DOI:
10.1007/978-3-319-78825-8_4

Document Version
Peer reviewed version

Recent Advances of Palindromic Factorization

Mai Alzamel[1⋆] and Costas S. Iliopoulos[1⋆⋆]

Department of Informatics, King's College London, UK
{mai.alzamel,costas.iliopoulos}@kcl.ac.uk

Abstract. This paper provides an overview of six particular problems of palindromic factorization and recent algorithmic improvements in solving them.

1 Introduction

1.1 General Definitions

Let $S = S[1]S[2]\cdots S[n]$ be a *string* of *length* $|S| = n$ over an alphabet Σ. We consider the case of an integer alphabet; in this case each letter can be replaced by its rank so that the resulting string consists of integers in the range $\{1, \ldots, n\}$. For two positions i and j, where $1 \leq i \leq j \leq n$, in S, we denote the *factor* $S[i]S[i+1]\cdots S[j]$ of S by $S[i\,..\,j]$. We denote the reverse string of S by S^R, i.e. $S^R = S[n]S[n-1]\cdots S[1]$. The empty string (denoted by ε) is the unique string over Σ of length 0. A string S is said to be a *palindrome* if and only if $S = S^R$. If $S[i\,..\,j]$ is a palindrome, the number $\frac{i+j}{2}$ is called the center of $S[i\,..\,j]$. Let $S[i\,..\,j]$, where $1 \leq i \leq j \leq n$, be a palindromic factor in S. It is said to be a *maximal palindrome* if there is no longer palindrome in S with center $\frac{i+j}{2}$. Note that a maximal palindrome can be a factor of another palindrome.

Definition 1. *A (maximal) palindromic decomposition of S such that the number of (maximal) palindromes is minimal is called a (maximal) palindromic factorization of S.*

Note that any single letter is a palindrome and, hence, every string can always be factorized into palindromes. However, not every string can be factorized into maximal palindromes; e.g. consider $S = \texttt{abaca}$ [2].

In this paper we present a survey of five novel algorithms of palindromic factorization. We start with maximal palindromic factorization presented by [2] in section 2. Later, we explain palindromic factorization with gaps, maximal palindromic factorization with errors and maximal palindromic factorization with gaps and errors presented by [1] in sections 3, 4 and 5.

Finally, we show in section 6 an efficient algorithm of palindromes in weighted strings presented by [4]

⋆ Fully supported by the Saudi Ministry of Higher Education and partially supported by the Onassis Foundation.

⋆⋆ Partially supported by the Onassis Foundation.

2 Maximal Palindromic Factorization

In this section we present an algorithm to compute the maximal palindromic factorization of a given string S presented by Alatabbi et al. [2]. They first present some notions required to present the algorithm. First of all, they use $\mathcal{MP}(S)$ to denote the set of center distinct maximal palindromes of S. They further extend this notation as follows. They use $\mathcal{MP}(S)[i]$, where $1 \leq i \leq n$ to denote the set of maximal palindromes with center i. Further, for the string S, they denote the set of all *prefix palindromes* (*suffix palindromes*) as $\mathcal{PP}(S)$ ($\mathcal{SP}(S)$).

Proposition 1. *The position i could be the center of at most two maximal palindromic factors, therefore; $\mathcal{MP}(S)[i]$ contains at most two elements, where $1 \leq i \leq n$, hence; there are at most $2n$ elements in $\mathcal{MP}(S)$.*

On the other hand, they use $\mathcal{MPL}(S)[i]$ to denote the set of the lengths of all maximal palindromes ending at position i,where $1 \leq i \leq n$ in S.

$$\mathcal{MPL}(s)[i] = \{2\ell - 1 \mid s[i-\ell+1 \ldots i+\ell-1] \in \mathcal{MP}(s)\} \cup \{2\ell' \mid s[i-\ell' \ldots i+\ell'-1] \in \mathcal{MP}(s)\} \quad (1)$$

where $1 \leq i \leq n$, with 2ℓ and $2\ell' + 1$ are the lengths of the odd and even palindromic factors respectively.

