Computing Smallest and Largest Repetition Factorizations in $O(n \log n)$ Time

Hiroe Inoue[1], Yoshiaki Matsuoka[1], Yuto Nakashima[1,2], Shunsuke Inenaga[1], Hideo Bannai[1], and Masayuki Takeda[1]

[1] Department of Informatics, Kyushu University, Japan
[2] Japan Society for the Promotion of Science (JSPS), Japan
{hiroe.inoue, yuto.nakashima, inenaga, bannai, takeda}@inf.kyushu-u.ac.jp

Abstract. A factorization $f_1, \ldots, f_m$ of a string w is called a *repetition factorization* of w if each factor f_i is a repetition, namely, $f_i = x^k x'$ for some non-empty string x, an integer $k \geq 2$, and x' being a proper prefix of x. Dumitran et al. (Proc. SPIRE 2015) proposed an algorithm which computes a repetition factorization of a given string w in $O(n)$ time, where n is the length of w. In this paper, we propose two algorithms which compute smallest/largest repetition factorizations in $O(n \log n)$ time. The first algorithm is a simple $O(n \log n)$ space algorithm while the second one uses only $O(n)$ space.

1 Introduction

A *factorization* of a string w is a sequence $f_1, \ldots, f_m$ of non-empty substrings of w such that $w = f_1 \cdots f_m$. The *size* of the factorization is the number m of factors contained in the factorization. Numerous types of factorizations of strings have been considered, of which the most well-studied is the Lempel-Ziv factorizations and its family [22,23,19,21]. Not only do they have an apparent application to data compression, but also they play key roles in other stringology problems [14,13,20]. The Lyndon factorizations [3] are classical subjects in combinatorics on words, and also have some application in data compression, i.e., in a variant of the Burrows-Wheeler transform [15]. It is known that given a string of length n, these factorizations for the string can be computed in $O(n)$ time [5,8,18].

Recently, the problems of factorizing a given string into some kinds of "combinatorial" structures have been studied [1,2,12,9]. In this paper, we are particularly interested in the following types of factorizations: A factorization $f_1, \ldots, f_m$ of a string w is said to be a *square factorization* of w if each factor f_i is a square (i.e., f_i is of form x^2 for some string x), and it is called a *repetition factorization* of w if each factor f_i is a repetition (i.e., f_i is of form $x^k x'$ with $k \geq 2$ and x' begin a proper prefix of x). Dumitran et al. showed how to compute a square factorization of an input string of length n in $O(n \log n)$ time, and a repetition factorization in $O(n)$ time [7]. Very recently, Matsuoka et al. [17] proposed an improved algorithm which finds a square factorization in $O(n)$ time, and also proposed algorithms which compute square factorizations of smallest/largest size in $O(n \log n)$ time.

In this paper, we tackle the problems of computing repetition factorizations of smallest/largest sizes of a given string w of length n, and show two algorithms for computing such factorizations in $O(n \log n)$ time. The first algorithm, which is based on a reduction of the problem to the classical shortest/longest path problem on a DAG, requires $O(n \log n)$ space as the underlying DAG requires $O(n \log n)$ space. On

Hiroe Inoue, Yoshiaki Matsuoka, Yuto Nakashima, Shunsuke Inenaga, Hideo Bannai, Masayuki Takeda: Computing Smallest and Largest Repetition Factorizations in $O(n \log n)$ Time, pp. 135–145.
Proceedings of PSC 2016, Jan Holub and Jan Žďárek (Eds.), ISBN 978-80-01-05996-8

the other hand, the second algorithm shaves the space requirement to $O(n)$ but retains $O(n \log n)$ running time, by simulating the first one with dynamic programming.

The rest of the paper is organized as follows. In Section 2, we state some notations and definitions on strings. In Section 3, we show how to find smallest/largest repetition factorizations in $O(n \log n)$ time and space. Section 4 shows an $O(n \log n)$-time and $O(n)$-space solution to the problem. Section 5 concludes and states some future work.

2 Preliminaries

2.1 Strings

Let Σ be a finite *alphabet*. An element of Σ^* is called a *string*. The length of a string w is denoted by $|w|$. The empty string ε is a string of length 0, namely, $|\varepsilon| = 0$. Let Σ^+ be the set of non-empty strings, i.e., $\Sigma^+ = \Sigma^* - \{\varepsilon\}$. For a string $w = xyz$, x, y and z are called a *prefix*, *substring*, and *suffix* of w, respectively. A prefix x is called a *proper prefix*, if $x \neq w$.

The i-th character of a string w is denoted by $w[i]$, where $1 \leq i \leq |w|$. For a string w and two integers $1 \leq i \leq j \leq |w|$, let $w[i..j]$ denote the substring of w that begins at position i and ends at position j. For convenience, let $w[i..j] = \varepsilon$ when $i > j$.

For any string w, let $w^1 = w$, and for any integer $k \geq 2$ let $w^k = ww^{k-1}$. A non-empty string w is called *primitive* if there is no string x s.t. $w = x^k$ for some integer $k \geq 2$.

