Fundamenta Informaticae XXI (2001) 1001–1013
DOI 10.3233/FI-2016-0000
IOS Press

Efficient Computation of Palindromes in Sequences with Uncertainties *

Mai Alzamel
Department of Informatics
King's College London
mai.alzamel@kcl.ac.uk

Jia Gao
Department of Informatics
King's College London
jia.gao@kcl.ac.uk

Costas S. Iliopoulos
Department of Informatics
King's College London
csi@kcl.ac.uk

Chang Liu
Department of Informatics
King's College London
chang.2.liu@kcl.ac.uk

Abstract. In this work, we consider a special type of uncertain sequence called weighted string. In a *weighted string* every position contains a subset of the alphabet and every letter of the alphabet is associated with a probability of occurrence such that the sum of probabilities at each position equals 1. Usually a *cumulative weight threshold* $1/z$ is specified, and one considers only strings

Address for correspondence: Address for correspondence goes here

*A preliminary version of this article has been accepted at the in Engineering Applications of Neural Networks: 18th International Conference, EANN 2017, Athens, Greece.

that match the weighted string with probability at least $1/z$. We provide an $\mathcal{O}(nz)$-time and $\mathcal{O}(nz)$-space off-line algorithm, where n is the length of the weighted string and $1/z$ is the given threshold, to compute a smallest maximal palindromic factorization of a weighted string. This factorization has applications in hairpin structure prediction in a set of closely-related DNA or RNA sequences. Along the way, we provide an $\mathcal{O}(nz)$-time and $\mathcal{O}(nz)$-space off-line algorithm to compute maximal palindromes in weighted strings.

Keywords: LaTeX 2_ε, document class file, BibTeX

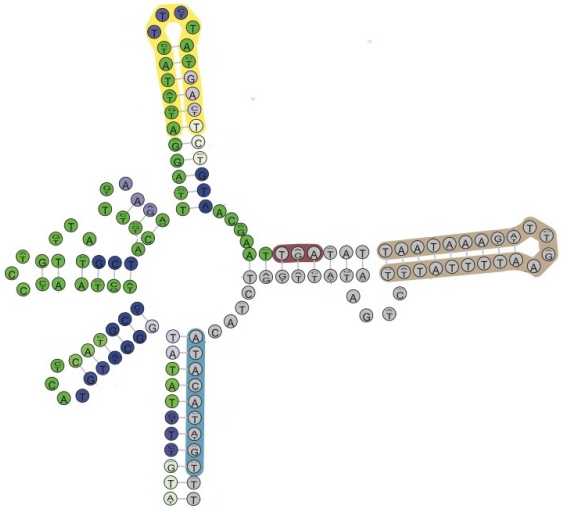
1. Introduction

A palindrome is a sequence that reads the same from left to right and from right to left. Detection of palindromic factors in texts is a classical and well-studied problem in algorithms on strings and combinatorics on words with a lot of variants arising out of different practical scenarios. In molecular biology, for instance, palindromic sequences are extensively studied: they are often distributed around promoters, introns, and untranslated regions, playing important roles in gene regulation and other cell processes (see e.g. [1]). In particular these are strings of the form $s\bar{s}^R$, also known as complemented palindromes, occurring in single-stranded DNA or, more commonly, in RNA, where s is a string and $\bar{s}^R$ is the reverse complement of s. In DNA, `C-G` are complements and `A-T` are complements; in RNA, `C-G` are complements and `A-U` are complements.

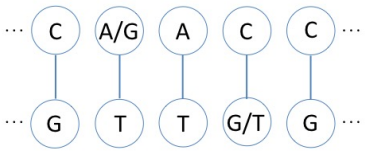
A string $x = x[0]x[1]\dots x[n-1]$ is said to have an initial palindrome of length k if its prefix of length k is a palindrome. Manacher first discovered an on-line algorithm that finds all initial palindromes in a string [2]. Later Apostolico et al observed that the algorithm given by Manacher is able to find all maximal palindromic factors in the string in $\mathcal{O}(n)$ time [3]. Gusfield gave an off-line linear-time algorithm to find all maximal palindromes in a string and also discussed the relation between biological sequences and gapped palindromes (i.e. strings of the form $sv\bar{s}^R$ where the complemented palindromes are separated by v) [4].

The problem that gained significant attention recently is the factorization of a string x of length n into a sequence of palindromes. We say that $x_1, x_2, \dots, x_\ell$ is a (maximal) palindromic factorization of string x, if every x_i is a (maximal) palindrome, $x = x_1x_2\dots x_\ell$, and ℓ is minimal. In biological applications we need to factorize a sequence into palindromes in order to identify *hairpins*, patterns that occur in single-stranded DNA or, more commonly, in RNA. Alatabbi et al gave an off-line $\mathcal{O}(n)$-time algorithm for finding a maximal palindromic factorization of x [5]. Fici et al presented an on-line $\mathcal{O}(n \log n)$-time algorithm for computing a palindromic factorization of x [6]; a similar algorithm was presented by I et al [7]. In addition, Rubinchik and Shur [8] devised an $\mathcal{O}(n)$-sized data structure that helps locating palindromes in x; they also showed how it can be used to compute a palindromic factorization of x in $\mathcal{O}(n \log n)$ time.

In this work, we consider a special type of uncertain sequence called weighted string (also known as position weight matrix or PWM). In a *weighted string* X every position contains a subset of the alphabet and every letter of the alphabet is associated with a probability of occurrence such that the sum of probabilities at each position equals 1. For example, we write $X =$ `a[(a,0.5),(b,0.5)]`$\dots$ to denote that the probability of occurrence of `a` at the first position is 1 while at the second one is



(a) Hairpins common to *Malvastrum yellow vein* virus, *Cotton leaf curl Multan* virus isolate, and *Bhendi yellow vein India* virus; figure taken from [9].



(b) Hairpin represented as a weighted string: `C[(A,0.5),(G,0.5)]ACC` (top) and `GTT[(G,0.5),(T,0.5)] G` (bottom).

Figure 1: Hairpins that are common to a set of closely-related sequences can be represented compactly as weighted strings.

$1/2$, and so on. X thus represents many different strings, each with probability of occurrence equal to the *product* of probabilities of its letters at subsequent positions of X. A great deal of research has been conducted on weighted strings for indexing [10, 11], for alignments [12, 13], for pattern matching [14, 15, 16], and for finding regularities [17, 18].

