W_n denote $|\mathbf{W}(n)|$. For example,

$$\mathbf{W}(3) = \{111, 113, 131, 311, 122, 212, 221, 123, 132, 213, 231, 312, 321\},$$

and $W_3 = 13$. The number of weak orders where there is a k-way tie for first is given by $\binom{n}{k}W_{n-k}$ for $k < n$; there is 1 weak order when $k = n$. Thus, $W_n = \sum_{k=1}^{n} \binom{n}{k} W_{n-k}$. This recurrence together with the Fisher-Yates shuffle can be applied to generate a weak order $w_1 w_2 \cdots w_n$ uniformly at random, as illustrated in Algorithm 2.

Table 2 shows the minimum, maximum, and average ratios of the cover time to total edges in $G(\mathbf{W}(n))$ by running Algorithm R for 10,000 iterations.

Algorithm 2 Random generation of a weak order $w_1 w_2 \cdots w_n$.

$w_1 w_2 \cdots w_n \leftarrow 0^n$
$t \leftarrow n$
$v \leftarrow 1$
▷ Randomly place the element(s) v based on the recurrence
while $t \geq 1$ **do**
  $r \leftarrow$ random integer in $[1, W_t]$
  for j **from** 1 **to** t **do**
    $p_j \leftarrow \binom{t}{j} W_{t-j}$
    if $r \leq p_j$ **then break**
    $r \leftarrow r - p_j$
  ▷ Randomly place j occurrences of i into the t empty spots
  $b_1 \cdots b_t \leftarrow$ a random binary string with length t and weight j ▷ Apply Fisher-Yates shuffle
  $i \leftarrow 1$
  for k **from** 1 **to** n **do**
    if $w_k = 0$ **and** $b_i = 1$ **then** $w_k \leftarrow v$; $i \leftarrow i + 1$
  $v \leftarrow v + j$
  $t \leftarrow t - j$

Universal cycles for weak orders can be constructed via a successor rule that generates the sequence in $O(n)$ per symbol using $O(n)$ space [22]. By applying concatenation trees, they can be generated in $O(1)$-amortized time using $O(n^2)$ space [19]. See the enumeration sequence A000670 for W_n in the Online Encyclopedia of Integer Sequences [1].

3.3. DE BRUIJN SEQUENCES

For de Bruijn sequences, the underlying de Bruijn graph $G(\mathbf{\Sigma}_k(n))$ has k^{n-1} vertices and k^n edges. A random k-ary string (edge) can be computed in $O(n)$ time by generating a random symbol in $[0, k-1]$ n times. Table 3 and Table 4 show the minimum, maximum, and average ratios of the cover time to total edges in $G(\mathbf{\Sigma}_k(n))$, for $k = 2, 3, 4$ by running Algorithm R for 10,000 iterations.

If an application does not require a sequence generated uniformly at random, an algorithm which applies a Burrows-Wheeler transform can be applied; it outputs each de Bruijn sequence with positive probability [15]. The algorithm requires exponential space and has an exponent time delay with respect to the order n, but produces the sequence in $\alpha(n)$-amortized time per symbol for $k = 2$, where $\alpha(n)$ is the inverse Ackerman function. The first estimate on the mean *discrepancy* of de Bruijn sequences is obtained using this algorithm [15].

3.4. WEIGHT-RANGE DE BRUIJN SEQUENCES

Let $\mathbf{WR}_k(n, [\ell, u])$ denote the subset of strings in $\mathbf{\Sigma}_k(n)$ with weight in the range $[\ell, u]$. Let $WR_k(n, [\ell, u])$ denote $|\mathbf{WR}_k(n, [\ell, u])|$. It is straightforward to observe that $WR_k(n, [\ell, u]) = 0$ if $u < 0$ or $\ell > n(k-1)$; otherwise, if $n = 1$ then $WR_k(n, [\ell, u]) =$

n	**Ratio**: cover time / 2^n **Min**	**Max**	**Avg**
4	0.4	8.4	1.3
5	0.5	7.1	1.7
6	0.6	8.3	2.1
7	0.9	8.7	2.4
8	1.1	8.6	2.7
9	1.4	9.0	3.1
10	1.9	9.0	3.4
11	2.2	9.9	3.8
12	2.5	10.7	4.1
13	2.9	9.0	4.5
14	3.4	10.2	4.8
15	3.8	10.1	5.1