An integer $p \geq 1$ is said to be a *period* of a string w if $w[i] = w[i + p]$ for all $1 \leq i \leq |w| - p$. The following well-known *periodicity lemma* is useful.

Lemma 1 (Periodicity Lemma [10]). *If two periods p, q of string w of length n satisfies $p + q - \gcd(p, q) \leq n$, then $\gcd(p, q)$ is also a period of w.*

2.2 Repetitive structures in strings

We define repetitive structures used in this paper below.

Definition 2 (Squares). *A non-empty string s is said to be a* square*, if $s = x^2$ for some string x.*

A square x^2 is called a *primitively rooted square* if x is primitive.

Definition 3 (Repetitions). *A triple (beg, end, p) is said to be a* repetition *of a string w, if the smallest period p of the substring $w[beg..end]$ satisfies $|w[beg..end]| \geq 2p$. In other words, the substring $w[beg..end]$ is of form $x^k x'$, where $x = w[beg..beg + p - 1]$ is primitive, $k \geq 2$, and x' is a possibly empty proper prefix of x.*

We will sometimes identify the triple (beg, end, p) as the corresponding substring $w[beg..fin]$ with the smallest period p, and will call the substring $w[beg..fin]$ as a repetition.

Definition 4 (Runs (maximal repetitions)). *A repetition (beg, end, p) is said to be a* run *(or a* maximal repetition*) of a string w of length n, if $beg = 1$ or $w[beg - 1] \neq w[beg + p - 1]$, and $end = n$ or $w[end + 1] \neq w[end - p + 1]$. In other words, the periodicity of the substring $w[beg..end]$ with the smallest period p cannot be extended to the left nor the right.*

Let $Runs(w)$ denote the set of runs of string w. The following result is well-known.

Theorem 5 ([4]). *For any string w of length n, $|Runs(w)| < n$. Also, $Runs(w)$ can be computed in $O(n)$ time for integer alphabets of size $n^{O(1)}$.*

In this paper, we suppose that the runs in $Runs(w)$ are sorted in increasing order of beginning positions and the i-th run in $Runs(w)$ is denoted by $r_i = (beg_i, end_i, p_i)$ for any $1 \leq i \leq |Runs(w)|$ (i.e., $beg_i \leq beg_{i+1}$ for any $1 \leq i < |Runs(w)|$). This makes it easy for us to explain our algorithm, and we can sort the runs in $Runs(w)$ in this order in $O(n)$ time by a bucket sort.

The following observation shows a relation between *runs* and *repetitions*.

Observation 1 *For any repetition $w[i..j]$ of the shortest period s occurring in a string w, there exist unique strings $x \in \Sigma^+$ and $y \in \Sigma^*$ s.t. x is a primitively rooted square of length $2s$ and $w[i..j] = xy$. For any repetition $w[i..j]$ of the shortest period s occurring in a string w, there exists a unique run $r = (beg, end, p)$ s.t. $beg \leq i < j \leq end$ and $s = p$.*

2.3 Problems

In this section, we formally define the problems we consider in this paper.

Definition 6 (Repetition factorizations). *A sequence $f_1, \ldots, f_m$ of non-empty strings is said to be a* repetition factorization *of a string w, if $w = f_1 \cdots f_m$ and f_i for each $1 \leq i \leq m$ is a repetition of w.*

Each f_i in a repetition factorization $f_1, \ldots, f_m$ of a string w is called a *factor* of the factorization. The *size* of the repetition factorization is the number m of factors in the factorization.

The problem we tackle in this paper is the following:

Problem 7 (Smallest/Largest Repetition Factorization). Given a string w of length n, compute a repetition factorization of smallest/largest size.

Example 8. If we are given a string $w =$ `abaabaababaabaabababa`, then we return one of the following as a largest repetition factorization of w.

- `abaaba|abab|aabaab|ababa`
- `abaaba|abab|aabaaba|baba`
- `abaaba|ababa|abaaba|baba`
- `abaabaa|baba|abaaba|baba`

We return one of the following as a smallest repetition factorization of w.

- `abaabaababaabaab|ababa`
- `abaabaababaabaaba|baba`

2.4 Graphs

Our solutions to Problem 7 are based on a certain DAG defined over the runs appearing in a given string.

For any DAG G, let $\pi_1 = v_i \cdots v_j$ and $\pi_2 = v_j \cdots v_k$ be any paths on G (v_ℓ is a node of G for all $i \leq \ell \leq k$). We denote the concatenated path $v_i \cdots v_k$ by $\pi_1 \oplus \pi_2$.

3 $O(n \log n)$-time and space solution

In this section, we show an $O(n \log n)$-time and space solution to Problem 7. Our main idea is to reduce the problem to a shortest/longest path problem on a directed acyclic graph. In Section 3.1, we define a DAG G that is based on the runs of the input string w. We will call G as the *repetition graph* of w.

In this paper, we focus on the largest repetition factorization problem, but the smallest repetition factorization problem can be solved in a similar way.

