Enumeration of Unbordered Words in Compressed Representation

Che-Wei Tsao* Yi-Hua Lin* Wing-Kai Hon* Dominik Köppl†

*National Tsing Hua University
Hsinchu, Taiwan

†The University of Yamanashi
Kofu, Japan

Abstract

We show that the enumeration of unbordered words can be done with amortized $O(1)$ time per reported word via a compressed representation of the reported words. More precisely, we represent unbordered words as cyclic shifts of Lyndon words, and enumerate all words corresponding to the same Lyndon word together. While storing explicitly a Lyndon word of length n with k unbordered cyclic shifts requires $kn \lg \sigma$ bits, the space can be reduced to $n \lg \sigma + n$ bits by storing the Lyndon word explicitly and the cyclic shifts *implicitly*, where σ denotes the size of an ordered finite alphabet. The time complexity stems from a linear-time algorithm to compute the border correlation function that specifies which cyclic shifts of a Lyndon word correspond to unbordered words. The experimental results show that the compression ratio is approximately $1/n$ for large values of n and σ (when $n \geq 8$ and $\sigma \geq 10$). In addition, the algorithm can also be used to randomly generate at least one unbordered word in expected $O(n)$ time, which is faster than the $O(n^2/\lg n)$-time method in literature.

1 Introduction

A word is called *bordered* if it has a nonempty proper prefix that is equal to a nonempty proper suffix; otherwise, it is called *unbordered* [1]. Bordered and unbordered words play an important role in combinatorics on words and have been studied in connection with various applications [2–5]. As noted by Gabric [6], combinatorial objects typically give rise to three fundamental problems: counting, enumeration, and random generation. In this paper, we focus on the enumeration and random generation of unbordered words.

In 1973, Nielsen proposed an efficient method to count and enumerate unbordered words [1], where the enumeration requires $O(n)$ time to generate the next unbordered word of length at most n. However, since the number of unbordered words increases exponentially (by $\Theta(\sigma^n)$ [1], where σ denotes the alphabet size), storing or transmitting all of them may demand excessive space. To address this issue, we propose an alternative method that enumerates unbordered words in a compressed representation using approximately $1/n$ of the explicit space. Moreover, our method takes $O(1)$ amortized time and $O(n)$ worst-case time per word, in contrast to the $O(n)$ per-word delay of the Nielsen's algorithm[1].

Regarding random generation, Gabric proposed an unrank algorithm for unbordered words with time complexity $O(\sigma n^4 \lg \sigma)$ [6]. Subsequently, Radoszewski et al. improved the complexity to $O(n^2/\lg n)$ [7]. However, unranking may not be the most efficient method for such a case, since unbordered words constitute a large fraction of all words [1, Table II]. While ranking and unranking are important problems in the study of combinatorial objects, for random generation we propose an alternative approach that produces at least one unbordered word in expected $O(n)$ time.

Our core idea is to exploit the relationship between unbordered words and Lyndon words. A Lyndon word is the lexicographically least cyclic shift (or, conjugate) of a primitive word [8]. It is well known that every unbordered word has a Lyndon conjugate [9], and every Lyndon word has

[1] Although this is not explicitly stated in Nielsen's paper, the algorithm outputs each unbordered word explicitly, which necessarily takes $O(n)$ time per word.

at least one unbordered conjugate (the word itself)[2]. For example, `abbaad` is an unbordered word, and its Lyndon conjugate is `aadabb`, which also admits other unbordered conjugates such as `bbaada` and `dabbaa`. If we represent each unbordered word by its Lyndon conjugate and the corresponding cyclic shift, then given a Lyndon word of length n with k unbordered conjugates, the space to store them decreases from $kn \lg \sigma$ bits in the explicit representation to $n \lg \sigma + \min\{n, k \lg n, (n-k) \lg n\}$ bits in the implicit representation. The actual space usage depends on how the information is represented: using an n-bit array to indicate which conjugates are unbordered/bordered, storing the k indices explicitly for unbordered conjugates, or storing the $(n-k)$ indices explicitly for bordered conjugates.[3]

Research on conjugates of words has also been actively pursued. Booth proposed a linear-time algorithm to find the lexicographically least conjugate of a word over an ordered alphabet [10], and Shiloach further reduced the number of comparisons [11]. Harju et al. introduced the *border correlation function*, a binary array indicating which conjugates of a word are bordered; however, they did not discuss how to compute the array in practice [12]. More recently, Clokie et al. proposed a practical method to compute the border correlation function for automatic sequences [13]. In this paper, we propose an algorithm that computes the border correlation function of a given word in linear time.

