R-enum Revisited: Speedup and Extension for Context-Sensitive Repeats and Net Frequencies

Kotaro Kimura*[1] and Tomohiro I†[1]

[1]Kyushu Institute of Technology, 680-4 Kawazu, Iizuka, Fukuoka 820-8502, Japan

November 17, 2025

Abstract

A *repeat* is a substring that occurs at least twice in a string, and is called a *maximal repeat* if it cannot be extended outwards without reducing its frequency. Nishimoto and Tabei [CPM, 2021] proposed *r-enum*, an algorithm to enumerate various characteristic substrings, including maximal repeats, in a string T of length n in $O(r)$ words of compressed working space, where $r \leq n$ is the number of runs in the Burrows-Wheeler transform (BWT) of T. Given the run-length encoded BWT (RLBWT) of T, r-enum runs in $O(n \log \log_w(n/r))$ time in addition to the time linear to the number of output strings, where $w = \Theta(\log n)$ is the word size. In this paper, we improve the $O(n \log \log_w(n/r))$ term to $O(n)$. We also extend r-enum to compute other context-sensitive repeats such as *near-supermaximal repeats (NSMRs)* and *supermaximal repeats*, as well as the *context-diversity* for every maximal repeat in the same complexities. Furthermore, we study the occurrences that witness NSMRs, which have recently attracted attention under the name of net occurrences: An occurrence of a repeat is called a *net occurrence* if it is not covered by another repeat, and the *net frequency* of a repeat is the number of its net occurrences. With this terminology, an NSMR is defined to be a repeat with a positive net frequency. Given the RLBWT of T, we show how to compute the set $\mathcal{S}^{\mathsf{nsmr}}$ of all NSMRs in T together with their net frequency/occurrences in $O(n)$ time and $O(r)$ space. We also show that an $O(r)$-space data structure can be built from the RLBWT to support queries of computing the net frequency of any query pattern P in optimal $O(|P|)$ time. The data structure is built in $O(r)$ space and in $O(n)$ time with high probability or deterministic $O(n + |\mathcal{S}^{\mathsf{nsmr}}| \log \log \min(\sigma, |\mathcal{S}^{\mathsf{nsmr}}|))$ time, where $\sigma \leq r$ is the alphabet size of T. To achieve this, we prove that the total number of net occurrences is less than $2r$. With the duality between net occurrences and *minimal unique substrings (MUSs)*, we get a new upper bound $2r$ of the number of MUSs in T, which may be of independent interest.

1 Introduction

Finding characteristic substring patterns in a given string T is a fundamental task for string processing. One of the simplest patterns based on frequencies would be a *repeat*, which occurs at least twice in T. Since the number of distinct repeats in T of length n can be $\Theta(n^2)$, maximality is often utilized to reduce the number down to $O(n)$ [13]. A

*kimura.kotaro365@mail.kyutech.jp
†tomohiro@ai.kyutech.ac.jp

substring of T is called *right-maximal* (resp. *left-maximal*) if it cannot be extended to the right (resp. to the left) without reducing its number of occurrences in T. If a repeat is both right-maximal and left-maximal, it is called a *maximal repeat (MR)*.

Gusfield [13] also introduced stricter criteria to reduce the redundancy in maximal repeats. A repeat is called a *supermaximal repeat (SMR)* if it is not a substring of another repeat, and a *near-supermaximal repeat (NSMR)* if at least one of its occurrences is not covered by another repeat. It is easy to see that an SMR is an NSMR, and an NSMR is an MR. NSMRs were studied under the name of *Chinese frequent strings* in [17, 18, 28], and *largest maximal repeats* in [23, 7]. Gallé and Tealdi [8, 9] proposed the concept of *context diversities* to define a general class of context-sensitive repeats including MRs and SMRs.

Enumerating these context-sensitive repeats has numerous applications in bioinformatics [13, 23, 1] and natural language processing [17, 18, 29, 20, 7] especially valuable for languages with no clear word segmentation such as Chinese and Japanese. Motivated by these applications, efficient enumeration algorithms have been extensively studied [29, 1, 27, 2, 26] using a family of suffix data structures such as suffix trees [31], directed acyclic word graphs (DAWGs) [5], compact DAWGs (CDAWGs) [4, 16], suffix arrays [19] and Burrows-Wheeler transform (BWT) [6].

In this paper, we focus on *r-enum* [26], an algorithm to enumerate the MRs, *minimal unique substrings (MUSs)* [14] and *minimal absent words (MAWs)* in $O(r)$ words of compressed working space, where $r \leq n$ is the number of runs in the BWT of T of length n. Given the *run-length encoded BWT (RLBWT)* of T, r-enum runs in $O(n \log \log_w(n/r))$ time in addition to the time linear to the number of output strings, where $w = \Theta(\log n)$ is the word size. We show that $O(n \log \log_w(n/r))$ time can be improved to $O(n)$ by resorting to the *move data structure* recently proposed by Nishimoto and Tabei [24] for constant-time LF-mapping on RLBWTs. We also extend r-enum to compute the NSMRs and SMRs, and the *context diversity* for every RMR in the same time and space complexities.

Furthermore, we consider problems for net occurrences and net frequencies, originated from the studies of Chinese frequent strings [17, 18] and have recently attracted attention from combinatorial and algorithmic points of view [10, 28, 12, 15, 11, 21]. An occurrence of a repeat is called a *net occurrence* if the occurrence is not covered by another repeat, and the *net frequency* of a repeat is the number of its net occurrences. We would like to point out the fact, which may have been overlooked for over two decades, that the strings with positive net frequencies are equivalent to the NSMRs [13] and the *largest maximal repeats* [23, 7], and thus appear to have broader applications. Also, net occurrences are conceptually equivalent to *maximum repeats* in [14].

Given the RLBWT of T, we show how to compute the set $\mathcal{S}^{\mathsf{nsmr}}$ of all NSMRs in T together with their net frequency/occurrences in $O(n)$ time and $O(r)$ words of space, solving the task called ALL-NF in [10] on RLBWTs. We also show that an $O(r)$-space data structure can be built from the RLBWT to support queries of computing the net frequency of any query pattern P in optimal $O(|P|)$ time. The data structure is built in $O(r)$ space and in $O(n)$ time with high probability or deterministic $O(n+|\mathcal{S}^{\mathsf{nsmr}}| \log \log \min(\sigma, |\mathcal{S}^{\mathsf{nsmr}}|))$ time, where $\sigma \leq r$ is the alphabet size of T. To achieve this, we prove that the total number of net occurrences is less than $2r$. With the duality between net occurrences and MUSs shown in [14, 21], we immediately get a new upper bound $2r$ of the number of MUSs in T, which may be of independent interest.

2 Preliminaries

2.1 Basic notation

We assume a standard word-RAM model with word size $w = \Theta(\log n)$, where n is the length of string T defined later. A space complexity is evaluated by the number of words unless otherwise noted. Any integer treated in this paper is represented in $O(1)$ space, i.e., $O(w)$ bits. A set of consecutive integers is called an *interval*, which can be represented in $O(1)$ space by storing its beginning and ending integers. For two integers b and e, let $[b..e]$ denote the interval $\{b, b+1, \ldots, e\}$ beginning at b and ending at e if $b \leq e$, and otherwise, the empty set. We also use $[b..e) := [b..e-1]$ to exclude the right-end integer.