3.1 Definition of repetition graphs

The repetition graph $G = (V, E)$ of a string w is an edge-weighted directed acyclic graph. First, we define the set V of nodes. It consists of the two mutually disjoint subsets V' and V'' of nodes, namely $V = V' \cup V''$ such that

$$V' = \{(0, j) \mid 0 \leq j \leq n\},$$
$$V'' = \bigcup_{i=1}^{|Runs(w)|} V''_i,$$

where $V''_i = \{(i, j) \mid beg_i + 2p_i - 1 \leq j \leq end_i\}$ for each $r_i = (beg_i, end_i, p_i) \in Runs(w)$.

The set E of edges consists of the three mutually disjoint subsets E', E'', and E''' of edges, namely $E = E' \cup E'' \cup E'''$, such that

$$E' = \{((i_1, j_1), (i_2, j_2)) \mid j_2 - j_1 = 2p_{i_2}, (i_1, j_1) \in V', (i_2, j_2) \in V''\},$$
$$E'' = \{((i_1, j_1), (i_2, j_2)) \mid i_1 = i_2, j_1 + 1 = j_2, (i_1, j_1), (i_2, j_2) \in V''\},$$
$$E''' = \{((i_1, j_1), (i_2, j_2)) \mid j_1 = j_2, (i_1, j_1) \in V'', (i_2, j_2) \in V'\}.$$

The weight of each edge $e \in E$, denoted $weight(e)$, is defined by

$$weight(e) = \begin{cases} 1 & \text{if } e \in E', \\ 0 & \text{if } e \in E'' \cup E'''. \end{cases}$$

We call the nodes $v_s = (0, 0) \in V'$ and $(0, n) \in V'$ as the *starting node* and *ending node* of the graph G, respectively.

For any node $v = (i, j)$, let $run(v) = i$ and $pos(v) = j$. Intuitively, each node $v = (run(v), pos(v))$ in V'' corresponds to the $run(v)$-th run in $Runs(w)$ and the position $pos(v)$ in w satisfying $beg_i + 2p_i - 1 \leq j \leq end_i$. Any node $v \in V'$ does not correspond to any run, but for convenience we let $run(v) = 0$. We also define an auxiliary node $v_s = (0, 0)$ for convenience. We remark that although $pos(v_s) = 0$, the index of any string w begins with 1 throughout this paper.

We describe an intuitive explanation for the repetition graphs (more details will be given in the next subsection). See Figure 1 which shows an example of a repetition graph. Let (v_1, v_2) be an edge in E' (drawn as a diagonal arrow). Edge (v_1, v_2) represents a primitively rooted square $w[pos(v_1) + 1..pos(v_2)]$ of length $2p_{run(v_2)}$. By Observation 1, any repetition z can be decomposed as xy, where x is a prefix of z which is the primitively rooted square having the same smallest period as z, and y is the remainder (possibly empty). The repetition graph G for string w represents every repetition in w as a path which is a concatenation of a diagonal arrow (i.e. the primitively rooted square part) and some horizontal arrows (i.e. the remainder). For

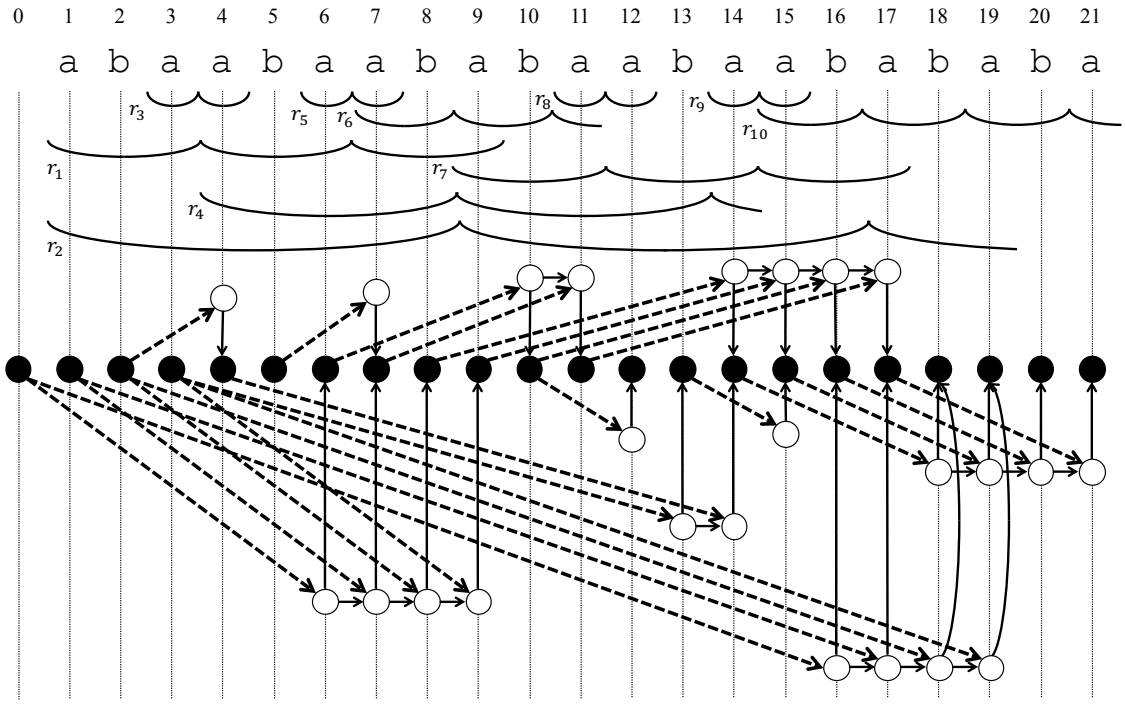

Figure 1. The repetition graph G of string abaabaababaabaabababa. Black circles represent the nodes in V' and white circles represent the nodes in V''. Dashed arrows represents the edges in E', horizontal solid arrows represent the edges in E'', and vertical solid arrow represent the edges in E'''.