Our contributions are summarized as follows:

1. We propose a linear-time algorithm to compute the border correlation function.
2. We present a method to enumerate unbordered words with amortized $O(1)$ time using the implicit representation.
3. We provide an algorithm to randomly generate at least one unbordered word in expected $O(n)$ time. In contrast, the unranking method of [7] requires worst-case $O(n^2/\lg n)$ time.

2 Preliminaries

Let Σ denote a finite ordered alphabet, and σ its size. An element of Σ is called a *symbol*. We define a *word* $S = S[0]S[1]\dots S[n-1]$ of length n ($|S| = n$), where all symbols are drawn from an alphabet of size σ. The set Σ^{+n} is the set of all words of length *at most* n. For any $0 \le i \le j < n$, we denote by $S[i,j] = S[i]S[i+1]\dots S[j]$ the factor of S from position i to j. If $i > j$, we define $S[i,j]$ to be the empty word. For $0 \le i < n$, let $\mathrm{rot}_i(S)$ denote the ith cyclic shift (or *conjugate*) of S, i.e., $\mathrm{rot}_i(S) = S[i, n-1]S[0, i-1]$.

When S is expressed as the concatenation of words U, V, W (each word possibly empty), i.e., $S = U \cdot V \cdot W$, then we call U and W a *prefix* and a *suffix* of S, respectively. A *proper prefix* (resp., *proper suffix*) of S is a prefix (resp., suffix) of S that is not equal to S itself. For any integer $k \ge 0$, let S^k denote the word obtained by repeating S itself k times. In particular, S^0 is the empty word, and S^2 is called a *square*. S is called *primitive* if and only if there is no word X and integer $k \ge 2$ with $S = X^k$.

2.1 Lyndon word

A *Lyndon word* is the lexicographically least conjugate of a primitive word [8]. The number of Lyndon words in Σ^{+n} is $\Theta(\sigma^n/n)$ [14]. One method to enumerate Lyndon words of length at most n is Duval's algorithm [15], which starts with the lexicographically smallest Lyndon word and stops at the lexicographically largest one, which is $\zeta := \max \Sigma$, i.e., the maximal symbol of the alphabet. For each symbol $b \ne \zeta$, let $\mathsf{succ}(b)$ denote the successor of b in Σ.

[2] Over an alphabet of size at least two, every primitive word of length greater than one has at least two unbordered conjugates: the lexicographically least and the colexicographically least conjugates.

[3] Alternative schemes, like Elias-Fano (using $2k + k \lg(n/k)$ bits) or compressed bit vector (using roughly $\lg \binom{n}{k}$ bits), may also be used to store the k indices. In our experiments, we always use the n-bit array representation for simplicity.

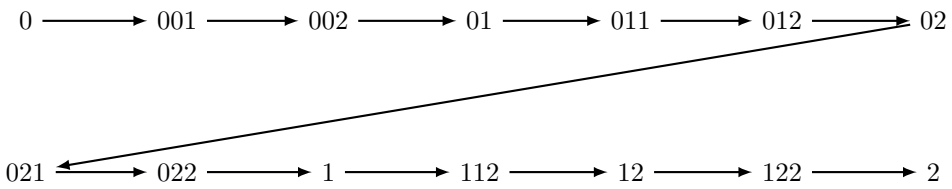

Figure 1: Duval's enumeration of Lyndon words over the alphabet $\{0, 1, 2\}$ with $n = 3$.

Given a number n (the maximum word length) and a Lyndon word $L \neq \zeta$, Duval's algorithm generates the next Lyndon word as follows:

1. Construct the word $W = L^h \cdot L' = P \cdot b \cdot \zeta^k$, where $h|L| + |L'| = n$, L' is a proper prefix of L, P is a proper prefix of W, $b \in \Sigma \setminus \{\zeta\}$ is a symbol, and $h, k \geq 0$ are non-negative integers.
2. Output the word $P \cdot \mathsf{succ}(b)$.

Duval proved that $P \cdot \mathsf{succ}(b)$ is the lexicographically next Lyndon word [15, 2.19 Théorème]. An example is shown in Figure 1. The worst-case time per word is $O(n)$, while the amortized cost to generate the next Lyndon word is constant [16].