Our Problem. Muhire et al [9] showed how a set of virus species can be clustered using multiple sequence alignment (MSA) to obtain subsets of viruses that have common hairpin structure (see Fig. 1(a)). A more compact representation of an MSA can be trivially obtained using weighted strings (see Fig. 1(b)). The non-trivial computational problem thus arising is how to factorize a weighted string in a sequence of palindromes.

Our Contribution. Usually a *cumulative weight threshold* $1/z$ is specified, and one considers only strings that match the weighted string with probability at least $1/z$. In this paper, we generalize Alatabbi et al's solution for standard strings [5] to compute a maximal palindromic factorization of a weighted string. In particular, we provide an $\mathcal{O}(nz)$-time and $\mathcal{O}(nz)$-space off-line algorithm, where n is the length of the weighted string and $1/z$ is the given threshold. Along the way, we provide an $\mathcal{O}(nz)$-time and $\mathcal{O}(nz)$-space off-line algorithm for computing maximal palindromes in weighted strings.

2. Preliminaries

Let $x = x[0]x[1]\dots x[n-1]$ be a *string* of length $|x| = n$ over a finite ordered alphabet Σ of size $|\Sigma| = \sigma = \mathcal{O}(1)$. For two positions i and j on x, we denote by $x[i\,..\,j] = x[i]\dots x[j]$ the *factor*

(sometimes called *substring*) of x that starts at position i and ends at position j. We recall that a *prefix* of x is a factor that starts at position 0 ($x[0\,..\,j]$) and a *suffix* is a factor that ends at position $n-1$ ($x[i\,..\,n-1]$). We denote the *reversal* of x by string x^R, i.e. $x^R = x[n-1]x[n-2]\ldots x[0]$. The *empty string* (denoted by ε) is the unique string over Σ of length 0. The *concatenation* of two strings x and y is the string of the letters of x followed by the letters of y. It is denoted by $x.y$ or, more simply, by xy.

Let y be a string of length m with $0 < m \leq n$. We say that there exists an *occurrence* of y in x, or, more simply, that y *occurs in* x, when y is a factor of x. Every occurrence of y can be characterised by a starting position in x. Thus we say that y occurs at the *starting position* i in x when $y = x[i\,..\,i+m-1]$.

A string w is said to be a *palindrome* if and only if $w = w^R$. If factor $x[i\,..\,j]$, $0 \leq i \leq j \leq n-1$, of string $x[0\,..\,n-1]$ is a palindrome, then $\frac{i+j}{2}$ is the *center* of $x[i\,..\,j]$ in x and $\frac{j-i+1}{2}$ is the *radius* of $x[i\,..\,j]$. Moreover, $x[i\,..\,j]$ is called a *palindromic factor*. It is said to be a *maximal palindrome* if there is no other palindrome in x with center $\frac{i+j}{2}$ and larger radius. Hence x has exactly $2n-1$ maximal palindromes. A maximal palindrome w can be encoded as a pair (c, r), where c is the center of w and r is the radius of w. By $\mathcal{MP}(x)$, we denote the set of center-distinct maximal palindromes of string x. The sequence $x_1, x_2, \ldots, x_\ell$ of ℓ non-empty strings is a *(maximal) palindromic factorization* of a string x if all strings x_i are (maximal) palindromes, $x = x_1 x_2 \ldots x_\ell$, and ℓ is minimal.

Definition 2.1. A *weighted string* X on an alphabet Σ is a finite sequence of n sets. Every $X[i]$, for all $0 \leq i < n$, is a set of ordered pairs $(s_j, \pi_i(s_j))$, where $s_j \in \Sigma$ and $\pi_i(s_j)$ is the probability of having letter s_j at position i. Formally, $X[i]$=$\{(s_j, \pi_i(s_j)) \mid s_j \neq s_l \text{ for } j \neq l, \text{ and } \Sigma\pi_i(s_j) = 1\}$. A letter s_j *occurs* at position i of X if and only if the occurrence probability of letter s_j at position i, $\pi_i(s_j)$, is greater than 0.

Note that for clarity we use upper case letters for weighted strings, e.g. X, and lower case letters, e.g. x, for standard strings.

Definition 2.2. A string u of length m is a *factor* of a weighted string X if and only if it occurs at starting position i with *cumulative probability* $\prod_{j=0}^{m-1} \pi_{i+j}(u[j]) > 0$. Given a *cumulative weight threshold* $1/z \in (0, 1]$, we say factor u is *z-valid*, if it occurs at position i with cumulative probability $\prod_{j=0}^{m-1} \pi_{i+j}(u[j]) \geq 1/z$.

Example 2.3. Let $X =$ `ab[(a,0.5),(b,0.5)][(a,0.5),(b,0.5)]bab` and $1/z = 1/8$. String $u =$ `baaba` is a z-valid factor of X since u occurs at position 1 with cumulative probability $1/4 \geq 1/z = 1/8$.

Definition 2.4. Given a cumulative weight threshold $1/z \in (0, 1]$, a weighted string X of length m is a *z-palindrome* if and only if there exists at least one z-valid factor u of X of length m which is a palindrome.

Example 2.5. Let $X =$ `a[(a,0.5),(b,0.5)]bab[(a,0.4),(b,0.6)]a` of length $m = 7$ and $1/z = 1/8$. $u =$ `abbabba` is a z-valid factor of X of length 7 and u is a palindrome. Hence we say X is a z-palindrome.

If the weighted string $X[i\,..\,j]$ is a z-palindrome, we analogously define the number $\frac{i+j}{2}$ as the center of $X[i\,..\,j]$ in X and $\frac{j-i+1}{2}$ as the radius of $X[i\,..\,j]$.

Definition 2.6. Let X be a weighted string of length n, $1/z \in (0, 1]$ a cumulative weight threshold, and $X[i\,..\,j]$, where $0 \leq i \leq j \leq n-1$, a z-palindrome. Then $X[i\,..\,j]$ is a *maximal z-palindrome* if there is no other z-palindrome in X with center $\frac{i+j}{2}$ and larger radius.

A maximal z-palindrome can thus also be encoded as a pair (c, r). We study the following computational problem.

SMALLEST MAXIMAL z-PALINDROMIC FACTORIZATION
Input: A weighted string X of length n and a cumulative weight threshold $1/z \in (0, 1]$
Output: $X_1, X_2, \ldots, X_\ell$, if any, such that $X = X_1X_2 \ldots X_\ell$, X_i, for all $1 \leq i \leq \ell$, is a maximal z-palindrome, and ℓ is minimal.