example, a repetition $w[10..17] =$ baabaaba in Figure 1 is represented by the path which is a concatenation of the diagonal arrow from the black node at position 9 to a white node at position 15, and the horizontal arrows from its white node at position 15 to the upper white node at position 17.

3.2 Relations between repetition factorizations and the repetition graph

We explain a correspondence between a path on G and a repetition factorization of w. For any integer $1 \leq t \leq n$, a path from node v_s to node $(0,t) \in V'$ is called an *s-t path* of G. Let $F = f_1, \ldots, f_M$ be a repetition factorization of $w[1..t]$ and $(0,0) = v_1, \ldots, v_{M+1} = (0,t)$ be the sequence of nodes in V' on an s-t path π. We will say that F corresponds to π (or π corresponds to F), if the sequence $|f_1|, |f_1 f_2|, \ldots, |f_1 \cdots f_M| = t$ of the ending positions of all factors is equal to the sequence $pos(v_2), \ldots, pos(v_{M+1})$. The following lemma states a one-to-one correspondence between paths and repetition factorizations.

Lemma 9. *For any string w of length n and integer $1 \leq t \leq n$, there is a one-to-one correspondence between s-t paths on the repetition graph G of w, and repetition factorizations of $w[1..t]$.*

Proof. First, we show that for any s-t path in G there exists a unique repetition factorization of $w[1..t]$. Let $\pi = v_1 \cdots v_k$ be any s-t path, where $v_1 = v_s$ and $v_k = (0,t)$. By the definition of G, there is a unique decomposition $\pi_1, \ldots, \pi_{M+1}$ of path π such that $\pi_i = (v_{j(i)} v_{j(i)+1}) \oplus (v_{j(i)+1} \cdots v_{j(i+1)-1})$, $v_{j(i)} \in V', v_{j(i)+1}, \ldots, v_{j(i+1)-1} \in V''$, and

$\pi_{M+1} = v_k$ (see also Figure 2). We remark that the subpath $(v_{j(i)+1} \cdots v_{j(i+1)-1})$ is possibly empty. By the definition of G, if $v_{j(i)}v_{j(i)+1}$ exists on G (i.e. $(v_{j(i)}, v_{j(i)+1}) \in E'$), then $w[pos(v_{j(i)})+1..pos(v_{j(i)+1})]$ is a primitively rooted square of length $2p_{run(v_{j(i)+1})}$. If $v_{j(i)+1} \cdots v_{j(i+1)-1}$ exists on G, then $w[pos(v_{j(i)+1})+1..pos(v_{j(i+1)-1})]$ is a substring of a run $r = (beg, end, p)$ such that $beg + 2p \leq pos(v_{j(i)+1})+1 \leq pos(v_{j(i+1)-1}) \leq end$. Thus, repetition $w[pos(v_{j(i)})+1..pos(v_{j(i+1)-1})]$ has p as its shortest period if π_i exists on G. Therefore, π corresponds to a unique repetition factorization of $w[1..t]$ since the path decomposition $\pi_1, \ldots, \pi_{M+1}$ is unique and π_i corresponds to a repetition.

Second, we show that for any repetition factorization of $w[1..t]$, there exists a corresponding s-t path in G. Let F be any repetition factorization of $w[1..t]$. By the definition of G, there exists an s-t path π which corresponds to F. We show that π is the only path on G which corresponds to F. On the contrary, suppose that there are two distinct s-t paths which correspond to some repetition factorization F of $w[1..t]$. Because of this assumption, some factor $f_i = w[c..d]$ of F with $1 \leq c < d \leq t$ corresponds to two distinct paths from v_γ to v_δ, where $pos(v_\gamma) = c$, $pos(v_\delta) = d$, and no nodes v on the two paths satisfy $c < pos(v) < d$. This implies that the factor f_i has two periods p, q such that p is its shortest period, $\gcd(p, q) \neq p$, and $2p, 2q \leq |f_i|$. Since $p < q$, we have $p + q - \gcd(p, q) \leq 2q \leq |f_i|$. By Lemma 1, $\gcd(p, q)$ is also a period of f_i. Since $p \neq \gcd(p, q)$ and $p < q$, we have $\gcd(p, q) < p$. However this contradicts that p is the shortest period of f_i. Thus only one path can correspond to any repetition f_i in F, and the path π which corresponds to the repetition factorization F is unique. □