A *string* (or an *array*) x over a set S is a sequence $x[1]x[2]\cdots x[|x|]$, where $|x|$ denotes the length of x and $x[i] \in S$ is the i-th element of x for any $i \in [1..|x|]$. The string of length 0 is called the *empty string* and denoted by ε. Let S^* denote the set of strings over S, and let $S^n \subset S^*$ be the set of strings of length $n \geq 0$. For integers $1 \leq b \leq e \leq |x|$, the *substring* of x beginning at b and ending at e is denoted by $x[b..e] = x[b]x[b+1]\cdots x[e]$. For convenience, let $x[b..e]$ with $b > e$ represent the empty string. We also use $x[b..e) = x[b..e-1]$ to exclude the right-end character. A *prefix* (resp. *suffix*) of x is a substring that matches $x[1..e]$ (resp. $x[b..|x|]$) for some $e \in [0..|x|]$ (resp. $b \in [1..|x|+1]$). We use the abbreviation $x[..e] = x[1..e]$ and $x[b..] = x[b..|x|]$. For a non-negative integer d, let x^d denote the string of length $|x|d$ such that $x^d[|x|(i-1)+1..i|x|] = x$ for any $i \in [1..d]$.

2.2 Tools

A function that injectively maps a set S of integers to $[1..O(|S|)]$ can be used as a *compact dictionary* for S to look up a value associated with an integer in S. We use the following known result:

Lemma 1 ([32, 30])**.** *Given $S \subseteq [1..u]$ of size $m \leq u \leq 2^{O(w)}$, we can build an $O(m)$-size dictionary in $O(m)$ time with high probability or deterministic $O(m \log\log m)$ time so that lookup queries can be supported in $O(1)$ worst-case time.*

A *predecessor query* for an integer i over the set S of integers asks to compute $\max\{j \in S \mid j \leq i\}$, which can be efficiently supported by the data structure of [3, Appendix A]:

Lemma 2 ([3])**.** *Given $S \subseteq [1..u]$ of size $m \leq u \leq 2^{O(w)}$, we can build an $O(m)$-size data structure in $O(m \log\log_w(u/m))$ time and $O(m)$ space to support predecessor queries in $O(\log\log_w(u/m))$ time for any $i \in [1..n]$.*

A *range distinct query* $\mathsf{RD}(x, p, q)$ for a string x and integers $1 \leq p \leq q \leq |x|$ asks to enumerate (c, p_c, q_c) such that p_c and respectively q_c are the smallest and largest positions in $[b..e]$ with $x[p_c] = x[q_c] = c$ for every distinct character c that occurs in $x[p..q]$. It is known that a range distinct query can be supported in output-optimal time after preprocessing x.

Lemma 3 ([22, 2])**.** *Given a string $x \in [1..\sigma]^m$, we can build an $O(m)$-size data structure in $O(m)$ time and space to support range distinct queries. With a null-initialized array of size σ, each query can be answered in $O(k)$ time, where k is the number of outputs.*

A minor remark is that the output of range distinct queries of Theorem 3 may not be sorted in any order of characters.

2.3 Context-sensitive repeats

Let $\Sigma_\$ = [1..\sigma]$ be an ordered alphabet with the smallest character \$, and let $\Sigma = \Sigma_\$ \setminus \{\$\}$. Throughout this paper, $T[1..n] \in \Sigma_\$^n$ is a string of length $n \geq 2$ that contains all characters in $\Sigma_\$$. We assume for convenience that \$ is used as a sentinel that exists at $T[n]$ and $T[0]$, and does not occur in $T[1..n-1]$. A position $b \in [1..n]$ is called an *occurrence* of a string $x \in \Sigma_\$^*$ in T if $T[b..b+|x|) = x$. Let $\mathsf{Occ}(x)$ denote the set of occurrences of x in T. The *left context* $\mathsf{lc}(x)$ (resp. *right context* $\mathsf{rc}(x)$) of a substring $x \in \Sigma^*$ of T is the set of characters immediately precedes (resp. follows) the occurrences of x, i.e., $\mathsf{lc}(x) = \{T[b-1] \mid b \in \mathsf{Occ}(x)\}$ (resp. $\mathsf{rc}(x) = \{T[b+|x|] \mid b \in \mathsf{Occ}(x)\}$). The *context diversity* of x is the pair $(|\mathsf{lc}(x)|, |\mathsf{rc}(x)|)$ of integers in $[1..\sigma]$. A substring x of T is called a *repeat* if $|\mathsf{Occ}(x)| \geq 2$, and called *unique* if $|\mathsf{Occ}(x)| = 1$. A repeat $x \in \Sigma^*$ in T is a *left-maximal repeat (LMR)* (resp. *right-maximal repeat (RMR)*) if $|\mathsf{lc}(x)| > 1$ (resp. $|\mathsf{rc}(x)| > 1$), and is a *maximal repeat (MR)* if x is both left- and right-maximal. A repeat x in T is a *supermaximal repeat (SMR)* if it is not a substring of another repeat. An occurrence b of a repeat x is called a *net occurrence* if there is no occurrence b' of another repeat x' such that $[b..b+|x|) \subset [b'..b'+|x'|)$. Let $\mathsf{NOcc}(x) \subseteq \mathsf{Occ}(x)$ be the set of net occurrences of x and let $\mathsf{NF}(x) := |\mathsf{NOcc}(x)|$ be the number of net occurrences called the *net frequency* of x. A *near-supermaximal repeat (NSMR)* is a repeat with a positive net frequency. Let $\mathcal{S}$, $\mathcal{S}^{\mathsf{r}}$, $\mathcal{S}^{\mathsf{lmr}}$, $\mathcal{S}^{\mathsf{rmr}}$, $\mathcal{S}^{\mathsf{mr}}$, $\mathcal{S}^{\mathsf{nsmr}}$ and $\mathcal{S}^{\mathsf{smr}}$ be the sets of all substrings, repeats, LMRs, RMRs, MRs, NSMRs and SMRs in $T[1..n]$, respectively.

To describe algorithms uniformly, we assume that the empty string is a repeat that occurs at every position b in $[1..n]$, i.e., $\mathsf{Occ}(\varepsilon) = [1..n]$. Since $\mathsf{lc}(\varepsilon) = \mathsf{rc}(\varepsilon) = \Sigma_\$$, ε is always in $\mathcal{S}^{\mathsf{lmr}}$, $\mathcal{S}^{\mathsf{rmr}}$ and $\mathcal{S}^{\mathsf{mr}}$. We set the following exception for the definition of net occurrences of ε: An occurrence $b \in \mathsf{Occ}(\varepsilon)$ is a net occurrence of ε if and only if $T[b-1]$ and $T[b]$ are both unique. This exception provides a smooth transition to another characterization of net occurrences based on the uniqueness of extended net occurrences [10, 21].

Table 2 summarizes the concepts introduced in this subsection for $T = \texttt{abcbbcbcabc\$}$.