We call this output sequence $X_1, X_2, \ldots, X_\ell$, i.e. when ℓ is minimal, a *smallest maximal z-palindromic factorization* of X.

The *suffix tree* $\mathcal{T}(x)$ of a non-empty string x of length n is a compact trie representing all suffixes of x. The nodes of the trie which become nodes of the suffix tree are called *explicit* nodes, while the other nodes are called *implicit*. Each edge of the suffix tree can be viewed as an upward maximal path of implicit nodes starting with an explicit node. Moreover, each node belongs to a unique path of that kind. Thus, each node of the trie can be represented in the suffix tree by the edge it belongs to and an index within the corresponding path. We let $\mathcal{L}(v)$ denote the *path-label* of a node v, i.e., the concatenation of the edge labels along the path from the root to v. We say that v is path-labelled $\mathcal{L}(v)$. Additionally, $\mathcal{D}(v) = |\mathcal{L}(v)|$ is used to denote the *string-depth* of node v. Node v is a *terminal* node if its path-label is a suffix of x, that is, $\mathcal{L}(v) = x[i\,..\,n-1]$ for some $0 \leq i < n$; here v is also labelled with index i. It should be clear that each factor of x is uniquely represented by either an explicit or an implicit node of $\mathcal{T}(x)$. The *suffix-link* of a node v with path-label $\mathcal{L}(v) = \alpha w$ is a pointer to the node with path-label w, where $\alpha \in \Sigma$ is a single letter and w is a string. The suffix-link of v is defined if v is an explicit node of $\mathcal{T}(x)$, different from the root. In standard suffix tree implementations, we assume that each node of the suffix tree is able to access its parent. Once $\mathcal{T}(x)$ is constructed, it can be traversed in a depth-first manner to compute $\mathcal{D}(v)$ for each node v. The suffix tree of a string of length n can be computed in time and space $\mathcal{O}(n)$ [19]. It can also be preprocessed in time and space $\mathcal{O}(n)$ so that *lowest common ancestor* (LCA) queries for any pair of explicit nodes can be answered in $\mathcal{O}(1)$ time per query [20].

Fact 2.7. ([4])
Given a string x, $\mathcal{MP}(x)$ can be computed in time $\mathcal{O}(|x|)$.

3. $\mathcal{O}(nz)$-time and $\mathcal{O}(nz)$-space algorithm

In this section, we present an algorithm to compute a smallest maximal z-palindromic factorization of a given weighted string X of length n for a given cumulative threshold $1/z \in (0, 1]$. Our algorithm

follows the one of Alatabbi et al for computing a smallest maximal palindromic factorization of standard strings [5] with some crucial modifications.

Why the algorithm of Alatabbi et al cannot be applied for weighted strings. Odd-length maximal palindromes centered at position i of a standard string x can be computed by finding the longest common prefix of suffixes $x[i\,..\,n-1]$ and $x^R[n-i-1\,..\,n-1]$. The longest common prefix of two suffixes can be found in $\mathcal{O}(1)$ time after $\mathcal{O}(n)$-time pre-processing of the suffix tree of $x\#x^R\$$, where $\#, \$ \notin \Sigma$, using LCA queries; using a similar computation, we can find all even-length maximal palindromes (see [4] for the details)

The length of the longest common z-valid prefix of any two suffixes of our weighted string X can be computed in time $\mathcal{O}(z)$ after $\mathcal{O}(nz)$-time pre-processing using the suffix-tree-based Weighted Index (WI) of [11] (inspect also Figure 2). However, this does not guarantee that the two corresponding common z-valid prefixes shall form a maximal z-palindrome: the two prefixes are z-valid by definition of the WI but their concatenation that forms a palindrome *may not* be z-valid because its occurrence probability drops below $1/z$.

We hence proceed as follows. By $\mathcal{MP}(X, z)$, we denote the set of center-distinct maximal z-palindromes of our weighted string X. Recall that we can represent a z-palindrome with center c and radius r by (c, r). For each position of X we define the *heaviest letter* as the letter with the maximum probability (breaking ties arbitrarily). We consider the string obtained from X by choosing at each position the heaviest letter. We call this the *heavy string* of X.

We define a collection $\mathcal{Z}_X$ of $\lfloor z \rfloor$ *special-weighted strings* of X, denoted by $\mathcal{Z}_k$, $0 \leq k < \lfloor z \rfloor$. Each $\mathcal{Z}_k$ is of length n and it has the following properties. Each position j in $\mathcal{Z}_k$ contains at most one letter with positive probability and it corresponds to position j in X. If f is a z-valid factor occurring at position j of X, then f occurs at position j in some of the $\mathcal{Z}_k$'s. The combinatorial observation telling us that this is possible is due to Barton et al [21]. For clarity of presentation we write $\mathcal{Z}_k$'s as standard strings.

Example 3.1. Given the weighted string

$$X = \texttt{[(a,0.5),(b,0.5)]bab[(a,0.5),(b,0.5)][(a,0.5),(b,0.5)]aaba}$$

and a cumulative weight threshold $1/z = 1/4$, we have:

$$\mathcal{Z}_X = \{\mathcal{Z}_0, \mathcal{Z}_1, \mathcal{Z}_2, \mathcal{Z}_3\} = \{\texttt{ababaaaaba}, \texttt{ababbaaaba}, \texttt{bbababaaba}, \texttt{bbabbbaaba}\}.$$

Lemma 3.2. ([21])
Given a weighted string X of length n and a cumulative weight threshold $1/z \in (0, 1]$, the $\lfloor z \rfloor$ special-weighted strings of X can be constructed in time and space $\mathcal{O}(nz)$.

Fact 3.3. Given a weighted string X of length n and a cumulative weight threshold $1/z \in (0, 1]$, we have that $\mathcal{MP}(X, z) \subseteq \mathcal{MP}(\mathcal{Z}_0, z) \cup \mathcal{MP}(\mathcal{Z}_1, z) \cup \ldots \cup \mathcal{MP}(\mathcal{Z}_{\lfloor z \rfloor - 1}, z)$.