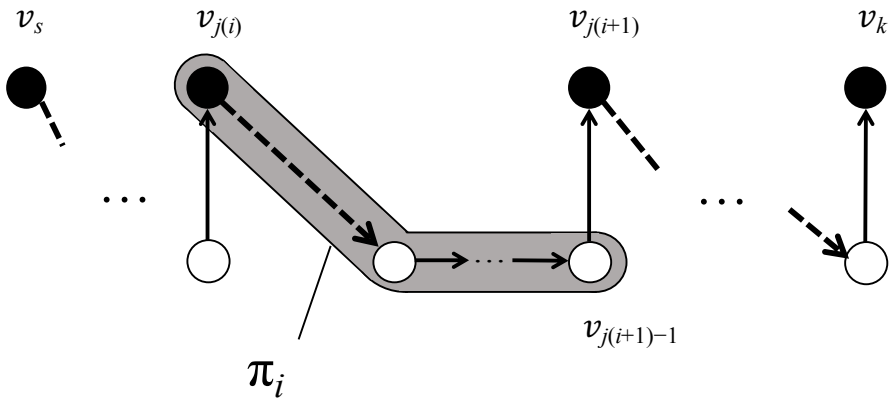

Figure 2. We can decompose an s-t path so that each factor corresponds to a sub-path from a node in V' to another node in V'.

We can regard each edge in E''' as a boundary of a repetition factorization. Figure 3 and Figure 4 show examples of a correspondence between the paths and factorizations for the largest and smallest problem, respectively.

For any $1 \leq t \leq n$, let $\pi = v_s \cdots (0, t)$ be an an s-t path on G. We denote by $RF_{w[1..t]}(\pi)$ the repetition factorization of $w[1..t]$ which corresponds to the s-t path π. One can see that a path from v_s to any node $v \in V''$ corresponds to a repetition factorization of $w[1..pos(v)]$ such that the shortest period of the rightmost factor is $p_{run(v)}$.

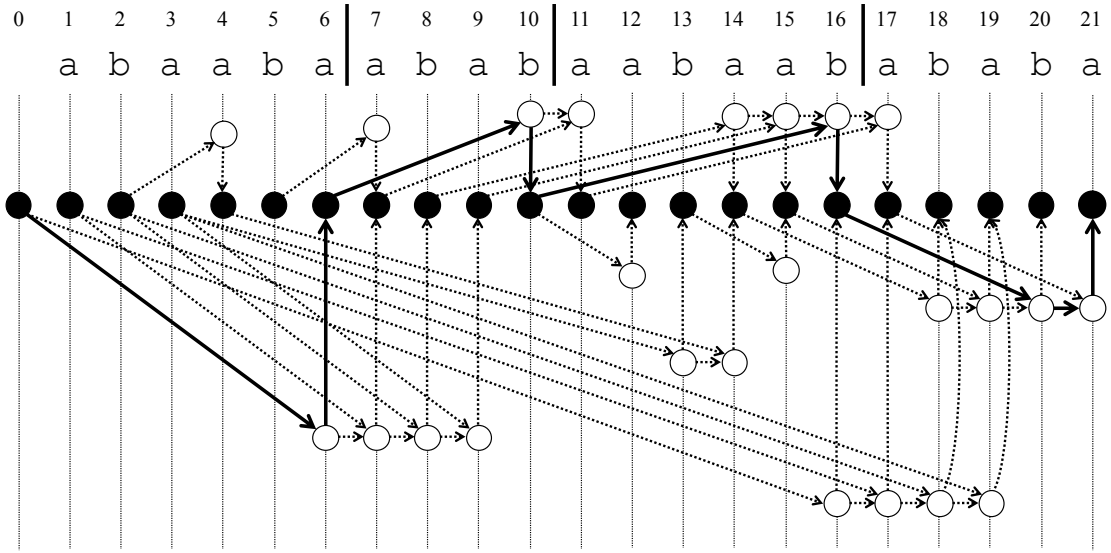

Figure 3. For the largest problem, one of answers is abaaba|abab|aabaab|ababa. The path consisting of bold arrows represents the *s*-*t* path which corresponds to abaaba|abab|aabaab|ababa.