2.2 Bordered and unbordered word

A word S is called *bordered* if it has a nonempty proper prefix that is a suffix of S; otherwise, it is called *unbordered*. Formally, a word S is called bordered if and only if it can be represented as $S = L \cdot B = B \cdot R$, where L, B and R are nonempty words. In such a case, B is called a *border* of S. The number of unbordered words in Σ^{+n} is $\Theta(\sigma^n)$ [1].

Every Lyndon word is unbordered. Conversely, since every unbordered word is primitive, it necessarily has a Lyndon conjugate [9]. Here we define two representations for storing a Lyndon word together with all its unbordered conjugates.

Definition 1 (Explicit and Implicit Representations)**.** *Let L be a Lyndon word of length n over an alphabet of size σ, and suppose L has k unbordered conjugates.*

- *The* explicit representation *refers to storing all k unbordered conjugates, which requires $kn \lg \sigma$ bits.*
- *The* implicit representation *refers to storing the Lyndon word L itself together with its border correlation function [12], which is an n-bit array to indicate which conjugates are unbordered/bordered. The total space usage is $n \lg \sigma + n$ bits.*

Example 1. *Consider the Lyndon word $L[0, 5] =$* `aadabb`*:*

- *Explicit representation:* $\{\mathrm{rot}_0(L) =$ `aadabb`*,* $\mathrm{rot}_2(L) =$ `dabbaa`*,* $\mathrm{rot}_3(L) =$ `abbaad`*,* $\mathrm{rot}_4(L) =$ `bbaada`$\}$*.*
- *Implicit representation: $L =$* `aadabb` *together with the border correlation function* `101110`*, where a* `0` *at the ith position indicates that the ith cyclic shift is bordered.*

2.3 Run structure

A *run* in a word S is represented by a tuple (s, t, p), where $S[s, t]$ has minimal period p and satisfies $2p \leq t - s + 1$ [17]. That is, $S[i] = S[i + p]$ for all $s \leq i \leq t - p$. Moreover, the interval $S[s, t]$ can be extended neither to the left nor to the right, meaning that $S[s - 1] \neq S[s + p - 1]$ and $S[t + 1] \neq S[t - p + 1]$. For example, `aabaabac` has three runs: $(0, 1, 1)$, $(0, 6, 3)$, and $(3, 4, 1)$, as shown in Figure 2.

The maximum number of runs in a word S of length n is $O(n)$ [17] and can be identified in $O(n)$ time for ordered alphabets [18].

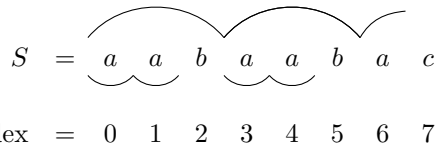

Figure 2: The run structure of `aabaabac`.

3 Main results

In this section, we first introduce an $O(n)$-time algorithm for computing the border correlation function that identifies which conjugates of a given word S of length n are unbordered. We then combine this algorithm with Duval's algorithm, which enumerates all Lyndon words, to enumerate all unbordered words of length at most n in amortized $O(1)$ time per word. Finally, we present another application of our algorithm, which randomly generates at least one unbordered word in $O(n)$ time.

3.1 Linear-time computation of the border correlation function

Given a word S of length n, the border correlation function of S can be computed by applying the (failure function computation of the) Knuth–Morris–Pratt (KMP) algorithm [19] to each conjugate. The ith entry of the failure function, which is equivalent to ith entry of the border array [20], represents the length of the longest border of $S[0, i-1]$. Hence, a conjugate of S is unbordered if and only if the last entry of its failure function is zero. Since KMP must be applied to each conjugate and each run of KMP requires $O(n)$ time to generate the failure function, the overall time complexity of computing the border correlation function is $O(n^2)$.

To reduce the time complexity to $O(n)$, we rely on Lemma 1 and Theorem 2, which are adapted from Proposition 1 in [21]. The key intuition is that if S is bordered, then S^2 must contain a square crossing the boundary between the two copies of S. For example, one occurrence of the period appearing as the suffix of $S^2[0, n-1]$ and the other as the prefix of $S^2[n, 2n-1]$. By examining all occurrences of square factors in S^2, we can determine which $\mathrm{rot}_i(S)$ is bordered for $0 \le i < n$.