2.4 Suffix trees and suffix arrays

The *suffix trie* of T is the tree defined on the set $\mathcal{S}$ of nodes and edges from $x \in \mathcal{S}$ to $xc \in \mathcal{S}$ with $c \in \Sigma_\$$. The *suffix tree* of T is the compacted suffix trie in which every non-branching internal node of the suffix trie is removed and considered to be an *implicit node*. Every internal node of the suffix tree is a right-maximal repeat x and has $|\mathsf{rc}(x)|$ children. A link from a node x to a (possibly implicit) node $ax \in \mathcal{S}$ for some $a \in \Sigma_\$$ is called a *Weiner link*. It is known that every node can be reached by following Weiner links from the root ε, and the total number of Weiner links is at most $3n$, which can be seen from the duality between suffix trees and DAWGs [5].

The *suffix array* $\mathsf{SA}[1..n]$ of T is the integer array such that, for any $i \in [1..n]$, $T[\mathsf{SA}[i]..]$ is the lexicographically i-th suffix among the non-empty suffixes of T. For any string $x \in \mathcal{S}$, the *SA-interval* of x, denoted by $\mathcal{I}(x)$, is the maximal interval $[p..q]$ such that x is a prefix of $T[\mathsf{SA}[i]..]$ for any $i \in [p..q]$. Then, it holds that $\mathsf{Occ}(x) = \{\mathsf{SA}[i] \mid i \in \mathcal{I}(x)\}$.

2.5 Burrows-Wheeler transform

The *Burrows-Wheeler transform (BWT)* $\mathsf{L}[1..n]$ of T is the string such that $\mathsf{L}[i] = T[\mathsf{SA}[i]-1]$ for any $i \in [1..n]$. By definition, it holds that $\mathsf{lc}(x) = \{\mathsf{L}[i] \mid i \in \mathcal{I}(x)\}$ for any string x. The LF-mapping LF is the permutation on $[1..n]$ such that $\mathsf{LF}(i)$ represents the lexicographic rank of the suffix $T[\mathsf{SA}[i]-1..]$ if $\mathsf{SA}[i] \neq 1$, and otherwise $\mathsf{LF}(i) = 1$. The

$x \in \mathcal{S}^{\mathsf{r}}$	occurrences	$\mathsf{Occ}(x)$	$\mathsf{lc}(x)$	$\mathsf{rc}(x)$	context diversity
ε	a b c b b c b c a b c $	$[1..12]$	$\{\$, \mathtt{a}, \mathtt{b}, \mathtt{c}\}$	$\{\$, \mathtt{a}, \mathtt{b}, \mathtt{c}\}$	$(4, 4)$
a	a b c b b c b c a b c $	$\{1, 9\}$	$\{\$, \mathtt{c}\}$	$\{\mathtt{b}\}$	$(2, 1)$
b	a b c b b c b c a b c $	$\{2, 4, 5, 7, 10\}$	$\{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$	$\{\mathtt{b}, \mathtt{c}\}$	$(3, 2)$
c	a b c b b c b c a b c $	$\{3, 6, 8, 11\}$	$\{\mathtt{b}\}$	$\{\$, \mathtt{a}, \mathtt{b}\}$	$(1, 3)$
ab	a b c b b c b c a b c $	$\{1, 9\}$	$\{\$, \mathtt{c}\}$	$\{\mathtt{c}\}$	$(2, 1)$
bc	a b c b b c b c a b c $	$\{2, 5, 7, 10\}$	$\{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$	$\{\$, \mathtt{a}, \mathtt{b}\}$	$(3, 3)$
cb	a b c b b c b c a b c $	$\{3, 6\}$	$\{\mathtt{b}\}$	$\{\mathtt{b}, \mathtt{c}\}$	$(1, 2)$
abc	a b c b b c b c a b c $	$\{1, 9\}$	$\{\$, \mathtt{c}\}$	$\{\$, \mathtt{b}\}$	$(2, 2)$
bcb	a b c b b c b c a b c $	$\{2, 5\}$	$\{\mathtt{a}, \mathtt{b}\}$	$\{\mathtt{b}, \mathtt{c}\}$	$(2, 2)$

$x \in \mathcal{S}^{\mathsf{r}}$	net occurrences	$\mathsf{NOcc}(x)$	$\mathsf{NF}(x)$	LMR	RMR	MR	NSMR	SMR
ε	a b c b b c b c a b c $	$\emptyset$	0	$\surd$	$\surd$	$\surd$		
a	a b c b b c b c a b c $	$\emptyset$	0	$\surd$				
b	a b c b b c b c a b c $	$\emptyset$	0	$\surd$	$\surd$	$\surd$		
c	a b c b b c b c a b c $	$\emptyset$	0		$\surd$			
ab	a b c b b c b c a b c $	$\emptyset$	0	$\surd$				
bc	a b c b b c b c a b c $	$\{7\}$	1	$\surd$	$\surd$	$\surd$	$\surd$	
cb	a b c b b c b c a b c $	$\emptyset$	0		$\surd$			
abc	a b c b b c b c a b c $	$\{1, 9\}$	2	$\surd$	$\surd$	$\surd$	$\surd$	$\surd$
bcb	a b c b b c b c a b c $	$\{2, 5\}$	2	$\surd$	$\surd$	$\surd$	$\surd$	$\surd$

Table 1: An example of the concepts introduced in Section 2.3 for $T = \mathtt{abcbbcbcabc\$}$. For every repeat $x \in \mathcal{S}^{\mathsf{r}}$, we show a visualization of the occurrences of x, $\mathsf{Occ}(x)$, $\mathsf{lc}(x)$, $\mathsf{rc}(x)$, and the context diversity of x in the upper table. In the lower table, we show a visualization of the net occurrences of x, $\mathsf{NOcc}(x)$, $\mathsf{NF}(x)$, and subsequently check if x is in each of $\mathcal{S}^{\mathsf{lmr}}$, $\mathcal{S}^{\mathsf{rmr}}$, $\mathcal{S}^{\mathsf{mr}}$, $\mathcal{S}^{\mathsf{nsmr}}$ and $\mathcal{S}^{\mathsf{smr}}$.

FL-mapping FL is the inverse mapping of LF. Given an integer i in $\mathcal{I}(x)$ for a string x, we can compute x by using FL-mapping $|x|$ times from i because $x[k] = \mathsf{L}[\mathsf{FL}^k(i)]$ holds for any k $(1 \le k \le |x|)$, where $\mathsf{FL}^k(i)$ is defined recursively $\mathsf{FL}^k(i) = \mathsf{FL}(\mathsf{FL}^{k-1}(i))$ with the base case $\mathsf{FL}^1(i) = \mathsf{FL}(i)$.

Table 2 shows an example of SA, $\mathsf{LF}(\cdot)$, $\mathsf{FL}(\cdot)$ and L for $T = \mathtt{abcbbcbcabc\$}$. Figure 1 also illustrates the suffix tree of $T = \mathtt{abcbbcbcabc\$}$ and a relationship between Weiner links and BWTs.