Proof:
Suppose $U = X[i\,..\,j]$ is a maximal z-palindrome of center $c = \frac{i+j}{2}$ and radius $r = \frac{j-i+1}{2}$. By

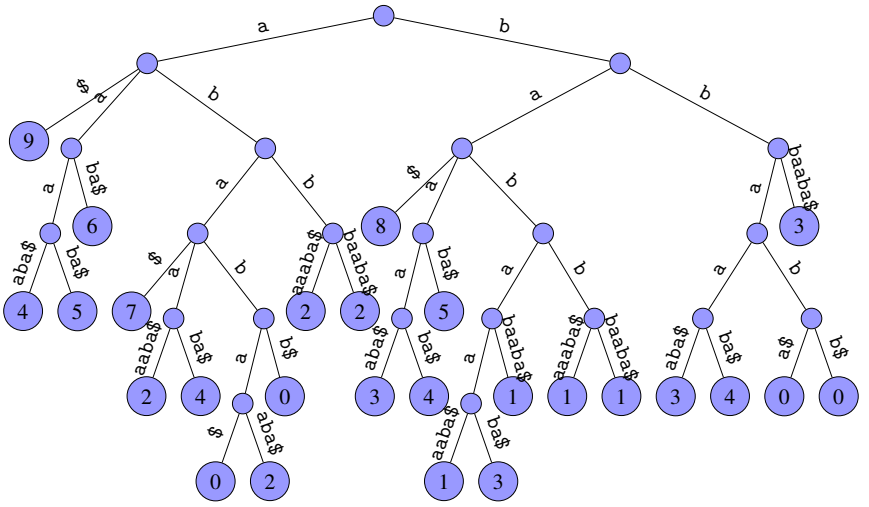


Figure 2: The WI for X and $1/z$ shown in Example 3.1 (labels of edges to terminal nodes are appended with a letter $\$ \notin \Sigma$ for convenience).

definition of U there must exist a z-valid palindromic factor u of U of radius r. Therefore, by definition of the special-weighted strings of X, u must be a z-valid factor of some $\mathcal{Z}_k$ and thus $(c, r) \in \mathcal{MP}mathcalZ_k, z)$. □

There are two steps for the correct computation of $\mathcal{MP}(X, z)$. First, we compute the set $\mathcal{A}_k$ of all maximal palindromes of the heavy string of $\mathcal{Z}_k$, for all $0 \leq k < \lfloor z \rfloor$, using Fact 2.7. We then need to adjust the radius of each reported palindrome for $\mathcal{Z}_k$ to ensure that it is z-valid in X (the center should not change). To achieve this, we compute an array $\mathcal{R}_k$, for each $\mathcal{Z}_k$, such that $\mathcal{R}_k[2c]$ stores the radius of the longest factor at center c in $\mathcal{Z}_k$ which is a z-valid factor of X at center c, e.g. $\mathcal{R}_k[2c] = \frac{j-i+1}{2}$, $c = (i + j)/2$, if $\mathcal{Z}_k[i\,..\,j]$ is a z-valid factor of X centered at c, and $\mathcal{Z}_k[i - 1\,..\,j + 1]$ is not a z-valid factor of X. By Fact 3.3, we cannot guarantee that all (c, r) in $\mathcal{MP}(\mathcal{Z}_k, z)$ are necessarily in $\mathcal{MP}(X, z)$. Hence, the second step is to compute $\mathcal{MP}(X, z)$ from $\mathcal{MP}(\mathcal{Z}_k, z)$ by taking the maximum radius per center and filtering out everything else.

Lemma 3.4. Given a weighted string X of length n, a cumulative weight threshold $1/z \in (0, 1]$, and the special-weighted strings $\mathcal{Z}_X$ of X, each $\mathcal{R}_k$, $0 \leq k < \lfloor z \rfloor$, can be computed in time $\mathcal{O}(n)$.

Proof:
By $< i, c, j >$, where $0 \leq i \leq c \leq j \leq n - 1$, we denote a factor of $\mathcal{Z}_k$ that has starting position i, ending position j and center $c = (i + j)/2$. We further denote the occurrence probability of $< i, c, j >$ in $\mathcal{Z}_k$ by $\Pi_{<i,c,j>} = \prod_{q=i}^{j} \pi_q(\mathcal{Z}_k[q])$. A factor $< i, c, j >$ of $\mathcal{Z}_k$ is called a special maximal z-valid factor of $\mathcal{Z}_k$ if $\Pi_{<i,c,j>} \geq 1/z$ and $\Pi_{<i-1,c,j+1>} < 1/z$.

For each $\mathcal{Z}_k$, we compute $\mathcal{R}_k$ from left to right. If we have $\Pi_{<0,0,0>} \geq 1/z$, we set $\mathcal{R}_k[0] = \frac{1}{2}$. If not, we go to the next position until we find a valid letter, say at position ℓ; then we have $\mathcal{R}_k[0] = \cdots =$

$\mathcal{R}_k[2\ell-1]=0$ and $\mathcal{R}_k[2\ell]=\frac{1}{2}$. Note that this corresponds to the first special maximal z-valid factor. Suppose we have a special maximal z-valid factor $< i, c, j >$ and $\mathcal{R}_k[2c]=\frac{j-i+1}{2}$, we show how to compute $\mathcal{R}_k[2c+1]$, which is the length of the special maximal z-valid factor at center $c'=\frac{2c+1}{2}$. We add the letter after $< i, c, j >$, so we have $< i, c', j+1 >$. We compute $\Pi_{<i,c',j+1>}$, which is simply $\Pi_{<i,c,j>} \times \pi_{j+1}(\mathcal{Z}_k[j+1])$. If $\Pi_{<i,c',j+1>} \geq 1/z$, the special maximal z-valid factor at center c' should be $< i, c', j+1 >$ and $\mathcal{R}_k[2c+1]=\mathcal{R}_k[2c]+\frac{1}{2}=\frac{j-i+2}{2}$. Factor $< i-1, c', j+2 >$ cannot be z-valid, since if $\Pi_{<i-1,c',j+2>} \geq 1/z$, we must have $\Pi_{<i-1,c,j+1>} \geq \Pi_{<i-1,c',j+2>} \geq 1/z$, which gives a longest special maximal z-valid at center c, namely $< i-1, c, j+1 >$, a contradiction. For $\Pi_{<i,c',j+1>} < 1/z$, the special maximal z-valid factor at center c' is $< i+1, c', j >$ since it always holds that $\Pi_{<i+1,c',j>} \geq \Pi_{<i,c,j>} \geq 1/z$. Therefore $\mathcal{R}_k[2c+1]=\mathcal{R}_k[2c]-\frac{1}{2}=\frac{j-i}{2}$.