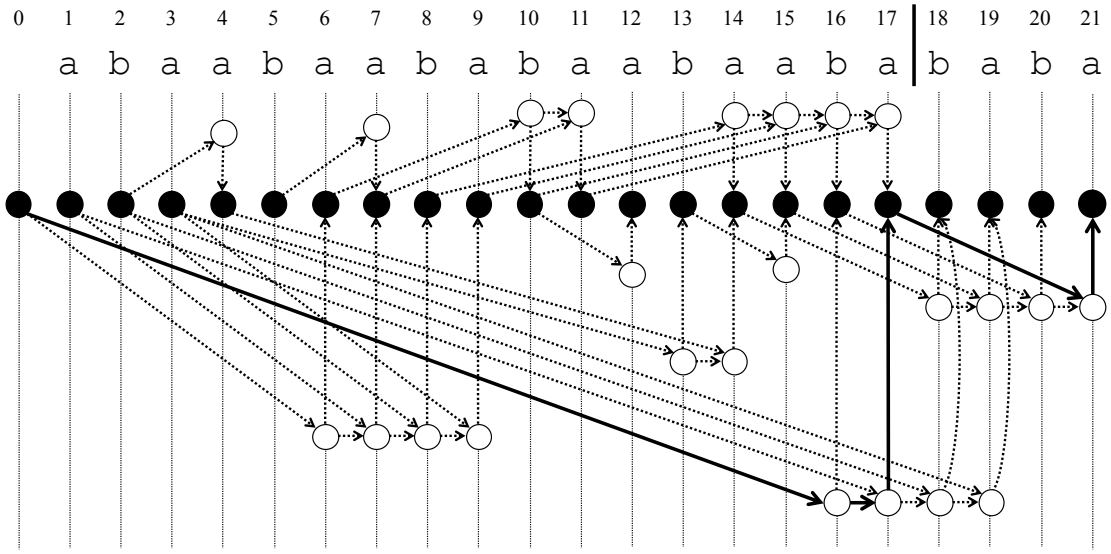

Figure 4. For the smallest problem, one of answers is abaabaababaabaaba|baba. The path consisting of bold arrows represents the *s*-*t* path which corresponds to abaabaababaabaaba|baba.

3.3 Reduction to the longest path problem

Now we describe a reduction from the problem of computing a repetition factorization of a given string w to the longest path problem on the repetition graph G for w.

Recall that for any edge $e \in E$, we defined $\mathit{weight}(e) = 1$ if $e \in E'$ and $\mathit{weight}(e) = 0$ otherwise. Let $\mathit{sum}(\pi)$ be the sum of $\mathit{weight}(e)$ for all edges e on the path π. The next lemma clearly holds.

Lemma 10. *For any s-t path $\pi = v_s \cdots (0, t)$, $\mathit{sum}(\pi) = |RF_{w[1..t]}(\pi)|$.*

It immediately follows from Lemma 10 that if $\pi_{\max}$ is a path from v_s to node $(0, t)$ of maximal total weight, then $\mathit{sum}(\pi_{\max})$ equals to the size of a largest repetition

factorization of $w[1..t]$. Thus, the problem of computing a largest repetition factorization of w reduces to the longest path problem on the repetition graph G, which can be solved in $O(|V| + |E|)$ time and space.

3.4 Complexity

The efficiency of our algorithm depends on the size of repetition graph G. We show upper and lower bounds of the size of repetition graphs.

Upper bound. We firstly show an upper bound of the size of G. It is clear that $|V'| \leq n + 1$ by the definition. The number of nodes $v \in V''$ s.t. $pos(v) = i$ is equal to the number of primitively rooted squares which end at position i. By applying the next lemma to all positions in the input string w, we can get $|V''| = O(n \log n)$.

Lemma 11 ([6]). *For any string x of length i, the number of primitively rooted squares that are suffixes of x is $O(\log i)$.*

Overall, we get $|V| = O(n \log n)$.

Now we analyze the number of edges in the graph G. For any node $v \in V''$, the number of incoming edges of v in E' is exactly 1 and the number of outgoing edges of v is at most 2 (i.e. exactly one edge in E''' and at most one edge in E''). Thus we can see $|E| = O(|V|) = O(n \log n)$. Because of this argument, we can solve the longest path problem on G in $O(n \log n)$ time after constructing G. In order to construct G, we need information of all runs. It follows from Theorem 5 that any string of length n contains less than n runs, and that all runs in a given string can be computed in linear time. Consequently, we obtain the following theorem.

Theorem 12. *Given a string w of length n, we can compute a largest repetition factorization of w in $O(n \log n)$ time and space.*

The following corollaries are immediate.

Corollary 13. *Given a string w of length n, we can compute the size of a largest repetition factorization of $w[1..i]$ for every $1 \leq i \leq n$ in $O(n \log n)$ total time and space.*

Corollary 14. *Given a string w of length n, we can compute the number of distinct repetition factorizations of w in $O(n \log n)$ time and space.*

Lower bound. As was shown above, the size of the repetition graph G is upper-bounded by $O(n \log n)$. This bound is indeed tight, namely, there exists a series of strings of which the repetition graphs contain $\Omega(n \log n)$ nodes and edges.

We consider the well-known *Fibonacci strings*:

Definition 15 (Fibonacci string [16]). *The k-th Fibonacci string Fib_k is recursively defined as follows.*

- $Fib_1 = b$,
- $Fib_2 = a$,
- $Fib_k = Fib_{k-1}Fib_{k-2}$ *for* $k \geq 3$.

Clearly, $|Fib_k| = F_k$, where F_k is the k-th Fibonacci number.

Example 16. $Fib_1 = b$, $Fib_2 = a$, $Fib_3 = ab$, $Fib_4 = aba$, $Fib_5 = abaab$, $Fib_6 = abaababa$, etc.

Fraenkel and Simpson [11] studied the number $R(k)$ of primitively rooted squares in k-th *Fibonacci string*. Let $\phi = \frac{1+\sqrt{5}}{2}$.

Lemma 17 ([11]). $R(k) = \frac{2}{5}(3-\phi)kF_k + O(F_k) = \frac{2(3-\phi)}{5\log\phi} F_k \log F_k + O(F_k)$ ($\frac{2(3-\phi)}{5\log\phi} \approx 0.7962$).