Proposition 2. *The set $\mathcal{MPL}(S)$ (Equation 1) can be computed in linear time from the set $\mathcal{MP}(s)$.*

They define the list $\mathcal{U}(S)$ such that for each $1 \leq i \leq n$,

$\mathcal{U}(S)[i]$ stores the position j such that $j+1$ is the starting position of a maximal palindromic factors ending at i and j is the end of another maximal palindromic substring.

Clearly, this can be easily computed once $\mathcal{MPL}(S)$ is computed.

$$\mathcal{U}[i][j] = i - \mathcal{MPL}(s)[i][j] \quad (2)$$

One can observe, from 1, that the sets $\mathcal{MPL}(S)$ and $\mathcal{U}(S)$ contain at most $2n$ elements. Given the list $\mathcal{U}(S)$ for a string S, they define a directed graph $\mathcal{G}_s = (\mathcal{V}, \mathcal{E})$ as follows. There are $\mathcal{V} = \{i \mid 1 \leq i \leq n\}$ and $\mathcal{E} = \{(i,j) \mid j \in \mathcal{U}(S)[i]\}$. Note that (i,j) is a directed edge where the direction is from i to j. The steps of the proposed algorithms are as follows.

MPF Algorithm: Maximal Palindromic Factorization Algorithm
Input: A String S of length n
Output: Maximal Palindromic Factorization of S

1. Compute the set of maximal palindromes $\mathcal{MP}(S)$ and identify the set of prefix palindromes $\mathcal{PP}(S)$.
2. Compute the list $\mathcal{MPL}(S)$.

3. Compute the list $\mathcal{U}(S)$.
4. Construct the graph $\mathcal{G}_s = (\mathcal{V}, \mathcal{E})$.
5. Do a breadth first search on $\mathcal{G}_s$ assuming the vertex n as the source.
6. Identify the shortest path $P \approx n \rightsquigarrow v$ such that v is the end position of a palindrome belonging to $\mathcal{PP}(S)$. Suppose $P \approx \langle n = p_k\ p_{k-1}\ \ldots\ p_2\ p_1 = v \rangle$.
7. Return $S = S[1..p_1]\ S[p_1 + 1..p_2]\ \ldots\ S[p_{k-1} + 1..p_k]$.

Theorem 1. *Given a string S of length n, (Maximal Palindromic Factorization (MPF)) Algorithm correctly computes the maximal palindromic factorization of S in $O(n)$ time.*

3 Palindromic Factorization with Gaps

In this section we present an an efficient solution to the Palindromic Factorization with Gaps problem has been introduced by Adamczyk et al. [1] .

It is based on several transformations of the algorithm for computing a palindromic factorization by Fici et al. [6]. For a string S of length n this algorithm works in $\mathcal{O}(n \log n)$ time. The algorithm consists of two steps:

1. Let P_j be the sorted list of starting positions of all palindromes ending at position j in S. This list may have size $\mathcal{O}(j)$. However, it follows from combinatorial properties of palindromes that the sequence of consecutive differences in P_j is non-increasing and contains at most $\mathcal{O}(\log j)$ distinct values. Let $P_{j,\Delta}$ be the maximal sublist of P_j containing elements whose predecessor in P_j is smaller by exactly Δ. Then there are $\mathcal{O}(\log j)$ such sublists in P_j. Hence, P_j can be represented by a set G_j of size $\mathcal{O}(\log j)$ which consists of triples of the form (i, Δ, k) that represent $P_{j,\Delta} = \{i, i+\Delta, \ldots, i+(k-1)\Delta\}$. The triples are sorted according to decreasing values of Δ and all starting positions in each triple are greater than in the previous one. Fici et al. show that G_j can be computed from G_{j-1} in $\mathcal{O}(\log j)$ time.
2. Let $PL[j]$ denote the number of palindromes in a palindromic factorization of $S[1\,..\,j]$. Fici et al. show that it can be computed via a dynamic programming approach, using all palindromes from G_j in $\mathcal{O}(\log j)$ time. Their algorithm works as follows. Let $PL_\Delta[j]$ be the minimum number of palindromes we can factorized $S[1\,..\,j]$ in, provided that we use a palindrome from $(i, \Delta, k) \in G_j$. Then $PL_\Delta[j]$ can be computed in constant time using $PL_\Delta[j - \Delta]$ based on the fact that if $(i, \Delta, k) \in G_j$ and $k \geq 2$, then $(i, \Delta, k-1) \in G_{j-\Delta}$. Exploiting this fact, $PL_\Delta[j]$ can be computed by only considering $PL_\Delta[j - \Delta]$ and the shortest palindrome in (i, Δ, k).
   Finally, $PL[j]$ can be computed from all such $PL_\Delta[j]$ values.