n	**Ratio**: cover time / 2^n **Min**	**Max**	**Avg**
16	4.1	10.4	5.5
17	4.5	9.8	5.9
18	4.9	9.1	6.2
19	5.3	9.8	6.5
20	5.8	9.8	6.9
21	6.1	9.4	7.3
22	6.6	10.4	7.8
23	7.0	9.2	8.0
24	7.7	11.6	9.1
25	7.6	10.5	8.5
26	8.1	10.7	8.9

TABLE 3. The minimum, maximum, and average ratios of the cover time to total edges in the shift graph $G(\mathbf{\Sigma}_2(n))$ by running Algorithm R for 10,000 iterations.

n	**Ratio**: cover time / 3^n **Min**	**Max**	**Avg**
3	0.3	3.9	0.9
4	0.5	4.5	1.3
5	0.7	4.6	1.7
6	1.0	5.6	2.0
7	1.5	5.1	2.4
8	1.8	5.6	2.8
9	2.2	6.3	3.1
10	2.6	6.0	3.5
11	3.0	6.1	3.9
12	3.4	6.3	4.3
13	4.0	6.4	4.6
14	4.3	6.3	4.9
15	4.6	6.6	5.3
16	5.0	6.7	5.7

n	**Ratio**: cover time / 4^n **Min**	**Max**	**Avg**
4	0.5	4.5	1.3
5	0.7	4.6	1.7
6	1.0	5.6	2.0
7	1.5	5.1	2.4
8	1.8	5.6	2.8
9	2.2	6.3	3.1
10	2.6	6.0	3.5
11	3.0	6.1	3.9
12	3.6	4.7	4.0
13	3.9	5.0	4.3

TABLE 4. The minimum, maximum, and average ratios of the cover time to total edges in the shift graph $G(\mathbf{\Sigma}_k(n))$ for $k = 3$ and $k = 4$ by running Algorithm R for 10,000 iterations.

$min(u, k-1) - max(\ell, 0) + 1$, and if $n > 1$:

$$WR_k(n, [\ell, u]) = \sum_{j=0}^{k-1} WR_k(n-1, [\ell-j, u-j]).$$

A *weight-range de Bruijn sequence* is a universal cycle for the set $\mathbf{WR}_k(n, [\ell, u])$. The shift graph $G(\mathbf{WR}_k(n, [\ell, u]))$ is generally not regular. A random edge $s_1 s_2 \cdots s_n$ can be generated using values for $WR_k(n, [\ell, u])$ as outlined in Algorithm 3. The algorithm essentially unranks a string in $\mathbf{WR}_k(n, [\ell, u])$ as it appears in lexicographic order.

Table 5 shows the minimum, maximum, and average ratios of the cover time to total edges in $G(\mathbf{WR}_2(n, [5, 10]))$ by running Algorithm R for 10,000 iterations.

Algorithm 3 Random generation of a string $s_1 s_2 \cdots s_n$ in $\mathbf{WR}_k(n, [\ell, u])$

$r \leftarrow$ random integer in $[1, WR_k(n, [\ell, u])]$
for j **from** 1 **to** $n-1$ **do**
  for i **from** 0 **to** $k-1$ **do** $n_i \leftarrow WR_k(n-j, [\ell-i, u-i])$
  $i \leftarrow 0$
  while $r > n_i$ **do** $r \leftarrow r - n_i$; $i \leftarrow i+1$
  $s_j \leftarrow i$
  $\ell \leftarrow \ell - i$; $u \leftarrow u - i$
if $\ell < 0$ **then** $\ell \leftarrow 0$
$s_n \leftarrow \ell + r - 1$

	Ratio: cover time / $WR_2(n, [5, 10])$		
n	**Min**	**Max**	**Avg**
10	2.3	19.4	5.1
11	2.5	15.9	5.5
12	3.2	13.0	6.0
13	3.9	15.6	6.5
14	4.2	15.8	7.2
15	5.2	14.7	7.9
16	6.0	16.0	8.6
17	6.7	15.2	9.3
18	7.4	17.0	10.2
19	8.1	17.5	10.8
20	9.1	17.2	11.5

TABLE 5. The minimum, maximum, and average ratios of the cover time to total edges in the shift graph $G(\mathbf{WR}_2(n, [5, 10]))$ by running Algorithm R for 10,000 iterations.