Each center needs only to be considered once and there exist $2n-1$ distinct centers in each $\mathcal{Z}_k$. Therefore each $\mathcal{R}_k$ can be computed in $\mathcal{O}(n)$ time. □

Fact 3.5. (Trivial)
Let $x[i\,..\,j]$ be a palindrome of string x with center c and let u, $|u| < j-i+1$, be a factor of x with center c. Then u is also a palindrome.

After computing $\mathcal{A}_k$ and $\mathcal{R}_k$, we perform the following check for each palindrome $(c, r) \in \mathcal{A}_k$. If $r > \mathcal{R}_k[2c]$, the palindrome with radius r is not z-valid but the factor with radius $\mathcal{R}_k[2c]$ is z-valid and maximal (by definition) and palindromic (by Fact 3.5); if $r \leq \mathcal{R}_k[2c]$, the palindrome with radius r_i must be z-valid and it is maximal. Therefore we set $(c, r) \in \mathcal{MP}(\mathcal{Z}_k, z)$, such that $r=\min\{r, \mathcal{R}_k[2c]\},\ 0 \leq 2c \leq 2n-2,$ and $r \geq 1/2$.

To go from $\mathcal{MP}(\mathcal{Z}_k, z)$ to $\mathcal{MP}(X, z)$ we need to take the maximum radius for each center. Therefore for each center $c/2,\ 0 \leq c \leq 2n-2$, we set $(c/2, r) \in \mathcal{MP}(X, z)$, such that $r = \max\{r_k | (c/2, r_k) \in \mathcal{MP}(\mathcal{Z}_k, z), 0 \leq k < \lfloor z \rfloor\}$. We thus arrive at the first result of this article.

Theorem 3.6. Given a weighted string X of length n and a cumulative weight threshold $1/z \in (0, 1]$, all maximal z-palindromes in X can be computed in time and space $\mathcal{O}(nz)$.

After the computation of $\mathcal{MP}(X, z)$, we are in a position to apply the algorithm by Alatabbi et al [5] to find a smallest maximal z-palindromic factorization. We define a list $\mathcal{F}$ such that $\mathcal{F}[i], 0 \leq i \leq n-1$, stores the set of the lengths of all maximal z-palindromes ending at position i in X. We also define a list $\mathcal{U}$ such that $\mathcal{U}[i], 0 \leq i \leq n-1$, stores the set of positions j, such that $j+1$ is the starting position of a maximal z-palindrome in X and i is the ending position of this z-palindrome. Thus for a given $\mathcal{F}[i]=\{\ell_0, \ell_1, \ldots, \ell_q\}$, we have that $\mathcal{U}[i]=\{i-\ell_0, i-\ell_1, \ldots, i-\ell_q\}$. Note that $\mathcal{U}[i]$ can contain a "-1" element if there exists a maximal z-palindrome starting at position 0 and ending at position i. Note that the number of elements in $\mathcal{MP}(X, z)$ is at most $2n-1$, and, hence, $\mathcal{F}$ and $\mathcal{U}$ can contain at most $2n-1$ elements. The lists $\mathcal{F}$ and $\mathcal{U}$ can be computed trivially from $\mathcal{MP}(X, z)$.

Finally, we define a directed graph $\mathcal{G}_X=(\mathcal{V}, \mathcal{E})$, where $\mathcal{V}=\{i \mid -1 \leq i \leq n-1\}$ and $\mathcal{E}=\{(i, j) \mid j \in \mathcal{U}[i]\}$. Note that (i, j) is a directed edge from i to j. We do a breath first search on $\mathcal{G}_X$ assuming the vertex $n-1$ as the source and identify the shortest path from $n-1$ to -1, which gives a factorization.

We formally present the above as Algorithm SMPF for computing a smallest maximal z-palindromic factorization and obtain the following result.

Theorem 3.7. Given a weighted string X of length n and a cumulative weight threshold $1/z \in (0, 1]$, Algorithm SMPF correctly solves the problem SMALLEST MAXIMAL z-PALINDROMIC FACTORIZATION in time and space $\mathcal{O}(nz)$.

Proof:
The correctness follows from Theorem 3.6 for computing $\mathcal{MP}(X, z)$ and from the correctness of the algorithm in [5] for computing a smallest maximal palindromic factorization.

By Lemma 3.2, the construction of the special-weighted strings can be done in time and space $\mathcal{O}(nz)$. Computing $\mathcal{A}_k$ and $\mathcal{R}_k$, for all $0 \leq k < \lfloor z \rfloor$, can be done in total time $\mathcal{O}(nz)$ by Fact 2.7 and Lemma 3.4, respectively. From there on, computing $\mathcal{MP}(X, z)$ can be done in time $\mathcal{O}(nz)$. The lists $\mathcal{F}$ and $\mathcal{U}$ can be computed in time $\mathcal{O}(n)$ since the size of $\mathcal{MP}(X, z)$ is no more than $2n - 1$. There exist in total $n + 1$ vertices in $\mathcal{G}_X$. The number of edges $|\mathcal{E}|$ depends on $\mathcal{U}$, which contains no more than $2n$ elements; we have $|\mathcal{E}| = \mathcal{O}(n)$. Therefore, the construction of $\mathcal{G}_X$ and the breadth first search can be done in time $\mathcal{O}(|\mathcal{V}| + |\mathcal{E}|) = \mathcal{O}(n)$. The identification of the desired path can also be done easily if we do some simple bookkeeping during the breadth first search. The total running time of Algorithm SMPF is thus $\mathcal{O}(nz)$ and the space required is $\mathcal{O}(nz)$. □