To solve the Palindromic Factorization with Gaps problem, Adamczyk et al. [1] algorithm firstly modify each of the triples in G_j to reflect the length constraint (m). More precisely, due to the length constraint, in each G_j some triples will disappear completely, and at most one triple will get *trimmed* (i.e. the parameter k will be decreased).

The algorithm then computes an array $MG[1\,..\,n][0\,..\,g]$ such that $MG[j][q]$ is the minimum possible total length of gaps in a palindromic factorization of $S[1\,..\,j]$, provided that there are at most q gaps. Simultaneously, Adamczyk et al. [1] algorithm computes an auxiliary array $MG'[1\,..\,n][0\,..\,g]$ such that $MG'[j][q]$ is the minimum possible total length of gaps up to position j provided that this position belongs to a gap: at most the q-th one.

the following formula for $j > 0$ and $q \geq 0$:

$$MG[j][q] = \min(MG'[j][q], \min_{\Delta}\{MG_{\Delta}[j][q]\})$$

where $MG_{\Delta}[j][q]$ is the partial minimum computed only using palindromes from $(i, \Delta, k) \in G_j$. The formula means: either there is a gap at position j, or using a palindrome ending at position j. Also $MG[0][q]$ is filled by zeros for any $q \geq 0$.

Adamczyk et al. [1] algorithm computes $MG_{\Delta}[j][q]$ for $(i, \Delta, k) \in G_j$ using the same approach as Fici et al. [6] used for PL_{Δ}, ignoring the triples that disappear due to the length constraint. If there is a triple that got trimmed, then the corresponding triple at position $j - \Delta$ (from which they reuse the values in the dynamic programming) must have got trimmed as well. More precisely, if the triple (i, Δ, k) is trimmed to (i, Δ, k') at position j, then at position $j - \Delta$ there is a triple $(i, \Delta, k-1)$ which is trimmed to $(i, \Delta, k'-1)$; that is, by the same number of palindromes. Consequently, to compute $MG_{\Delta}[j][q]$ from $MG_{\Delta}[j - \Delta][q]$, they need to include one additional palindrome (the shortest one in the triple) just as in Fici et al.'s approach.

Finally, for $j > 0$ and $q > 0$ they compute MG' using the following formula:

$$MG'[j][q] = \min(MG'[j-1][q], MG[j-1][q-1]) + 1.$$

The first case corresponds to continuing the gap from position j, whereas the second to using a palindrome finishing at position $j - 1$ or a gap finishing at position $j - 1$ (the latter will be suboptimal). Here the border cases are $MG'[j][0] = \infty$ for $j \geq 0$ and $MG'[0][q] = \infty$ for $q > 0$.

Thus Adamczyk et al. [1] arrive at the complete solution to the problem.