Weight-range de Bruijn sequences can be constructed via an $O(n)$ time per symbol successor rule when the minimum weight is 0, or the maximum weight is $(k-1)^n$ [GSWW20]. In the binary case, they can be constructed for any weight range in $O(1)$-amortized time [SWW13]. When $k = 2$ and $\ell + 1 = u$, weight-range de Bruijn sequences correspond to the universal cycles for (shorthand) subsets discussed in Section 3.1.

3.5. DE BRUIJN SEQUENCES WITH FORBIDDEN 0^z

A *necklace class* is an equivalence class of strings under rotation; we call the lexicographically smallest string in the class a *necklace*. The necklace class containing α is denoted $[\alpha]$. For example, if $\alpha = 0001$ then $[\alpha] = \{0001, 0010, 0100, 1000\}$. Let $\mathbf{N}_k(n, z)$ denote the set of all necklaces in $\mathbf{\Sigma}_k(n)$ with no 0^z substring for $z > 1$. All such necklaces end with 1 when $z \leq n$. Let $\mathbf{Z}_k(n, z) = \bigcup_{\alpha \in \mathbf{N}_k(n,z)} [\alpha]$. It is known that $\mathbf{Z}_k(n, z)$ admits a maximal length universal cycle that does not contain the substring 0^z [4]. We call a maximum length universal cycle that does not contain 0^z as a substring, a *de Bruijn sequence with forbidden* 0^z.

The shift graph $G(\mathbf{Z}_k(n, z))$ is not necessarily regular. A random edge can be generated by applying the following recurrences. Let $F_k(n, z)$ denote the number of k-ary strings of length n with no 0^z substring. It satisfies the following recurrence for $z < n$:

$$F_k(n, z) = (k-1) \sum_{j=1}^{z} F_k(n - j, z),$$

where $F_k(n, z) = k^n$ for $z > n$ and $F_k(n, n) = k^n - 1$.

Let $Z_k(n, z)$ denote the number of k-ary strings of length n with no 0^z substring, including the wraparound. It satisfies the following recurrence for $z < n$ obtained by partitioning the strings into those beginning with a non-zero, and those with j zeros in the wraparound, in which case there are $k-1$ possibilities for each of the first non-zero and last non-zero:

$$Z_k(n, z) = (k-1)F_k(n - 1, z) + (k-1)^2 \sum_{j=1}^{z-1} j \cdot F_k(n - j - 2, z),$$

where $Z_k(n, z) = k^n$ for $z > n$ and $Z_k(n, n) = k^n - 1$.

Given these recurrences, we can compute a random string in $\mathbf{F}_k(n, z)$ following a similar unranking strategy using lexicographic order as done with $\mathbf{WR}_k(n, [\ell, u])$ in the previous subsection. We omit the details in this case. Table 6 shows the minimum, maximum, and average ratios of the cover time to total edges in $G(\mathbf{F}_2(n, 2))$ and $G(\mathbf{F}_2(n, 3))$ by running Algorithm R for 10,000 iterations.

The lexicographically smallest de Bruijn sequences with forbidden 0^z can be generated via a simple greedy algorithm [21]; it can also be generated efficiently by concatenating the aperiodic prefixes of necklaces with no 0^z substring as they appear in lexicographic order [10, 21]. An exponential number of such sequences can be efficiently generated by applying cycle-joining as described in [4].