Algorithm *SMPF*$(X, n, 1/z)$
  Construct the set $\mathcal{Z}_X$ of special-weighted strings of X;
  foreach $\mathcal{Z}_k \in \mathcal{Z}_X$ **do**
    $\mathcal{A}_k \leftarrow$ maximal palindromes of the heavy string of $\mathcal{Z}_k$;
    Compute $\mathcal{R}_k$ for $\mathcal{Z}_k$;
    $\mathcal{MP}(\mathcal{Z}_k, z) \leftarrow$ EMPTYLIST();
    foreach $(c, r) \in \mathcal{A}_k$ **do**
      $r \leftarrow \min\{r, \mathcal{R}_k[2c]\}$;
      if $r \geq \frac{1}{2}$ **then**
        Insert (c, r) in $\mathcal{MP}(\mathcal{Z}_k, z)$;
  $\mathcal{MP}(X, z) \leftarrow$ EMPTYLIST();
  foreach $c \in [0, 2n-2]$ **do**
    $r \leftarrow \max\{r_k | (c/2, r_k) \in \mathcal{MP}(\mathcal{Z}_k, z), 0 \leq k < \lfloor z \rfloor\}$;
    Insert $(c/2, r)$ in $\mathcal{MP}(X, z)$;
  $\mathcal{F} \leftarrow$ EMPTYLIST();
  $\mathcal{U} \leftarrow$ EMPTYLIST();
  foreach $(c, r) \in \mathcal{MP}(X, z)$ **do**
    $j \leftarrow \lfloor c + r \rfloor$;
    Insert $2r$ in $\mathcal{F}[j]$;
    Insert $j - 2r$ in $\mathcal{U}[j]$;
  Construct directed graph $\mathcal{G}_X = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V} = \{i \mid -1 \leq i \leq n-1\}$, $\mathcal{E} = \{(i, j) \mid j \in \mathcal{U}[i]\}$ and (i, j) is a directed edge from i to j;
  Breadth first search on $\mathcal{G}_X$ assuming the vertex $n-1$ as the source;
  Identify the shortest path $P \equiv \langle n-1 = p_\ell, p_{\ell-1}, \ldots, p_2, p_1, p_0 = -1 \rangle$;
  Return $X[0\,..\,p_1], X[p_1+1\,..\,p_2], \ldots, X[p_{\ell-1}+1\,..\,p_\ell]$;

4. Experiments

Algorithm SMALLEST MAXIMAL Z-PALINDROMIC FACTORIZATION was implemented as a program to compute the smallest maximal z-palindromic factorization in one or more input sequences.

Experiment 1. In the first experiment, our task was to establish the fact that the elapsed time and memory usage of the program grow linearly with z. As input datasets, for this experiment, we used synthetic DNA sequence of length 1MB. For this sequence we used different values of z. The results, for elapsed time and maximal memory usage, are plotted in Fig. 3. It becomes evident from the results that the elapsed time and memory usage of the program grow linearly with z.

Experiment 2. In the second experiment, our task was to establish the fact that the elapsed time and memory usage of the program grow linearly with n, the length of the input sequence. As input datasets, for this experiment, we used synthetic DNA sequences ranging from 250KB to 4000KB. For

each sequence we used constant values for $z = 8$. The results, for elapsed time and maximal memory usage, are plotted in Fig. 4. It becomes evident from the results that the elapsed time and memory usage of the program grow linearly with n.

References

[1] Almirantis Y, Charalampopoulos P, Gao J, Iliopoulos CS, Mohamed M, Pissis SP, Polychronopoulos D. On avoided words, absent words, and their application to biological sequence analysis. *Algorithms for Molecular Biology*, 2017. **12**(1):5. doi:10.1186/s13015-017-0094-z.

[2] Manacher G. A New Linear-Time "On-Line" Algorithm for Finding the Smallest Initial Palindrome of a String. *Journal of the ACM*, 1975. **22**(3):346–351.

[3] Apostolico A, Breslauer D, Galil Z. Parallel detection of all palindromes in a string. *Theoretical Computer Science*, 1995. **141**(1):163–173.

[4] Gusfield D. Algorithms on Strings, Trees, and Sequences: Computer Science and Computational Biology. Cambridge University Press, New York, NY, USA, 1997. ISBN 0-521-58519-8.

[5] Alatabbi A, Iliopoulos CS, Rahman MS. Maximal Palindromic Factorization. In: PSC. 2013 pp. 70–77.

[6] Fici G, Gagie T, Kärkkäinen J, Kempa D. A Subquadratic Algorithm for Minimum Palindromic Factorization. *Journal of Discrete Algorithms*, 2014. **28**:41–48. doi:10.1016/j.jda.2014.08.001. URL `http://dx.doi.org/10.1016/j.jda.2014.08.001`.

[7] I T, Sugimoto S, Inenaga S, Bannai H, Takeda M. Computing Palindromic Factorizations and Palindromic Covers On-line. In: CPM, volume 8486 of *LNCS*. Springer International Publishing, 2014 pp. 150–161.

[8] Rubinchik M, Shur AM. EERTREE: An Efficient Data Structure for Processing Palindromes in Strings. In: IWOCA, volume 9538 of *LNCS*, pp. 321–333. Springer International Publishing, 2016.

[9] Muhire BM, Golden M, Murrell B, Lefeuvre P, Lett JM, Gray A, Poon AY, Ngandu NK, Semegni Y, Tanov EP, et al. Evidence of pervasive biologically functional secondary structures within the genomes of eukaryotic single-stranded DNA viruses. *Journal of virology*, 2014. **88**(4):1972–1989.

[10] Iliopoulos CS, Makris C, Panagis Y, Perdikuri K, Theodoridis E, Tsakalidis A. The weighted suffix tree: an efficient data structure for handling molecular weighted sequences and its applications. *Fundamenta Informaticae*, 2006. **71**(2, 3):259–277.

[11] Barton C, Kociumaka T, Pissis SP, Radoszewski J. Efficient Index for Weighted Sequences. In: CPM, volume 54 of *LIPIcs*. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2016 pp. 4:1–4:13.

[12] Amir A, Gotthilf Z, Shalom BR. Weighted LCS. *Journal of Discrete Algorithms*, 2010. **8**(3):273–281.

[13] Cygan M, Kubica M, Radoszewski J, Rytter W, Walen T. Polynomial-time approximation algorithms for weighted LCS problem. *Discrete Applied Mathematics*, 2016. **204**:38–48.

[14] Kociumaka T, Pissis SP, Radoszewski J. Pattern Matching and Consensus Problems on Weighted Sequences and Profiles. In: ISAAC, volume 64 of *LIPIcs*. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2016 pp. 46:1–46:12.

[15] Barton C, Liu C, Pissis SP. Linear-time computation of prefix table for weighted strings & applications. *Theoretical Computer Science*, 2016. **656**:160–172.

[16] Barton C, Liu C, Pissis SP. On-Line Pattern Matching on Uncertain Sequences and Applications. In: COCOA, volume 10043 of *LNCS*, pp. 547–562. Springer International Publishing, 2016.

[17] Barton C, Iliopoulos CS, Pissis SP. Optimal Computation of all Tandem Repeats in a Weighted Sequence. *Algorithms for Molecular Biology*, 2014. **9**(21). doi:10.1186/s13015-014-0021-5.