The *run-length encoded BWT (RLBWT)* of T represents L in $O(r)$ space by string $\mathsf{L}'[1..r] \in \Sigma_{\$}^r$ and array $\mathsf{d}[1..r] \in [1..n]^r$ such that $\mathsf{L} = \mathsf{L}'[1]^{\mathsf{d}[1]}\mathsf{L}'[2]^{\mathsf{d}[2]} \dots \mathsf{L}'[r]^{\mathsf{d}[r]}$ and $\mathsf{L}'[i] \neq \mathsf{L}'[j]$ for any $1 \le i < j \le r$. Each component $\mathsf{L}'[i]^{\mathsf{d}[i]}$ is called a *run* of L. While $r \le n$ is obvious, inequality $\sigma \le r$ holds under the assumption that T contains all characters in $\Sigma_{\$}$.

Example 4. *For* $T = \mathtt{abcbbcbcabc\$}$*, its BWT* $\mathsf{L} = \mathtt{cc\$cacabbbbb}$ *has* $r = 7$ *runs. Then, we have* $\mathsf{L}' = \mathtt{c\$cacab}$ *and* $\mathsf{d} = [2, 1, 1, 1, 1, 1, 5]$.

A notable property of LF is that $\mathsf{LF}(i) = k_1 + k_2$ holds, where k_1 is the number of characters smaller than $\mathsf{L}[i]$ in L and k_2 is the number of occurrences of $\mathsf{L}[i]$ in $\mathsf{L}[1..i]$. In particular, if $\mathsf{L}[p..q]$ consists of a single character, then $\mathsf{LF}(i) = \mathsf{LF}(p) + i - p$ for any $i \in [p..q]$, and $\mathsf{FL}(i') = \mathsf{FL}(p') + i' - p'$ for any $i' \in [p'..q']$ with $p' = \mathsf{LF}(p)$ and $q' = \mathsf{LF}(q)$. Hence, LF-mapping can be implemented in $O(r)$ space by constructing the predecessor data structure for the set of beginning positions of runs of L and remembering the $\mathsf{LF}(\cdot)$ values for those positions (FL-mapping can be implemented similarly). Using the predecessor data structure of Theorem 2, we get:

i	$\mathsf{SA}[i]$	$\mathsf{LF}(i)$	$\mathsf{FL}(i)$	$\mathsf{L}[i]$	$T[\mathsf{SA}[i]..]$
1	12	9	3	c	\$
2	9	10	5	c	abc\$
3	1	1	7	\$	abcbbcbcabc\$
4	4	11	8	c	bbcbcabc\$
5	10	2	9	a	bc\$
6	7	12	10	c	bcabc\$
7	2	3	11	a	bcbbcbcabc\$
8	5	4	12	b	bcbcabc\$
9	11	5	1	b	c\$
10	8	6	2	b	cabc\$
11	3	7	4	b	cbbcbcabc\$
12	6	8	6	b	cbcabc\$

Table 2: An example of $\mathsf{SA}[i]$, $\mathsf{LF}(i)$, $\mathsf{FL}(i)$ and $\mathsf{L}[i]$ for $T = \texttt{abcbbcbcabc\$}$.

Lemma 5. *Given the RLBWT of T, we can construct an $O(r)$-size data structure in $O(r \log \log_w(n/r))$ time and $O(r)$ space to compute $\mathsf{LF}(i)$ and $\mathsf{FL}(i)$ in $O(\log \log_w(n/r))$ time for any $i \in [1..n]$.*

Nishimoto and Tabei [24] proposed a novel data structure, called the *move data structure*, to implement LF-mapping by a simple linear search on a list of intervals without predecessor queries. We consider a list $[p_1..p_2), [p_2..p_3), \ldots, [p_{\hat{r}}..p_{\hat{r}+1})$ of $\hat{r} = O(r)$ intervals such that $p_1 = 1$, $p_{\hat{r}+1} = n + 1$, and for any $k \in [1..\hat{r}]$, $\mathsf{L}[p_k..p_{k+1})$ consists of a single character and $[\mathsf{LF}(p_k)..\mathsf{LF}(p_{k+1} - 1)]$ intersects with $O(1)$ intervals in the list. We let each $[p_k..p_{k+1})$ have a pointer to the interval $[p_{k'}..p_{k'+1})$ that contains $\mathsf{LF}(p_k)$ and offset $o_k = \mathsf{LF}(p_k) - p_{k'}$. Then, $\mathsf{LF}(i) = \mathsf{LF}(p_k) + i - p_k = p_{k'} + o_k + i - p_k$ holds for any $i \in [p_k..p_{k+1})$. Given i and interval $[p_k..p_{k+1})$ that contains i, we can compute $\mathsf{LF}(i)$ and find the interval that contains $\mathsf{LF}(i)$ by assessing $O(1)$ intervals from $[p_{k'}..p_{k'+1})$ to the right in the list. Note that $\mathsf{LF}(i)$ computation on the move data structure works along with the pointer to the interval in the list that contains i, which is handled implicitly hereafter. Under the assumption that $\sigma \leq r$ in this paper, we can construct the data structure needed for LF-mapping (and FL-mapping) in $O(r \log r)$ time [25, Appendix D]:

Lemma 6. *Given the RLBWT of T, we can construct an $O(r)$-size data structure in $O(r \log r)$ time and $O(r)$ space to compute $\mathsf{LF}(i)$ and $\mathsf{FL}(i)$ in $O(1)$ time for any $i \in [1..n]$.*

3 Upper bound $2r$ of net occurrences and MUSs

In this section, we consider combinatorial properties of net occurrences on BWTs.

We start with the following lemma.

Lemma 7. *For any integer $i \in [p..q] = \mathcal{I}(x)$ for a repeat x of T, $\mathsf{SA}[i]$ is a net occurrence of x if and only if $\mathsf{L}[i]$ is unique in $\mathsf{L}[p..q]$ and $T[\mathsf{SA}[i]..\mathsf{SA}[i] + |x|]$ is unique in T.*

Proof. If $a = \mathsf{L}[i]$ is not unique in $\mathsf{L}[p..q]$, then $\mathsf{SA}[i]$ is not a net occurrence of x because $ax = T[\mathsf{SA}[i] - 1..\mathsf{SA}[i] + |x|)$ is a repeat that covers $[\mathsf{SA}[i]..\mathsf{SA}[i] + |x|)$. Similarly, if $T[\mathsf{SA}[i]..\mathsf{SA}[i] + |x|]$ is not unique in T, then $\mathsf{SA}[i]$ is not a net occurrence of x because $T[\mathsf{SA}[i]..\mathsf{SA}[i] + |x|]$ is a repeat that covers $[\mathsf{SA}[i]..\mathsf{SA}[i] + |x|)$.