3.6. ORIENTABLE SEQUENCES

Recall the definitions of a necklace class and necklace from the previous subsection. A *bracelet class* is an equivalence class of strings under rotation and reversal; we call the

n	**Ratio**: cover time / $F_2(n,2)$ **Min**	**Max**	**Avg**
8	1.0	20.9	3.6
9	1.2	15.9	3.9
10	1.7	21.2	4.6
11	2.0	13.5	4.8
12	2.4	17.6	5.7
13	2.9	15.7	5.7
14	3.3	19.0	6.4
15	3.8	18.1	6.9
16	4.3	21.9	7.4
17	4.3	15.9	7.8
18	5.2	21.9	8.4
19	5.9	18.7	8.8
20	6.4	17.2	9.3
21	6.9	21.2	9.7
22	7.6	20.1	10.2
23	8.0	18.7	10.7
24	8.3	24.8	11.2

n	**Ratio**: cover time / $F_2(n,3)$ **Min**	**Max**	**Avg**
8	1.4	21.5	4.3
9	1.8	13.3	4.6
10	2.5	18.7	5.3
11	3.1	14.8	5.8
12	3.8	17.5	6.5
13	4.1	16.5	7.0
14	4.5	17.0	7.7
15	5.5	16.7	8.2
16	6.1	20.0	8.9
17	6.7	19.8	9.5
18	7.4	17.7	10.1
19	8.2	16.3	10.6
20	8.8	16.9	11.3
21	9.6	15.8	11.8
22	10.2	17.7	12.5
23	10.8	18.0	13.1
24	11.0	22.8	13.9

TABLE 6. The minimum, maximum, and average ratios of the cover time to total edges in the shift graphs $G(\mathbf{F}_2(n,2))$ (left) and $G(\mathbf{F}_2(n,3))$ (right) by running Algorithm R for 10,000 iterations.

lexicographically smallest string in the class a *bracelet*. A *bracelet* is said to be *asymmetric* if it is not in the same necklace class as its reversal. For example, 001011 is an asymmetric bracelet, but 001001 is not. Let $\mathbf{AB}_k(n)$ denote the set of all k-ary asymmetric bracelets of length n, and let $\mathbf{OS}_k(n) = \bigcup_{\alpha \in \mathbf{AB}_k(n)} [\alpha]$. Let $OS_k(n)$ denote $|\mathbf{OS}_k(n)|$.

For example, $\mathbf{AB}_2(7) = \{0001011, 0010111\}$, and

$$\begin{aligned}\mathbf{OS}_2(7) \quad = \quad & \{0001011, 0010110, 0101100, 1011000, 0110001, 1100010, 1000101\} \cup \\ & \{0010111, 0101110, 1011100, 0111001, 1110010, 1100101, 1001011\},\end{aligned}$$

where $OS_2(7) = 14$.

An *orientable sequence* is cyclic sequence such that each length-n substring occurs at most once in *either direction*. For example, a maximum-length orientable sequence for $n = 5$ and $k = 2$ is 001101. A universal cycle for $\mathbf{OS}_k(n)$ is known to be an orientable sequence with asymptotically optimal length [2, 3]. A formula for $OS_k(n)$, is given in [6, 11]. Generating a random string from $\mathbf{OS}_k(n)$ does not appear to be a trivial matter. However, by randomly generating k-ary strings with rejection, on average only two random strings need to be generated to obtain a string in $\mathbf{OS}_k(n)$ as n gets large. Thus, the expected time to generate a random edge in $G(\mathbf{OS}_k(n))$ is $\Theta(n)$.

	Ratio: cover time / $OS_2(n)$		
n	**Min**	**Max**	**Avg**
6	0.8	0.8	0.8
7	0.9	8.5	1.5
8	0.9	18.8	3.2
9	1.3	13.2	4.3
10	2.1	14.7	5.0
11	2.7	18.4	5.7
12	3.6	16.3	6.5
13	4.2	17.0	7.2
14	5.1	17.6	7.9
15	5.3	18.0	8.5
16	6.2	18.9	9.2
17	7.0	18.4	9.9
18	7.8	18.6	10.6
19	8.6	17.7	11.2
20	9.5	17.7	11.8

TABLE 7. The minimum, maximum, and average ratios of the cover time to total edges in the shift graph $G(\mathbf{OS}_2(n))$ by running Algorithm R for 10,000 iterations.

Table 7 illustrates the minimum, maximum, and average ratios of the cover time to total edges in $G(\mathbf{OS}_2(n))$ by running Algorithm R for 10,000 iterations.

Orientable sequences with asymptotically optimal length can be constructed in $O(n)$-time per symbol using $O(n)$ space [11]; in the binary case, they can be constructed in $O(1)$-amortized time per bit using $O(n^2)$ space.