[18] Barton C, Pissis SP. Crochemore's Partitioning on Weighted Strings and Applications. *Algorithmica*, 2017. doi:10.1007/s00453-016-0266-0.

[19] Farach M. Optimal suffix tree construction with large alphabets. In: FOCS. IEEE Computer Society, 1997 pp. 137–143.

[20] Bender MA, Farach-Colton M. The LCA Problem Revisited. In: LATIN, volume 1776 of *LNCS*. Springer-Verlag, 2000 pp. 88–94.

[21] Barton C, Kociumaka T, Liu C, Pissis SP, Radoszewski J. Indexing Weighted Sequences: Neat and Efficient. *CoRR*, 2017. **abs/1704.07625**.

[22] Breslauer D, Galil Z. Finding all periods and initial palindromes of a string in parallel. *Algorithmica*, 1995. **14**(4):355–366.

[23] Galil Z. Real-time algorithms for string-matching and palindrome recognition. In: Proceedings of the Eighth Annual ACM Symposium on Theory of Computing. ACM, 1976 pp. 161–173.

[24] Hsu PH, Chen KY, Chao KM. Finding all approximate gapped palindromes. In: International Symposium on Algorithms and Computation. Springer, 2009 pp. 1084–1093.

[25] Kolpakov R, Kucherov G. Searching for gapped palindromes. *Theoretical Computer Science*, 2009. **410**(51):5365–5373.

[26] Knuth DE, Morris JH Jr, Pratt VR. Fast pattern matching in strings. *SIAM journal on computing*, 1977. **6**(2):323–350.

[27] Droubay X. Palindromes in the Fibonacci word. *Information Processing Letters*, 1995. **55**(4):217–221.

[28] Droubay X, Pirillo G. Palindromes and Sturmian Words. *Theor. Comput. Sci.*, 1999. **223**(1-2):73–85. doi:10.1016/S0304-3975(97)00188-6. URL `http://dx.doi.org/10.1016/S0304-3975(97)00188-6`.

[29] Porto AH, Barbosa VC. Finding approximate palindromes in strings. *Pattern Recognition*, 2002. **35**(11):2581–2591.

[30] Matsubara W, Inenaga S, Ishino A, Shinohara A, Nakamura T, Hashimoto K. Efficient algorithms to compute compressed longest common substrings and compressed palindromes. *Theoretical Computer Science*, 2009. **410**(8):900–913.

[31] Gupta S, Prasad R, Yadav S. Searching Gapped Palindromes in DNA Sequences using Dynamic Suffix Array. *Indian Journal of Science and Technology*, 2015. **8**(23):1.

[32] Kolpakov R, Kucherov G. Searching for gapped palindromes. *Theor. Comput. Sci.*, 2009. **410**(51):5365–5373. doi:10.1016/j.tcs.2009.09.013. URL `http://dx.doi.org/10.1016/j.tcs.2009.09.013`.

[33] Crochemore M, Hancart C, Lecroq T. Algorithms on Strings. Cambridge University Press, 2007.

[34] Fujishige Y, Nakamura M, Inenaga S, Bannai H, Takeda M. Finding Gapped Palindromes Online. In: Mäkinen et al. [35], 2016 pp. 191–202. doi:10.1007/978-3-319-44543-4_15. URL `http://dx.doi.org/10.1007/978-3-319-44543-4_15`.

[35] Mäkinen V, Puglisi SJ, Salmela L (eds.). Combinatorial Algorithms - 27th International Workshop, IWOCA 2016, Helsinki, Finland, August 17-19, 2016, Proceedings, volume 9843 of *Lecture Notes in Computer Science*. Springer, 2016. ISBN 978-3-319-44542-7. doi:10.1007/978-3-319-44543-4. URL http://dx.doi.org/10.1007/978-3-319-44543-4.

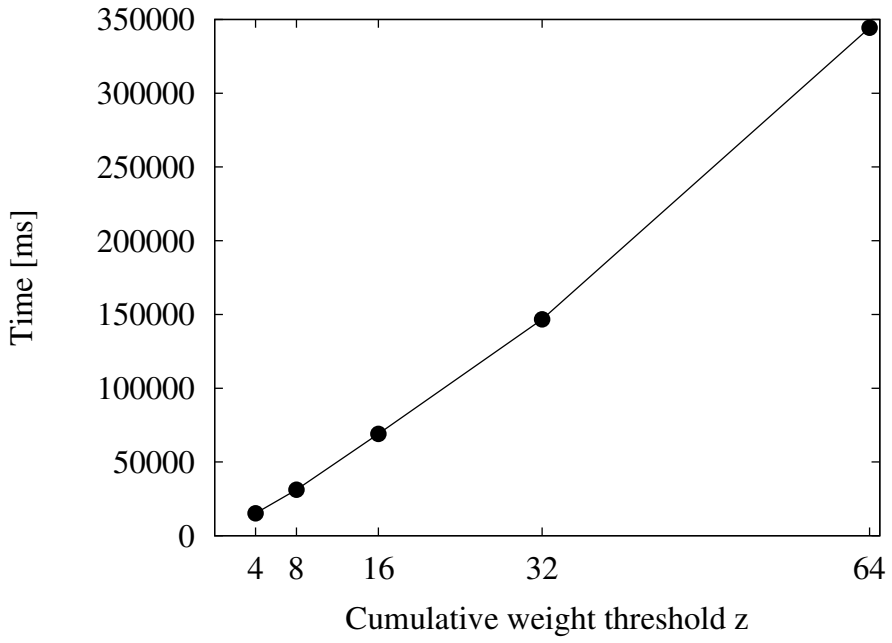
[36] Crochemore M, Rytter W. Jewels of stringology: text algorithms. World Scientific, 2003.

[37] Kosolobov D, Rubinchik M, Shur AM. Pal^k is Linear Recognizable Online. In: Proc. 41th Int. Conf. on Theory and Practice of Computer Science (SOFSEM 2015). 2015 pp. 289–301. doi: 10.1007/978-3-662-46078-8_24. URL http://dx.doi.org/10.1007/978-3-662-46078-8_24.