The next lemma immediately follows from Lemma 17.

Lemma 18. *For any Fibonacci string of length n, the size of G is $\Theta(n \log n)$.*

4 $O(n \log n)$-time and $O(n)$-space solution

In this section, we propose an $O(n \log n)$-time and $O(n)$-space solution to Problem 7. This space-efficient algorithm follows the idea of the algorithm in the previous section, but does not explicitly construct the repetition graph G which requires $O(n \log n)$ space. Instead, the space-efficient algorithm proposed in this section simulates the traversal on G using only $O(n)$ space, by dynamic programming. In this section, we again focus on the largest repetition factorization problem, but the smallest version can also be solved analogously.

4.1 Simulating the previous algorithm

Each node v of G is assigned to a value $value(v)$, defined as follows: $value(v)$ is the size of a largest repetition factorization of $w[1..pos(v)]$ if $v \in V'$, $value(v)$ is the size of a largest repetition factorization of $w[1..pos(v)]$ s.t. the shortest period of the rightmost factor is $p_{run(v)}$ if $v \in V''$. It is easy to see that $value(v)$ is equal to the value for v which is computed by the longest path problem on G. Thus, it suffices to compute $value(v)$ for all nodes $v \in V$. This can be done using the following formula. For any $v \in V'$, let U_v be the set of nodes u such that $(u, v) \in E$, and for any $v \in V''$, let u_1 and u_2 be the unique two nodes such that $(u_1, v) \in E$ and $u_1 \in V'$, and $(u_2, v) \in E$ and $u_2 \in V''$, respectively.

$$value(v) = \begin{cases} \max\{value(u) \mid u \in U_v\} & \text{if } v \in V', \quad (1) \\ \max\{value(u_1) + 1, value(u_2)\} & \text{if } v \in V''. \quad (2) \end{cases}$$

Let RFA_w be an array of length n s.t. $RFA_w[i]$ stores the size of a largest repetition factorization of $w[1..i]$, namely, $RFA_w[i] = value((0, i))$. Our new algorithm computes RFA_w from left to right. We suppose that $RFA_w[1..i-1]$ is already computed. Below, we show how to compute $RFA_w[i]$.

Let $subRuns_w[i]$ be a set of indices j s.t. $beg_j + 2p_j - 1 < i \leq end_j$ (i.e. a set of indices of runs which correspond to a node $v \in V''$ s.t. $pos(v) = i$). In order to compute $RFA_w[i]$, we need $value(v)$ s.t. $run(v) \in subRuns_w[i]$ due to Equation (1). We also suppose that for each $j' \in subRuns_w[i-1]$, the run $r_{j'}$ maintains $value(v)$ s.t. $run(v) = j'$ and $pos(v) = i-1$. For any $j \in subRuns_w[i]$, we can compute $value((j, i))$ by Equation (2) (since we know $value((j, i-1))$ and $RFA_w[i - 2p_j]$). Thus we can compute $value((j, i))$ in $O(1)$ time for each $j \in subRuns_w[i]$. If we have $subRuns_w[i]$, then we can compute $RFA_w[i]$ in $O(\log n)$ time, since $|subRuns_w[i]| = O(\log n)$.

Finally, we show how to compute $subRuns_w[i]$. Let $I_b(i) = \{j \mid beg_j + 2p_j - 1 = i (1 \leq j \leq |Runs(w)|)\}$, and let $I_e(i) = \{j \mid end_j = i (1 \leq j \leq |Runs(w)|)\}$. We can compute $I_b(i)$ and $I_e(i)$ for all i in $O(n)$ time and these sets takes $O(n)$ space. We assume that $subRuns_w[i-1]$ has already computed. It is easy to see that we can compute $subRuns_w[i]$ by removing j s.t. $j \in I_e(i-1)$ from $subRuns_w[i-1]$ and by adding j s.t. $j \in I_b(i)$ to $subRuns_w[i-1]$. From these operations, we can compute $subRuns_w[i]$ in $O(\log n)$ time if we have $I_b(i)$ and $I_e(i-1)$.

4.2 Complexity

First, we compute $Runs(w)$ and construct $I_b(i)$ and $I_e(i)$ for all i in $O(n)$ time and these sets takes $O(n)$ space. This requires $O(n \log n)$ time and $O(n)$ space by Theorem 5. As was explained, for each position i in w, $RFA_w[i]$ can be computed in $O(\log n)$ time, and RFA_w requires a total of $O(n)$ space. We have proved the next theorem and corollaries.

Theorem 19. *Given a string w of length n, we can compute a largest repetition factorization of w in $O(n \log n)$ time and $O(n)$ space.*

Corollary 20. *Given a string w of length n, we can compute the size of a largest repetition factorization of $w[1..i]$ for every $1 \leq i \leq n$ in $O(n \log n)$ total time and $O(n)$ total space.*

Corollary 21. *Given a string w of length n, we can compute the number of distinct repetition factorizations of w in $O(n \log n)$ time and $O(n)$ space.*

5 Conclusions and open question

We showed how to compute a smallest/largest repetition factorization of a given string w in $O(n \log n)$ time, where n is the length of w. The key idea is the reduction of the problems to the shortest/longest path problems on the repetition graph G, which is defined by the runs occurring in w. We first developed an algorithm which uses $O(n \log n)$ space, and showed that this space requirement is unavoidable if we explicitly construct G. We then showed how to simulate the first algorithm only with $O(n)$ space, by a dynamic programming approach.