Lemma 1. *Given a run (s, t, p) of a word S, $S[i-p, i+p-1]$ is a square for $s+p \le i \le t-p+1$.*

Proof. Since (s, t, p) is a run of S, we have $S[j] = S[j+p]$ for all $s \le j \le t-p$. Consider any i with $s+p \le i \le t-p+1$. The interval $[i-p,\ i+p-1]$ has length $2p$, which is a multiple of p. By the periodicity, the first p symbols $S[i-p,\ i-1]$ are identical to the next p symbols $S[i,\ i+p-1]$. Hence $S[i-p,\ i+p-1]$ is a square. □

Since the run structure of S can be identified in linear time [18] and the number of runs is $O(n)$ [17], Lemma 1 can identify the locations of all squares in $O(n)$ time.[4]

Theorem 2. *Let S be a word of length n and $W = S^2$. If there exists a run (s, t, p) of W such that $s+p \le i \le t-p+1$ and $p \le n/2$, then $\mathrm{rot}_{(i \bmod n)}(S)$ is bordered.*

Proof. By Lemma 1, the condition implies that $W[i-p,\ i+p-1]$ is a square. In particular, $W[i-p,\ i-1]$ is a suffix of $\mathrm{rot}_{(i \bmod n)}(S)$ and $W[i,\ i+p-1]$ is a prefix of $\mathrm{rot}_{(i \bmod n)}(S)$. Hence $\mathrm{rot}_{(i \bmod n)}(S)$ is bordered. Moreover, if $\mathrm{rot}_{(i \bmod n)}(S)$ is bordered, then it has a border of length no greater than $n/2$ [22, Lemma 4]. Thus, it suffices to check only runs with period $p \le n/2$. □

Figure 3 shows an example of Theorem 2. Based on this observation, we now develop a linear-time method to compute the border correlation function of S, which we present in Algorithm 1. Step 1 of Algorithm 1 can be performed in $O(n)$ time [18], and Step 2 can also be accomplished

[4]There can be $O(n^2)$ many squares. However, for our purpose it suffices to have the linear-sized representation of all runs.

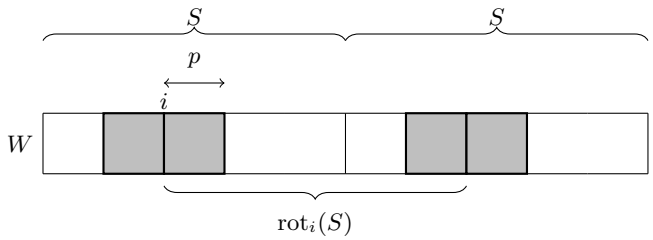

Figure 3: Example illustrating Theorem 2. The gray squares indicate the border of $\mathrm{rot}_i(S)$.

in $O(n)$ time using techniques such as the sweep-line algorithm. Therefore, Algorithm 1 computes the border correlation function in overall $O(n)$ time.

Algorithm 1: Compute the border correlation function in $O(n)$ time

Given a string $S \in \Sigma^{+n}$ of length $m \leq n$:
1. Compute the run structure of $W = S^2$.
2. For each run (s, t, p) with $p \leq m/2$, mark $\mathrm{rot}_{(i \bmod m)}(S)$ as bordered for all $s + p \leq i \leq t - p + 1$.

3.2 Amortized $O(1)$-time enumeration of unbordered words

As mentioned in the preliminaries, every unbordered word has a Lyndon conjugate [9]. Therefore, unbordered words can be enumerated by combining Duval's algorithm [15] with Algorithm 1, as described in Algorithm 2.

Algorithm 2: Amortized $O(1)$ time enumeration of unbordered words

1. For each Lyndon word L enumerated by Duval's algorithm:
2. Apply Algorithm 1 to L (invariant: $L \in \Sigma^{+n}$).

The unbordered words are stored in the implicit representation, ensuring amortized $O(1)$ time and worst-case $O(n)$ time per word. Otherwise, if each unbordered word were output explicitly, the amortized time would increase to $O(n)$ per word.