Theorem 2. *The* Palindromic Factorization with Gaps *problem can be solved in* $\mathcal{O}(n \log n \cdot g)$ *time and* $\mathcal{O}(n \cdot g)$ *space.*

4 Computing Maximal Palindromes with Errors

We show an algorithm presented by Adamczyk et al. [1] to compute maximal δ-palindromes under the edit distance within $O(n \cdot \delta)$. If u is a δ-palindrome under the edit distance, then there exists a palindrome v such that the minimal number of edit operations (insertion, deletion, substitution) required to transform u to v is at most δ. The following simple observation shows that it can restrict edit operations to deletions and substitutions only, which Adamczyk et al. [1] call in what follows the *restricted edit operations*. Intuitively, instead of inserting at position i a character to match the character at position $|u| - i + 1$, the character can be deleted at position $|u| - i + 1$.

Observation 3 *Let u be a δ-palindrome and v a palindrome such that the edit distance between u and v is minimal. Then there exists a palindrome v' such that the number of restricted edit operations needed to transform u to v' is equal to the edit distance between u and v.*

Definition 2. *A (LGPal-queries) is a maximal palindromes are computed using Gusfield's approach [7]*

Adamczyk et al. [1] extend a maximal δ-palindrome $S[i\,..\,j]$ to a maximal $(\delta+1)$-palindrome in three ways; either ignore the letter $S[i-1]$ and then perform an LGPal-query, or ignore the letter $S[j+1]$ and then perform an LGPal-query, or ignore both and then perform the LGPal-query. More formally:

Definition 3. *Assume that $S[i\,..\,j]$ is a δ-palindrome. Then it says that each of the factors $S[i'\,..\,j']$ for:*

- *$i' = i-1-d$, $j' = j+d$, where $d = LGPal(i-2, j+1)$*
- *$i' = i-d$, $j' = j+1+d$, where $d = LGPal(i-1, j+2)$*
- *$i' = i-1-d$, $j' = j+1+d$, where $d = LGPal(i-2, j+2)$*

is an extension *of $S[i\,..\,j]$. If the index i' is smaller than 1 or the index j' is greater than $|S|$, the corresponding extension is not possible. They also say that $S[i\,..\,j]$ can be extended to any of the three strings $S[i'\,..\,j']$.*

Clearly, the extensions of a δ-palindrome are always $(\delta+1)$-palindromes.

To facilitate the case of δ-palindromes being prefixes or suffixes of the text, they also introduce the following *border-reductions* for $S[i\,..\,j]$ being a δ-palindrome:

- If $i = 1$, a border reduction leads to $S[1\,..\,j-1]$.
- If $j = n$, a border reduction leads to $S[i+1\,..\,n]$.

If any of the reductions is possible, they also say that $S[i\,..\,j]$ can be border-reduced to the corresponding strings. As previously, border-reductions of a δ-palindrome are always $(\delta+1)$-palindromes.

Lemma 1. *Given a maximal δ-palindrome $S[i'\,..\,j']$ with $\delta > 0$, there exists a maximal $(\delta-1)$-palindrome $S[i\,..\,j]$ which can be extended or border-reduced to $S[i'\,..\,j']$.*

The combinatorial characterization of Lemma 1 yields an algorithm for generating all maximal d-palindromes, for all centers and subsequent $d = 0, \ldots, \delta$.

Recall maximal 0-palindromes are computed using Gusfield's approach (LGPal-queries). For a given $d < \delta$, they consider all the maximal d-palindromes and try to extend each of them in all three possible ways (and border-reduce, if possible). This way they obtain a number of $(d+1)$-palindromes amongst which, by Lemma 1, are all maximal $(d+1)$-palindromes. To exclude the non-maximal ones, they group the $(d+1)$-palindromes by their centers (in $\mathcal{O}(n)$ time via bucket sort) and retain only the longest one for each center.

They arrive at the following intermediate result.

Lemma 2. *Under the edit distance, all maximal δ-palindromes in a string of length n can be computed in $\mathcal{O}(n \cdot \delta)$ time and $\mathcal{O}(n)$ space.*

5 Maximal Palindromic Factorization with Gaps and Errors

We show an algorithm presented by Adamczyk et al. [1] to solve maximal palindromic factorization with gaps and errors problem in $\mathcal{O}(n \cdot (g + \delta))$ time and $\mathcal{O}(n \cdot g)$ space.