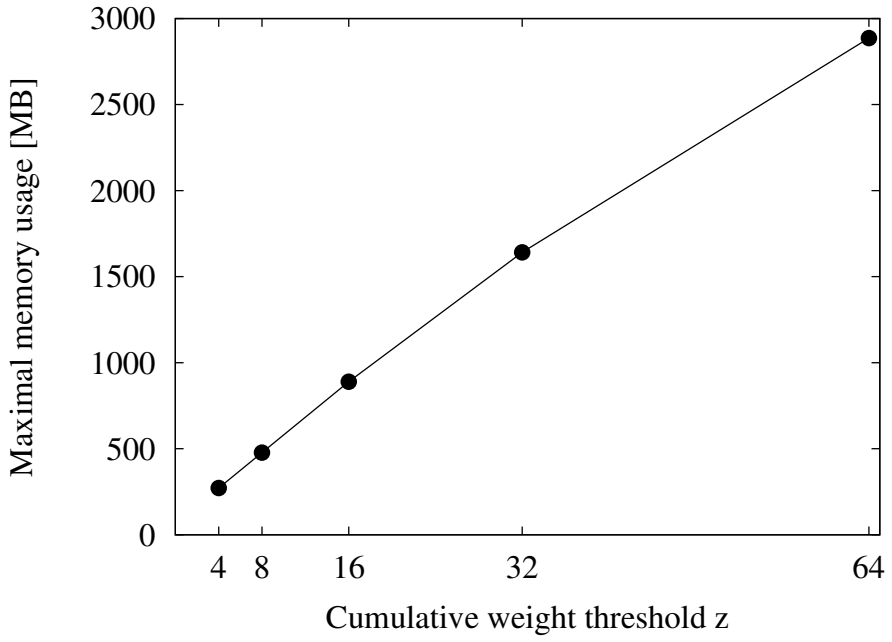
[38] Amir A, Chencinski E, Iliopoulos C, Kopelowitz T, Zhang H. Property matching and weighted matching. *Theoretical Computer Science*, 2008. **395**(2-3):298–310.

[39] Kolpakov R, Kucherov G. Searching for Gapped Palindromes. *Theor. Comput. Sci.*, 2009. **410**(51):5365–5373. doi:10.1016/j.tcs.2009.09.013. URL http://dx.doi.org/10.1016/j.tcs.2009.09.013.

[40] Amir A, Iliopoulos C, Kapah O, Porat E. Approximate matching in weighted sequences. In: CPM, volume 4009 of *LNCS*. Springer, 2006 pp. 365–376.

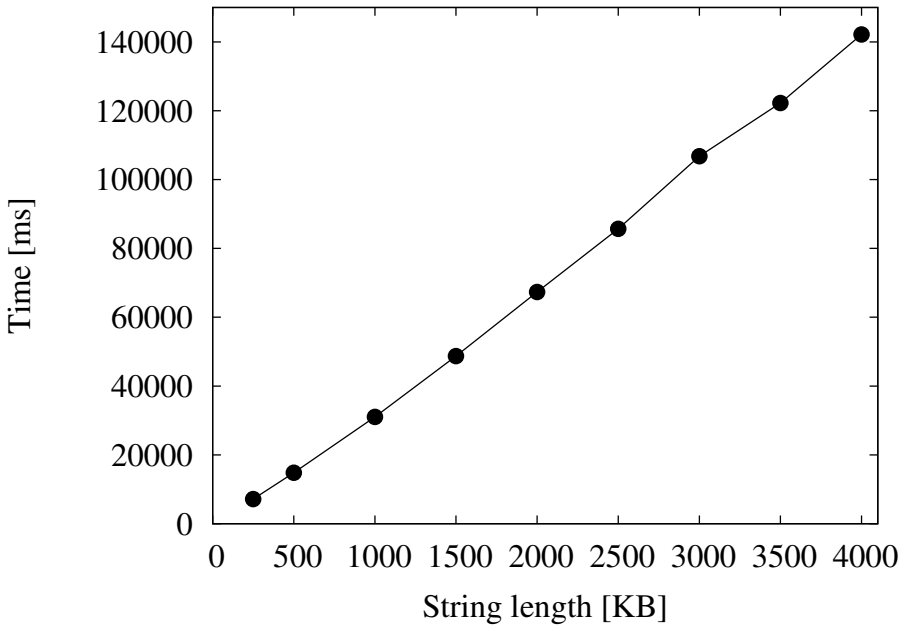


(a) Time for $n = 1$MB

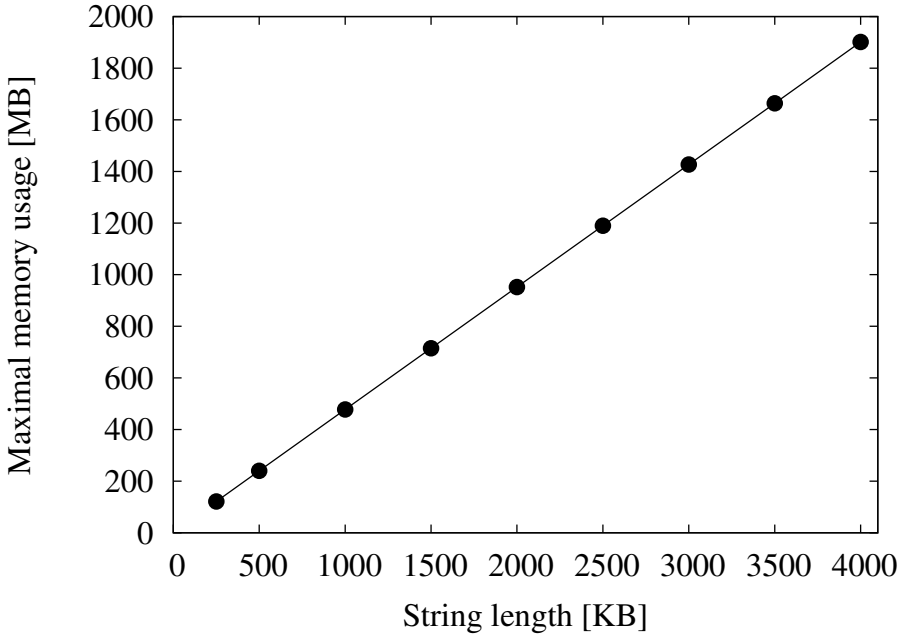


(b) Memory for $n = 1$MB

Figure 3: Experiment 1. Elapsed time and maximal memory usage of Algorithm Smallest Maximal z-Palindromic Factorization using synthetic DNA ($\sigma = 4$) data of length 1MB for variable z.



(a) Time for $z = 8$



(b) Memory for $z = 8$

Figure 4: Experiment 2. Elapsed time and maximal memory usage of Algorithm Smallest Maximal z-Palindromic Factorization using synthetic DNA ($\sigma = 4$) data of length 250KB to 4000KB.