On the other hand, if $\mathsf{L}[i]$ is unique in $\mathsf{L}[p..q]$ and $T[\mathsf{SA}[i]..\mathsf{SA}[i] + |x|]$ is unique in T, there is no repeat that covers $[\mathsf{SA}[i]..\mathsf{SA}[i] + |x|)$, and hence, $\mathsf{SA}[i]$ is a net occurrence of x. □

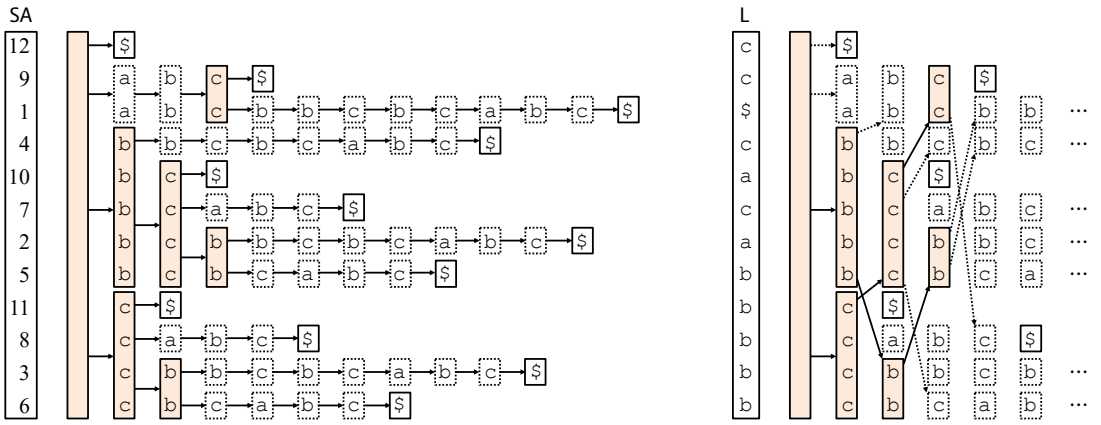

Figure 1: The left figure shows the suffix tree of $T = \texttt{abcbbcbcabc\$}$ illustrated over sorted suffixes, on which each node x can be represented by $\mathcal{I}(x)$ and $|x|$. A solid box is a node (highlighted for internal nodes), and a dotted box is an implicit node. The right figure shows the Weiner links outgoing from all internal nodes (the Weiner links from leaves are omitted). The Weiner links that points to internal nodes are depicted with solid arrows, and the other ones with dotted arrows. Observe that there is a Weiner link from a node x to ax for any character $a \in \mathsf{lc}(x) = \{\mathsf{L}[i] \mid i \in \mathcal{I}(x)\}$, where the case with $a = \$$ is excluded unless $x = \varepsilon$.

Example 8. *Let us consider a repeat* $\texttt{bc}$ *with* $\mathcal{I}(\texttt{bc}) = [5..8]$ *in the example of Table 2.* $\mathsf{SA}[6] = 7$ *is a net occurrence of* $\texttt{bc}$ *because* $\mathsf{L}[6] = \texttt{c}$ *is unique in* $\mathsf{L}[5..8] = \texttt{acab}$*, and* $T[\mathsf{SA}[6]..\mathsf{SA}[6]+2] = T[7..9] = \texttt{bca}$ *is unique in* T*.* $\mathsf{SA}[5] = 10$ *and* $\mathsf{SA}[7] = 2$ *are not net occurrences of* $\texttt{bc}$ *because* $\mathsf{L}[5] = \mathsf{L}[7] = \texttt{a}$ *is not a unique character in* $\mathsf{L}[5..8]$*.* $\mathsf{SA}[7] = 2$ *and* $\mathsf{SA}[8] = 5$ *are not net occurrences of* $\texttt{bc}$ *because* $T[2..4] = T[5..7] = \texttt{bcb}$ *is a repeat.*

The following lemma was implicitly proved in [10, Theorem 24] to show the total length of all net occurrences is bounded by twice the sum of irreducible LCPs.

Lemma 9. *If* $\mathsf{SA}[i]$ *is a net occurrence of a repeat* x *of* T*, then* i *is at the beginning or ending position of a run in* L*, and* $\mathsf{SA}[i]$ *cannot be a net occurrence of another repeat.*

Proof. Let $[p..q] = \mathcal{I}(x)$. If $i \in [p..q]$ is not at the run's boundaries of L, then at least one of $\mathsf{L}[i] = \mathsf{L}[i+1]$ with $i+1 \in [p..q]$ or $\mathsf{L}[i-1] = \mathsf{L}[i]$ with $i-1 \in [p..q]$ holds, which means that $\mathsf{SA}[i]$ is not a net occurrence of x by Theorem 7. $\mathsf{SA}[i]$ cannot be a net occurrence of two distinct repeats, since otherwise, the occurrence of the shorter repeat is covered by the longer one. □

Theorem 9 leads to the following theorem:

Theorem 10. *The total number of net occurrences of* T *is less than* $2r$*.*

Proof. Let r_1 and respectively $r_{>1}$ be the number of runs in L of length 1 and longer than 1, i.e., $r = r_1 + r_{>1}$. Since there are $r_1 + 2r_{>1}$ positions that correspond to beginning or ending positions of runs in L, it is clear from Theorem 9 that the total number of net occurrences is at most $r_1 + 2r_{>1} \leq 2r$. The number is shown to be less than $2r$ by looking closer at some special positions like 1 and n on L: Since $\mathsf{SA}[1]$ (resp. $\mathsf{SA}[n]$) cannot be a net occurrence if the first (resp. last) run of L is longer than 1, only one net occurrence is charged to the first (resp. last) run of L. □

A non-empty interval $[b..e]$ is called a *minimal unique substring (MUS)* if $T[b..e]$ is unique while $T[b+1..e]$ and $T[b..e-1]$ are repeats. In [14, 21], it has been proved that the net occurrences and MUSs are dual concepts, basically saying that for any two consecutive MUSs $[b..e]$ and $[b'..e']$ with $b < b'$, $b+1$ is a net occurrence of $T[b+1..e'-1]$. With this duality, we immediately get the following:

Corollary 11. *The number of MUSs of T is less than $2r$.*

4 R-enum revisited

4.1 Rough sketch of r-enum and speedup

Conceptually, r-enum performs a breadth-first traversal on the internal nodes $\mathcal{S}^{\mathsf{rmr}}$ of the suffix tree by following Weiner links using RLBWT-based data structures. Each node $x \in \mathcal{S}^{\mathsf{rmr}}$ is visited with its suffix interval $\mathcal{I}(x)$ and length $|x|$ together with the information about its right extensions $\mathsf{rlist}(x)$, which is the list of the set $\{(c, \mathcal{I}(xc)) \mid c \in \mathsf{rc}(x)\}$ sorted by the character c. The triplet $\mathsf{repr}(x) = (\mathcal{I}(x), \mathsf{rlist}(x), |x|)$ is called the *rich representation* [2, 26] of x, which can be stored in $O(|\mathsf{rc}(x)|)$ space.

Given $\mathsf{repr}(x) = ([p..q], \{(c_i, [p_i..q_i])\}_{i=1}^{k}, |x|)$, we can compute $\{\mathsf{repr}(ax) \mid ax \in \mathcal{S}, a \in \Sigma\}$ as follows: A null-initialized array $A[1..\sigma]$ is used as a working space to build $\mathsf{rlist}(ax)$ at $A[a]$ for $a \in \mathsf{lc}(x) \setminus \{\$\}$. For every $i \in [1..k]$, we compute $\mathsf{lc}(xc_i)$ by a range distinct query on $\mathsf{L}[p_i..q_i]$, and for each $a \in \mathsf{lc}(xc_i) \setminus \{\$\}$ we compute $\mathcal{I}(axc_i)$ by LF-mapping and append $(c_i, \mathcal{I}(axc_i))$ to the list being built at $A[a]$. After processing all $i \in [1..k]$, $A[a]$ becomes $\mathsf{rlist}(ax)$. Finally, we enumerate $\mathsf{lc}(x)$ by a range distinct query on $\mathsf{L}[p..q]$ to go through only the used entries of A to compute $\mathsf{repr}(ax)$ and clear the entries. For the next round of the breadth-first traversal on $\mathcal{S}^{\mathsf{rmr}}$, we keep only the set $\{\mathsf{repr}(ax) \mid ax \in \mathcal{S}^{\mathsf{rmr}}\}$, discarding $\mathsf{repr}(ax)$ if $\mathsf{rlist}(ax)$ contains only one element.