Theorem 3. *We can enumerate all unbordered words of length at most n of an ordered finite alphabet in amortized $O(1)$ time per reported word by using the implicit representation.*

Proof. The number of Lyndon words in Σ^{+n} is $\Theta(\sigma^n/n)$ [14] and it takes amortized constant time to generate the next Lyndon word using Duval's algorithm [16]. For each Lyndon word, Algorithm 1 requires $O(n)$ time to identify all of its unbordered conjugates. Since the total number of unbordered words is $\Theta(\sigma^n)$ [1], the amortized cost per unbordered word is constant. □

3.3 Random generation of at least one unbordered word in $O(n)$ expected time

As shown in Algorithm 3, we begin by randomly generating a word S of length n. If S is not primitive, we regenerate S until it becomes primitive. Since checking whether S is primitive can be done in $O(n)$ time and it takes expected constant time to obtain a primitive word[5], and since

[5]For words of length 2, the probability of being primitive is $1 - \frac{1}{\sigma}$. For general n, this probability approaches 1 as $n \to \infty$ [14].

every primitive word has at least one unbordered conjugate, such an S is guaranteed to yield an unbordered conjugate. Therefore, the expected time to generate at least one unbordered word is $O(n)$.

Algorithm 3: Generate a random unbordered word in expected $O(n)$ time

Given a random string S of length n:

1. While S is not primitive, regenerate S.
2. Compute the run structure of $W = S^2$.
3. For each run (s, t, p) with $p \leq n/2$, mark $\mathrm{rot}_{(i \bmod n)}(S)$ as bordered for all $s + p \leq i \leq t - p + 1$.

4 Experiments

We evaluated all our experiments on a machine with AMD Ryzen 5 3600 clocked at 3.6GHz running Ubuntu 18.04.5 LTS. The used compiler was `g++ 7.5.0` with compile options `-std=c++17 -O3`. The query time (wall-clock time) was measured using the C++ `<chrono>` library. Our implementation can be found at the following link: `https://github.com/CheWeiTsao/unbordered_enumeration/`.

Table 1: Comparison of serialized space (KiB) for unbordered word enumeration, between explicit and implicit representations. Rows marked with an asterisk (*) use 400,000 randomly generated primitive words of length n to sample the set of Lyndon words.

σ	n	implicit (KiB)	explicit (KiB)	compression ratio	Lyndon words count	unbordered words count	unbordered / Lyndon
3	6	2	4	39.75%	196	633	3.23
10	6	1,639	6,654	24.63%	189,343	989,110	5.22
17	6	37,721	162,932	23.15%	4,328,625	24,049,441	5.56
26	6	471,867	2,096,440	22.51%	53,979,471	308,441,926	5.71
3	8	13	46	28.98%	1,318	5,553	4.21
10	8	149,853	858,429	17.46%	14,116,663	98,891,200	7.01
*17	8	4,297	26,376	16.29%	400,000	3,000,989	7.50
*26	8	4,297	27,004	15.91%	400,000	3,072,493	7.68
3	10	121	508	23.92%	9,382	49,545	5.28
*10	10	5,469	38,252	14.30%	400,000	3,560,940	8.90
*17	10	5,469	40,291	13.57%	400,000	3,750,716	9.38
*26	10	5,469	41,259	13.25%	400,000	3,840,802	9.60
* 3	50	23,047	554,853	4.15%	400,000	11,140,572	27.85
*10	50	23,047	886,562	2.60%	400,000	17,800,775	44.50
*17	50	23,047	934,074	2.47%	400,000	18,754,731	46.89
*26	50	23,047	956,288	2.41%	400,000	19,200,759	48.00
* 3	100	44,922	2,197,728	2.04%	400,000	22,281,920	55.70
*10	100	44,922	3,511,595	1.28%	400,000	35,602,705	89.01
*17	100	44,922	3,699,772	1.21%	400,000	37,510,561	93.78
*26	100	44,922	3,787,836	1.19%	400,000	38,403,403	96.01

The space usage is measured in terms of serialized space, and the compression ratio is defined as the ratio of the implicit space usage to the explicit one. We enumerated all unbordered words of length at most n, with each symbol drawn from an alphabet of size σ, and stored them using both

representations. To simplify the implementation, each unbordered word (explicit representation) or Lyndon word (implicit representation) is stored in a standard byte-oriented format, where each symbol occupies 8 bits, and the border correlation function is stored using n bits. For larger values of n and σ, storing the enumeration explicitly may require excessive space. Therefore, we randomly generate 400,000 primitive words of length n to sample the Lyndon words. Rows marked with an asterisk ($*$) indicate such cases.