Since there exists a string of length n for which the repetition graph G occupies $\Theta(n \log n)$ space, further speed-up seems difficult to achieve as long as we use the repetition graph G implicitly or explicitly. Thus, an intriguing open question is whether there exists an efficient algorithm which computes repetition factorizations of smallest/largest size without relying on the repetition graph G.

References

1. G. Badkobeh, H. Bannai, K. Goto, T. I, C. S. Iliopoulos, S. Inenaga, S. J. Puglisi, and S. Sugimoto: *Closed factorization*, in Proc. PSC 2014, 2014, pp. 162–168.
2. H. Bannai, T. Gagie, S. Inenaga, J. Kärkkäinen, D. Kempa, M. Piatkowski, S. J. Puglisi, and S. Sugimoto: *Diverse palindromic factorization is np-complete*, in Proc. DLT 2015, 2015, pp. 85–96.
3. K. T. Chen, R. H. Fox, and R. C. Lyndon: *Free differential calculus. iv. the quotient groups of the lower central series.* Annals of Mathematics, 68(1) 1958, pp. 81–95.

4. M. Crochemore and L. Ilie: *Computing longest previous factor in linear time and applications*. Inf. Process. Lett., 106(2) 2008, pp. 75–80.
5. M. Crochemore, L. Ilie, and W. F. Smyth: *A simple algorithm for computing the Lempel Ziv factorization*, in Proc. DCC 2008, 2008, pp. 482–488.
6. M. Crochemore and W. Rytter: *Sqares, cubes, and time-space efficient string searching*. Algorithmica, 13(5) 1995, pp. 405–425.
7. M. Dumitran, F. Manea, and D. Nowotka: *On prefix/suffix-square free words*, in Proc. SPIRE, 2015, pp. 54–66.
8. J. Duval: *Factorizing words over an ordered alphabet*. J. Algorithms, 4(4) 1983, pp. 363–381.
9. G. Fici, T. Gagie, J. Kärkkäinen, and D. Kempa: *A subquadratic algorithm for minimum palindromic factorization*. J. Discrete Algorithms, 28 2014, pp. 41–48.
10. N. J. Fine and H. S. Wilf: *Uniqueness theorems for periodic functions*. Proceedings of American Mathematical Society, 16(1) 1965, pp. 109–114.
11. A. S. Fraenkel and J. Simpson: *The exact number of squares in fibonacci words*. Theor. Comput. Sci., 218(1) 1999, pp. 95–106.
12. T. I, S. Sugimoto, S. Inenaga, H. Bannai, and M. Takeda: *Computing palindromic factorizations and palindromic covers on-line*, in Proc. CPM 2014, 2014, pp. 150–161.
13. R. Kolpakov, M. Podolskiy, M. Posypkin, and N. Khrapov: *Searching of gapped repeats and subrepetitions in a word*, in Proc. CPM 2014, 2014, pp. 212–221.
14. R. M. Kolpakov and G. Kucherov: *Finding maximal repetitions in a word in linear time*, in Proc. FOCS 1999, 1999, pp. 596–604.
15. M. Kufleitner: *On bijective variants of the Burrows-Wheeler transform*, in Proc. PSC 2009, 2009, pp. 65–79.
16. M. Lothaire: *Combinatorics on Words*, Addison-Wesley, 1983.
17. Y. Matsuoka, S. Inenaga, H. Bannai, M. Takeda, and F. Manea: *Factorizing a string into squares in linear time*, in 27th Annual Symposium on Combinatorial Pattern Matching, CPM 2016, June 27-29, 2016, Tel Aviv, Israel, 2016, pp. 27:1–27:12.
18. Y. Nakashima, T. I, S. Inenaga, H. Bannai, and M. Takeda: *Constructing LZ78 tries and position heaps in linear time for large alphabets*. Inf. Process. Lett., 115(9) 2015, pp. 655–659.
19. J. Storer and T. Szymanski: *Data compression via textual substitution*. J. ACM, 29(4) 1982, pp. 928–951.
20. Y. Tanimura, Y. Fujishige, T. I, S. Inenaga, H. Bannai, and M. Takeda: *A faster algorithm for computing maximal α-gapped repeats in a string*, in Proc. SPIRE 2015, 2015, pp. 124–136.
21. T. A. Welch: *A technique for high performance data compression*. IEEE Computer, 17 1984, pp. 8–19.
22. J. Ziv and A. Lempel: *A universal algorithm for sequential data compression*. IEEE Transactions on Information Theory, IT-23(3) 1977, pp. 337–349.
23. J. Ziv and A. Lempel: *Compression of individual sequences via variable-length coding*. IEEE Transactions on Information Theory, 24(5) 1978, pp. 530–536.