Example 12. *In the r-enum for $T = \mathtt{abcbbcbcabc\$}$, we visit a repeat $\mathtt{bc} \in \mathcal{S}^{\mathsf{rmr}}$ with its rich representation $\mathsf{repr}(\mathtt{bc}) = (\mathcal{I}(\mathtt{bc}), \mathsf{rlist}(\mathtt{bc}), |\mathtt{bc}|) = ([5..8], \{(\$, [5..5]), (\mathtt{a}, [6..6]), (\mathtt{b}, [7..8])\}, 2)$. With a null-initialized array $A[1..\sigma]$, we compute $\mathsf{repr}(\mathtt{abc}) = ([2..3], \{(\$, [2..2]), (\mathtt{b}, [3..3])\}, 3)$ at $A[\mathtt{a}]$, $\mathsf{repr}(\mathtt{bbc}) = ([4..4], \{(\mathtt{b}, [4..4])\}, 3)$ at $A[\mathtt{b}]$, and $\mathsf{repr}(\mathtt{cbc}) = ([12..12], \{(\mathtt{a}, [12..12])\}, 3)$ at $A[\mathtt{c}]$.*

R-enum executes the above procedure in $O(r)$ space using range distinct queries on $\mathsf{L}'[1..r]$ and RLBWT-based LF-mapping. The breadth-first traversal on $\mathcal{S}^{\mathsf{rmr}}$ can be made in $O(r)$ space because the space needed to store the rich representations for all strings in $\{ax \in \mathcal{S} \mid a \in \Sigma, x \in \mathcal{S}^{\mathsf{rmr}}, |x| = t\}$ for any $t \in [0..n]$ was proved to be $O(r)$ in Section 3.3 of [26]. The total number of outputs of range distinct queries is upper bounded by $O(n)$ because each output can be charged to a distinct Weiner link. Similarly, the total number of LF-mapping needed is bounded by $O(n)$.

The original r-enum used Theorem 5 to implement LF-mapping and runs in $O(n \log \log_w(n/r))$ time in total (excluding the time to output patterns for MAWs). We show that this can be improved to $O(n)$ time.

Theorem 13. *Given the RLBWT of T, r-enum runs in $O(r)$ working space and $O(n)$ time in addition to the time linear to the number of output strings.*

Proof. If $r = \Omega(n/\log n)$, the original r-enum with Theorem 5 is already upper bounded by $O(n)$ because $O(n \log \log_w(n/r)) = O(n \log \log_w \log n)$ and $w = \Theta(\log n)$. For the case with $r = o(n/\log n)$, we use the move data structure of Theorem 6 to implement LF-mapping, which can be constructed in $O(r \log r) = O(n)$ time. Since LF-mapping can be done in $O(1)$ time on the move data structure, the r-enum runs in $O(n)$ time. □

4.2 Extension for NSMRs, SMRs, and context diversities

In this subsection, we extend r-enum to compute the NSMRs and SMRs, and the context diversity for every RMR.

Recall that r-enum traverses $x \in \mathcal{S}^{\mathsf{rmr}}$ while computing $\{\mathsf{repr}(ax) \mid ax \in \mathcal{S}\}$ from $\mathsf{repr}(x) = ([p..q], \{(c_i, [p_i..q_i])\}_{i=1}^{k}, |x|)$. First, we can compute the context diversity $(|\mathsf{lc}(x)|, |\mathsf{rc}(x)|)$ of $x \in \mathcal{S}^{\mathsf{rmr}}$ because $|\mathsf{rc}(x)|$ is k and $|\mathsf{lc}(x)|$ is the number of outputs of a range distinct query on $\mathsf{L}[p..q]$. Since x is an SMR if and only if $|\mathsf{lc}(x)| = |\mathsf{rc}(x)| = |\mathsf{Occ}(x)| = q - p + 1$, the context diversity has enough information to choose SMRs from RMRs. On the other hand, NSMRs cannot be determined from the context diversities.

To determine if $x \in \mathcal{S}^{\mathsf{rmr}}$ is an NSMR or not, we check if there is a net occurrence of x in $\{\mathsf{SA}[j] \mid j \in [p..q]\}$ using the array $A[1..\sigma]$ that stores $\mathsf{rlist}(ax)$ at $A[a]$ for any $a \in \mathsf{lc}(x) \setminus \{\$\}$. Thanks to Theorem 7, we only have to consider $\mathsf{SA}[p_i]$ for intervals $[p_i..q_i]$ with $p_i = q_i$ as candidates of net occurrences. For an interval $[p_i..q_i]$ with $p_i = q_i$, we check if $a = \mathsf{L}[p_i]$ is \$ or $\mathcal{I}(ax)$ computed in $A[a]$ is a singleton, and then, $\mathsf{SA}[p_i]$ is a net occurrence if and only if the condition holds. We can also compute $\mathsf{NOcc}(x)$ if we store $\mathsf{SA}[i]$ for all run's boundaries i, which can be precomputed using LF-mapping $O(n)$ times.

Since we can check the conditions for NSMRs and SMRs along with the traversal of $\mathcal{S}^{\mathsf{rmr}}$, the time and space complexities of r-enum are retained.

Theorem 14. *Given the RLBWT of T, we can compute the NSMRs and SMRs, and the context diversity for every RMR in $O(n)$ time and $O(r)$ working space. The net occurrences of output patterns can also be computed in the same time and space complexities.*

5 An $O(r)$-size data structure for NF-queries

In this section, we tackle the problem called Single-NF [10] on RLBWTs: We consider preprocessing the RLBWT of T to construct a data structure of size $O(r)$ so that we can support *NF-queries* of computing $\mathsf{NF}(P)$ for any query pattern P.

We define the *reversed trie* for a set S of strings as follows:

- The nodes consist of the suffixes of strings in S.
- There is an edge from node x to ax with a character a.