The difference in space usage between the explicit and implicit representations is shown in Table 1. The compression ratio decreases as n and σ increase, with n having a stronger influence. The trend with respect to σ results from the increasing number of unbordered conjugates per Lyndon word. The trend with respect to n arises because the number of Lyndon words ($\Theta(\sigma^n/n)$) and the number of unbordered words ($\Theta(\sigma^n)$) of Σ^{+n} differ by a factor of $1/n$. As a result, we can observe that the compression ratio approaches $1/n$ as both n and σ increase.

Figure 4(a) shows the running time comparison for computing the border correlation function between the KMP algorithm and Algorithm 1. Each data point represents the average over 100 randomly generated words, and the observed trend in running time is consistent with the theoretical time complexity.

Figure 4(b) presents the running time comparison for unbordered word enumeration using Algorithm 2 and Nielsen's algorithm [1]. Within our experimental configuration, Nielsen's algorithm outperforms Algorithm 2. Due to hardware limitations, we could not determine the value of n at which Algorithm 2 becomes faster. Nevertheless, it should be noted that Nielsen's algorithm generates unbordered words *explicitly*, resulting in high space usage, whereas Algorithm 2 generates them *implicitly*, making it more suitable for larger n.

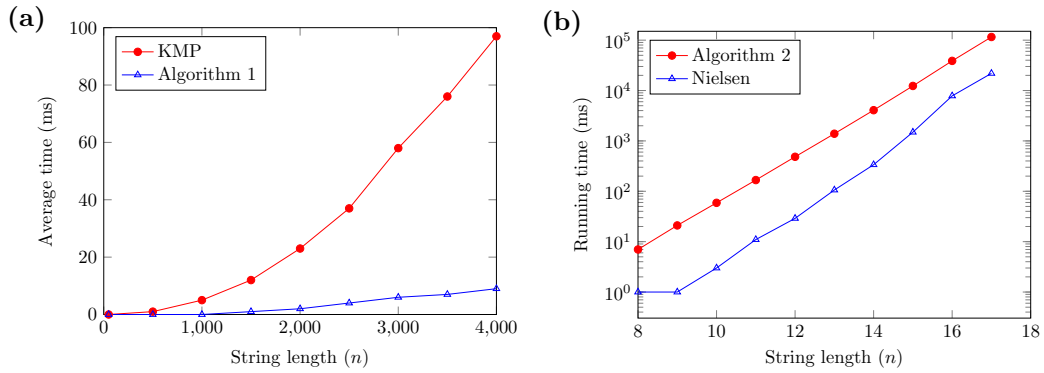

Figure 4: (a) Running time comparison for computing the border correlation function between KMP and Algorithm 1 on input strings of different lengths. Each data point represents the average over 100 randomly generated strings of the corresponding length. (b) Running time comparison for unbordered word enumeration using Algorithm 2 and Nielsen's algorithm.

5 Conclusion

We have proposed a linear-time algorithm to compute the border correlation function, a binary array that indicates which conjugates of a word are unbordered. The algorithm can enumerate unbordered words in a compressed representation with amortized $O(1)$ time per reported word. The experimental results show that the compression ratio is approximately $1/n$ for large values of n and σ (when $n \geq 8$ and $\sigma \geq 10$). In addition, the algorithm can also be used to randomly generate at least one unbordered word in expected $O(n)$ time, which is faster than the $O(n^2/\lg n)$ method in the literature [7]. We outline several directions for future work:

1. Duval's algorithm generates Lyndon words in amortized constant time. This suggests that further space reductions might be possible by recording the differences between consecutive

Lyndon words. A possible alternative approach is to use a Gray code to generate Lyndon words in amortized $O(1)$ time per word [23], while maintaining a fixed distance between consecutive Lyndon words.

2. Duval's algorithm currently has an $O(n^2)$-time worst-case delay[6] in generating two consecutive Lyndon words of length n. We believe that this can be reduced to $O(n)$ by appropriately adapting Duval's algorithm. With such an improvement, it would be possible to enumerate all unbordered words of length n in amortized $O(1)$ time and worst-case $O(n)$-time delay.

3. A faster algorithm for generating the run structure would be desirable, so that Algorithm 2 can start to outperform Nielsen's algorithm at smaller values of n.

Acknowledgments This research was supported by JSPS KAKENHI (grant numbers: `JP25K21150` and `JP23H04378`) and NSTC (grant number: `113-2221-E-007-139`).

6 References

References

[1] P. Nielsen, "A Note on Bifix-free Sequences," *IEEE Transactions on Information Theory*, vol. 19, no. 5, pp. 704–706, 1973.