REFERENCES

[1] OEIS Foundation Inc. (2025), Entry A000670 in The On-Line Encyclopedia of Integer Sequences, https://oeis.org/A000670.

[2] ALHAKIM, A., MITCHELL, C. J., SZMIDT, J., AND WILD, P. R. Orientable sequences over non-binary alphabets. In *Cryptography and Communications (to appear)* (2024).

[3] BURNS, J., AND MITCHELL, C. Position sensing coding schemes. In *Cryptography and Coding III (M.J.Ganley, ed.)* (1993), Oxford University Press, pp. 31–66.

[4] CHEE, Y. M., ETZION, T., NGUYEN, T. L., TA, D. H., TRAN, V. D., AND VU, V. K. Maximum length RLL sequences in de Bruijn graph, arXiv preprint arXiv:2403.01454, 2024.

[5] COOPER, C., AND FRIEZE, A. The cover time of random regular graphs. *SIAM Journal on Discrete Mathematics 18*, 4 (2005), 728–740.

[6] DAI, Z.-D., MARTIN, K., ROBSHAW, B., AND WILD, P. Orientable sequences. In *Cryptography and Coding III (M.J.Ganley, ed.)* (1993), Oxford University Press, pp. 97–115.

[7] FEIGE, U. A tight lower bound on the cover time for random walks on graphs. *Random Structures & Algorithms 6*, 4 (1995), 433–438.

[8] FEIGE, U. A tight upper bound on the cover time for random walks on graphs. *Random Structures & Algorithms 6*, 1 (1995), 51–54.

[9] FEIGE, U. Collecting coupons on trees, and the cover time of random walks. *computational complexity 6*, 4 (1996), 341–356.
[10] GABRIĆ, D., AND SAWADA, J. Constructing de Bruijn sequences by concatenating smaller universal cycles. *Theoret. Comput. Sci. 743* (2018), 12–22.
[11] GABRIĆ, D., AND SAWADA, J. Constructing k-ary orientable sequences with asymptotically optimal length. *Designs, Codes and Cryptography* (Feb 2025).
[12] GABRIĆ, 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.
[13] HOLROYD, A. E., RUSKEY, F., AND WILLIAMS, A. Shorthand universal cycles for permutations. *Algorithmica 64*, 2 (2012), 215–245.
[14] KANDEL, D., MATIAS, Y., UNGER, R., AND WINKLER, P. Shuffling biological sequences. *Discrete Applied Mathematics 71*, 1 (1996), 171–185.
[15] LIPTÁK, Z., AND PARMIGIANI, L. A BWT-based algorithm for random de Bruijn sequence construction. In *LATIN 2024, LNCS 14578* (2024), pp. 130–145.
[16] RUSKEY, F. Combinatorial generation. *Working Version (1j-CSC 425/520)* (2003).
[17] RUSKEY, F., SAWADA, J., AND WILLIAMS, A. De Bruijn sequences for fixed-weight binary strings. *SIAM J. Discrete Math. 26*, 2 (2012), 605–617.
[18] RUSKEY, F., AND WILLIAMS, A. An explicit universal cycle for the (n-1)-permutations of an n-set. *ACM Trans. Algorithms 6*, 3 (July 2010), 1–12.
[19] SAWADA, J., SEARS, J., TRAUTRIM, A., AND WILLIAMS, A. Concatenation trees: A framework for efficient universal cycle and de Bruijn sequence constructions. *arXiv preprint arXiv:2308.12405* (2024).
[20] SAWADA, J., AND WILLIAMS, A. A universal cycle for strings with fixed-content. *Manuscript* (2021).
[21] SAWADA, J., WILLIAMS, A., AND WONG, D. Generalizing the classic greedy and necklace constructions of de Bruijn sequences and universal cycles. *Electron. J. Combin. 23*, 1 (2016), Paper 1.24, 20.
[22] SAWADA, J., AND WONG, D. Efficient universal cycle constructions for weak orders. *Discrete Mathematics 343*, 10 (2020), 112022.

Communicated by (The editor will be set by the publisher).
(The dates will be set by the publisher).