Theorem 15. *Given the RLBWT of T, we can build an $O(r)$-size data structure in $O(r)$ space and in $O(n)$ time with high probability or deterministic $O(n+|\mathcal{S}^{\mathsf{nsmr}}| \log\log \min(\sigma, |\mathcal{S}^{\mathsf{nsmr}}|))$ time to support NF-queries for any query pattern P in $O(|P|)$ time.*

Proof. We execute r-enum of Theorem 14, which actually performs a breadth-first traversal of the reversed trie for $\mathcal{S}^{\mathsf{rmr}}$ in $O(n)$ time and $O(r)$ space. Our idea is to build the compacted reversed trie $\mathcal{T}$ for $\mathcal{S}^{\mathsf{nsmr}}$ during the traversal, which can be stored in $O(|\mathcal{S}^{\mathsf{nsmr}}|) = O(r)$ space because $|\mathcal{S}^{\mathsf{nsmr}}| \le 2r$ by Theorem 9. For each node x of $\mathcal{T}$, which corresponds to a string in $\mathcal{S}^{\mathsf{nsmr}}$ or a branching node, we store the string length $|x|$ and an integer i_x in $\mathcal{I}(x)$ so that the edge label between x and its parent y can be retrieved by using FL-mapping $|x| - |y|$ times from i_x. For a node $x \in \mathcal{S}^{\mathsf{nsmr}}$, we also store the net frequency of x. See Figure 2 for an illustration of $\mathcal{T}$.

We show that we can build $\mathcal{T}$ in $O(n)$ time and $O(r)$ working space during the breadth-first traversal of r-enum. For each string depth t, we maintain the compacted reversed trie $\mathcal{T}_t$ for the set $\mathcal{S}_t^{\mathsf{rmr}} \cup \mathcal{S}_{\le t}^{\mathsf{nsmr}}$, where $\mathcal{S}_t^{\mathsf{rmr}}$ is the set of RMRs of length t and $\mathcal{S}_{\le t}^{\mathsf{nsmr}}$ is the set of NSMRs of length $\le t$. Since $|\mathcal{S}_t^{\mathsf{rmr}}| = O(r)$ and $|\mathcal{S}_{\le t}^{\mathsf{nsmr}}| = O(r)$, the size of $\mathcal{T}_t$ is $O(r)$. In the next round of the breadth-first traversal, r-enum extends every string in $\mathcal{S}_t^{\mathsf{rmr}}$ by one

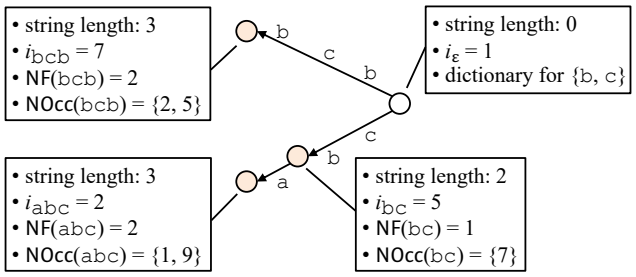

Figure 2: An illustration of the compacted reversed trie $\mathcal{T}$ for $\mathcal{S}^{\mathsf{nsmr}} = \{\mathtt{bc}, \mathtt{abc}, \mathtt{bcb}\}$ in our running example $T = \mathtt{abcbbcbcabc\$}$. A node corresponding to an NSMR is highlighted. Storing $\mathsf{NOcc}(\cdot)$ is optional. Note that edge labels are not stored explicitly. For example, the string $\mathtt{bcb}$ on the edge from ε to $\mathtt{bcb}$ is retrieved using FL-mapping $i_{\mathtt{bcb}} - i_{\varepsilon} = 3$ times from $i_{\mathtt{bcb}} = 7$ when necessary.

character to the left to get $\mathcal{S}^{\mathsf{rmr}}_{t+1}$. We update $\mathcal{T}_t$ to $\mathcal{T}_{t+1}$ by processing $x \in \mathcal{S}^{\mathsf{rmr}}_t$ as follows, where $A_x = \{ax \mid ax \in \mathcal{S}^{\mathsf{rmr}}_{t+1}\}$:

- If $|A_x| \geq 2$, we add new edges from x to the strings in A_x.
- If $|A_x| = 1$, we add a new edge from x to the string in A_x, and remove the non-branching node x if x is not an NSMR.
- If $|A_x| = 0$ and $x \notin \mathcal{S}^{\mathsf{nsmr}}$, we remove the node x and the edge to its parent y. We also remove y, if it becomes a non-branching node not in $\mathcal{S}^{\mathsf{nsmr}}$.

In total, these tasks can be done in $O(n)$ time and $O(r)$ space along with r-enum of Theorem 14. For each branching node of $\mathcal{T}$ with $k \leq \min(\sigma, |\mathcal{S}^{\mathsf{nsmr}}|)$ outgoing edges, we build a dictionary of Theorem 1 in $O(k)$ space and $O(k)$ time w.h.p. or deterministic $O(k \log\log \min(\sigma, k))$ time so that we can decide which child to proceed by the first characters of edges in $O(1)$ time. The construction time of the dictionaries for all branching nodes is bounded by $O(|\mathcal{S}^{\mathsf{nsmr}}|)$ time w.h.p. or deterministic $O(|\mathcal{S}^{\mathsf{nsmr}}| \log\log \min(\sigma, |\mathcal{S}^{\mathsf{nsmr}}|))$ time.

Given a query pattern P, we read the characters of P from right to left and traverse $\mathcal{T}$ from the root in $O(|P|)$ time. If we can reach a node $P \in \mathcal{S}^{\mathsf{nsmr}}$, we output $\mathsf{NF}(P)$ stored in the node. □

We can also support queries of computing the net occurrences of a query pattern P in $O(|P| + |\mathsf{NOcc}(P)|)$ time by storing $\mathsf{NOcc}(x)$ for each node $x \in \mathcal{S}^{\mathsf{nsmr}}$ in $\mathcal{T}$, which can be stored in $O(r)$ space in total due to Theorem 9.

6 Conclusions

We improved the time complexity of r-enum, and extended its enumeration power for other context-sensitive repeats in the literature. We also proposed the first data structure of size $O(r)$ to support NF-queries in optimal time. As a combinatorial property, we formally proved that the total number of net occurrences, as well as MUSs, is upper bounded by $O(r)$. We finally remark that our work leads to the first algorithm to compute the sorted list of net occurrences and MUSs in $O(n)$ time and $O(r)$ space: We enumerate all net occurrences by Theorem 13, and sort them in $O(n)$ time and $O(r)$ space by [25, Lemma 13].

Acknowledgements

Tomohiro I was supported by JSPS KAKENHI (Grant Numbers JP24K02899) and JST AIP Acceleration Research JPMJCR24U4, Japan.

References

[1] Verónica Becher, Alejandro Deymonnaz, and Pablo Ariel Heiber. Efficient computation of all perfect repeats in genomic sequences of up to half a gigabyte, with a case study on the human genome. *Bioinform.*, 25(14):1746–1753, 2009.

[2] Djamal Belazzougui, Fabio Cunial, Juha Kärkkäinen, and Veli Mäkinen. Linear-time string indexing and analysis in small space. *ACM Trans. Algorithms*, 16(2):17:1–17:54, 2020.

[3] Djamal Belazzougui and Gonzalo Navarro. Optimal lower and upper bounds for representing sequences. *ACM Trans. Algorithms*, 11(4):31:1–31:21, 2015.

[4] Anselm Blumer, J. Blumer, David Haussler, Ross M. McConnell, and Andrzej Ehrenfeucht. Complete inverted files for efficient text retrieval and analysis. *J. ACM*, 34(3):578–595, 1987.