[2] J. P. Duval, T. Harju, and D. Nowotka, "Unbordered Factors and Lyndon Words," *Discrete Mathematics*, vol. 308, no. 11, pp. 2261–2264, 2008.

[3] P. H. Cording, T. Gagie, M. B. T. Knudsen, and T. Kociumaka, "Maximal Unbordered Factors of Random Strings," *Theoretical Computer Science*, vol. 852, pp. 78–83, 2021.

[4] M. Anselmo, M. Flores, and M. Madonia, "Density of k-ary Words with 0, 1, 2-error Overlaps," *Theoretical Computer Science*, vol. 1025, p. 114958, 2025.

[5] S. Sekizaki and T. Mieno, "Longest Unbordered Factors on Run-Length Encoded Strings," in *Proc. SPIRE*, 2025, pp. 233–247.

[6] D. Gabric, "Ranking and Unranking Bordered and Unbordered Words," *Information Processing Letters*, vol. 184, p. 106452, 2024.

[7] J. Radoszewski, W. Rytter, and T. Waleń, "Faster Algorithms for Ranking/Unranking Bordered and Unbordered Words," in *Proc. SPIRE*, 2025, pp. 257–271.

[8] K. T. Chen, R. H. Fox, and R. C. Lyndon, "Free Differential Calculus, IV. The Quotient Groups of the Lower Central Series," *Ann. Math.*, vol. 68, no. 1, pp. 81–95, 1958.

[9] J. P. Duval, "Factorizing Words over an Ordered Alphabet," *Journal of Algorithms*, vol. 4, no. 4, pp. 363–381, 1983.

[10] K. S. Booth, "Lexicographically Least Circular Substrings," *Information Processing Letters*, vol. 10, no. 4, pp. 240–242, 1980.

[11] Y. Shiloach, "Fast Canonization of Circular Strings," *Journal of Algorithms*, vol. 2, no. 2, pp. 107–121, 1981.

[12] T. Harju and D. Nowotka, "Border Correlation of Binary Words," *Journal of Combinatorial Theory, Series A*, vol. 108, no. 2, pp. 331–341, 2004.

[6] Delay is the computation time needed between two consecutive reported words.

[13] T. Clokie, D. Gabric, and J. Shallit, "Circularly Squarefree Words and Unbordered Conjugates: A New Approach," in *Proc. WORDS*, 2019, pp. 133–144.

[14] M. Lothaire, *Combinatorics on Words*. Cambridge University Press, 1997, vol. 17.

[15] J. P. Duval, "Génération d'une Section des Classes de Conjugaison et Arbre des Mots de Lyndon de Longueur Bornée," *Theoretical Computer Science*, vol. 60, no. 3, pp. 255–283, 1988.

[16] J. Berstel and M. Pocchiola, "Average Cost of Duval's Algorithm for Generating Lyndon Words," *Theoretical Computer Science*, vol. 132, no. 1, pp. 415–425, 1994.

[17] R. Kolpakov and G. Kucherov, "Finding Maximal Repetitions in a Word in Linear Time," in *Proc. FOCS*, 1999, pp. 596–604.

[18] J. Ellert and J. Fischer, "Linear Time Runs Over General Ordered Alphabets," in *Proc. ICALP*, 2021, pp. 63:1–63:16.

[19] D. E. Knuth, J. H. Morris, Jr., and V. R. Pratt, "Fast Pattern Matching in Strings," *SIAM Journal on Computing*, vol. 6, no. 2, pp. 323–350, 1977.

[20] D. Moore, W. F. Smyth, and D. Miller, "Counting Distinct Strings," *Algorithmica*, vol. 23, no. 1, pp. 1–13, 1999.

[21] M. Crochemore, C. S. Iliopoulos, J. Radoszewski, W. Rytter, J. Straszyński, T. Waleń, and W. Zuba, "Shortest Covers of All Cyclic Shifts of a String," *Theoretical Computer Science*, vol. 866, pp. 70–81, 2021.

[22] G. Chen, S. J. Puglisi, and W. F. Smyth, "Fast and Practical Algorithms for Computing All the Runs in a String," in *Proc. CPM*, 2007, pp. 307–315.

[23] J. Sawada and A. Williams, "A Gray Code for Fixed-density Necklaces and Lyndon Words in Constant Amortized Time," *Theoretical Computer Science*, vol. 502, pp. 46–54, 2013.