[5] Anselm Blumer, Janet Blumer, David Haussler, Andrzej Ehrenfeucht, M. T. Chen, and Joel Seiferas. The smallest automaton recognizing the subwords of a text. *Theoretical Computer Science*, 40:31–55, 1985.

[6] Michael Burrows and David J Wheeler. A block-sorting lossless data compression algorithm. Technical report, HP Labs, 1994.

[7] Matthias Gallé. The bag-of-repeats representation of documents. In Gareth J. F. Jones, Paraic Sheridan, Diane Kelly, Maarten de Rijke, and Tetsuya Sakai, editors, *Proc. 36th International ACM SIGIR conference on research and development in Information Retrieval 2013*, pages 1053–1056. ACM, 2013.

[8] Matthias Gallé and Matías Tealdi. On context-diverse repeats and their incremental computation. In Adrian-Horia Dediu, Carlos Martín-Vide, José Luis Sierra-Rodríguez, and Bianca Truthe, editors, *Proc. 8th International Conference on Language and Automata Theory and Applications (LATA) 2014*, volume 8370 of *Lecture Notes in Computer Science*, pages 384–395. Springer, 2014.

[9] Matthias Gallé and Matías Tealdi. *xkcd*-repeats: A new taxonomy of repeats defined by their context diversity. *J. Discrete Algorithms*, 48:1–16, 2018.

[10] Peaker Guo, Patrick Eades, Anthony Wirth, and Justin Zobel. Exploiting new properties of string net frequency for efficient computation. In Shunsuke Inenaga and Simon J. Puglisi, editors, *Proc. 35th Annual Symposium on Combinatorial Pattern Matching (CPM) 2024*, volume 296 of *LIPIcs*, pages 16:1–16:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2024.

[11] Peaker Guo and Kaisei Kishi. Net occurrences in fibonacci and thue-morse words. In Paola Bonizzoni and Veli Mäkinen, editors, *Proc. 36th Annual Symposium on Combinatorial Pattern Matching (CPM) 2025*, volume 331 of *LIPIcs*, pages 16:1–16:22. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2025.

[12] Peaker Guo, Seeun William Umboh, Anthony Wirth, and Justin Zobel. Online computation of string net frequency. In Zsuzsanna Lipták, Edleno Silva de Moura, Karina Figueroa, and Ricardo Baeza-Yates, editors, *Proc. 31st International Symposium on String Processing and Information Retrieval (SPIRE) 2024*, volume 14899 of *Lecture Notes in Computer Science*, pages 159–173. Springer, 2024.

[13] Dan Gusfield. *Algorithms on Strings, Trees, and Sequences - Computer Science and Computational Biology*. Cambridge University Press, 1997.

[14] Lucian Ilie and William F. Smyth. Minimum unique substrings and maximum repeats. *Fundam. Informaticae*, 110(1-4):183–195, 2011.

[15] Shunsuke Inenaga. Faster and simpler online computation of string net frequency, 2024. arXiv:2410.06837.

[16] Shunsuke Inenaga, Hiromasa Hoshino, Ayumi Shinohara, Masayuki Takeda, Setsuo Arikawa, Giancarlo Mauri, and Giulio Pavesi. On-line construction of compact directed acyclic word graphs. *Discrete Applied Mathematics*, 146(2):156–179, 2005.

[17] Yih-Jeng Lin and Ming-Shing Yu. Extracting chinese frequent strings without dictionary from a chinese corpus, its applications. *J. Inf. Sci. Eng.*, 17(5):805–824, 2001.

[18] Yih-Jeng Lin and Ming-Shing Yu. The properties and further applications of chinese frequent strings. *Int. J. Comput. Linguistics Chin. Lang. Process.*, 9(1), 2004.

[19] Udi Manber and Eugene W. Myers. Suffix arrays: A new method for on-line string searches. *SIAM J. Comput.*, 22(5):935–948, 1993.

[20] Tomonari Masada, Atsuhiro Takasu, Yuichiro Shibata, and Kiyoshi Oguri. Clustering documents with maximal substrings. In *Proc. 13th International Conference on Enterprise Information Systems ICEIS 2011*, pages 19–34, 2011.

[21] Takuya Mieno and Shunsuke Inenaga. Space-efficient online computation of string net occurrences. In Paola Bonizzoni and Veli Mäkinen, editors, *Proc. 36th Annual Symposium on Combinatorial Pattern Matching (CPM) 2025*, volume 331 of *LIPIcs*, pages 23:1–23:13. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2025.

[22] S. Muthukrishnan. Efficient algorithms for document retrieval problems. In *Proc. 13th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA) 2002*, pages 657–666, 2002.

[23] Jacques Nicolas, Christine Rousseau, Anne Siegel, Pierre Peterlongo, François Coste, Patrick Durand, Sébastien Tempel, Anne-Sophie Valin, and Frédéric Mahé. Modeling local repeats on genomic sequences. Research Report RR-6802, INRIA, 2008. URL: `https://inria.hal.science/inria-00353690`.

[24] Takaaki Nishimoto and Yasuo Tabei. Optimal-time queries on bwt-runs compressed indexes. In Nikhil Bansal, Emanuela Merelli, and James Worrell, editors, *Proc. 48th International Colloquium on Automata, Languages and Programming (ICALP) 2021*, volume 198 of *LIPIcs*, pages 101:1–101:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021.

[25] Takaaki Nishimoto and Yasuo Tabei. Optimal-time queries on bwt-runs compressed indexes, 2021. arXiv:2006.05104v3.

[26] Takaaki Nishimoto and Yasuo Tabei. R-enum: Enumeration of characteristic substrings in BWT-runs bounded space. In Pawel Gawrychowski and Tatiana Starikovskaya, editors, *Proc. 32nd Annual Symposium on Combinatorial Pattern Matching (CPM) 2021*, volume 191 of *LIPIcs*, pages 21:1–21:21. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021.

[27] Enno Ohlebusch and Timo Beller. Alphabet-independent algorithms for finding context-sensitive repeats in linear time. *J. Discrete Algorithms*, 34:23–36, 2015.

[28] Enno Ohlebusch, Thomas Büchler, and Jannik Olbrich. Faster computation of chinese frequent strings and their net frequencies. In Zsuzsanna Lipták, Edleno Silva de Moura, Karina Figueroa, and Ricardo Baeza-Yates, editors, *Proc. 31st International Symposium on String Processing and Information Retrieval (SPIRE) 2024*, volume 14899 of *Lecture Notes in Computer Science*, pages 249–256. Springer, 2024.

[29] Daisuke Okanohara and Jun'ichi Tsujii. Text categorization with all substring features. In *Proc. SIAM International Conference on Data Mining (SDM) 2009*, pages 838–846, 2009.

[30] Milan Ruzic. Constructing efficient dictionaries in close to sorting time. In *Proc. 35th International Colloquium on Automata, Languages and Programming (ICALP) 2008*, pages 84–95, 2008.

[31] Peter Weiner. Linear pattern-matching algorithms. In *Proc. 14th IEEE Ann. Symp. on Switching and Automata Theory*, pages 1–11, 1973.

[32] Dan E. Willard. Examining computational geometry, van emde boas trees, and hashing from the perspective of the fusion tree. *SIAM J. Comput.*, 29(3):1030–1049, 2000.