Let $\mathcal{F}$ be a set of factors of the text $S[1\,..\,n]$. In this section they develop a general framework that allows to factorized S into factors from $\mathcal{F}$, allowing at most g gaps. They call such a factorization a $(g, \mathcal{F})$-factorization of S.

The goal is to find a $(g, \mathcal{F})$-factorization of S that minimizes the total length of gaps. The authors aim at the time complexity $\mathcal{O}((n + |\mathcal{F}|) \cdot g)$ and space complexity $\mathcal{O}(n \cdot g + |\mathcal{F}|)$.

In the proposed solution Adamczyk et al. [1] use dynamic programming to compute two arrays, similar to the ones used in Section 3:

$MG[1\,..\,n][0\,..\,g]$: $MG[j][q]$ is the minimum total length of gaps in a $(q, \mathcal{F})$-factorization of $S[1\,..\,j]$.

$MG'[1\,..\,n][0\,..\,g]$: $MG'[j][q]$ is the minimum total length of gaps in a $(q, \mathcal{F})$-factorization of $S[1\,..\,j]$ for which the position j belongs to a gap.

They use the following formulas, for $j > 0$ and $q > 0$:

$$MG[j][q] = \min(MG'[j][q], \min_{S[a\,..\,j] \in \mathcal{F}} MG[a-1][q])$$

$$MG'[j][q] = \min(MG[j-1][q-1], MG'[j-1][q]) + 1$$

The border cases are exactly the same as in Section 3.

They apply this approach to maximal δ-palindromes in each of the considered metrics (see the classic result from [7] for the Hamming distance and Lemma 2 for the edit distance) to obtain the following result.

Theorem 4. *The* Maximal δ-Palindromic Factorization with Gaps *problem under the Hamming distance or the edit distance can be solved in $\mathcal{O}(n \cdot (g + \delta))$ time and $\mathcal{O}(n \cdot g)$ space.*

6 Maximal Palindromic Factorization of Weighted String

In this section, we show an algorithm to compute a smallest maximal z-palindromic factorization of a given weighted string X of length n for a given cumulative threshold $1/z \in (0, 1]$ has been presented by alzamel et at[4]. Our algorithm follows the one of Alatabbi et al for computing a smallest maximal palindromic factorization of standard strings [3] with some crucial modifications. Recall by $\mathcal{MP}(x)$, we denote the set of center-distinct maximal palindromes of string x We will use the below two facts related to palindromes:

Fact 5 ([7]) *Given a string x, $\mathcal{MP}(x)$ can be computed in time $\mathcal{O}(|x|)$.*

Fact 6 (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.*

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.

we start with some definitions related to weighted strings :

Definition 4. *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, $X[i]$=$\{(s_j, \pi_i(s_j)) \mid s_j \neq s_l$ for $j \neq l$, 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.

Definition 5. *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 $\prod_{j=0}^{m-1} \pi_{i+j}(u[j]) \geq 1/z$.*

Definition 6. *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.*

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 7. *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.*

We proceed as follows: By $\mathcal{MP}(X, z)$, we denote the set of center-distinct maximal z-palindromes of our weighted string X. We present 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 [5]. For clarity of presentation we write $\mathcal{Z}_k$'s as standard strings.

Lemma 3 ([5]). *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 7 *Given a weighted string X of length n and a cumulative weight threshold $1/z \in (0,1]$, we have that $\mathcal{MP}(X,z) \subseteq \mathcal{MP}(\mathcal{Z}_0,z) \cup \mathcal{MP}(\mathcal{Z}_1,z) \cup \ldots \cup \mathcal{MP}(\mathcal{Z}_{\lfloor z \rfloor -1},z)$.*

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 5. 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 7, 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 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 $\mathcal{O}(n)$.*

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 6); 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 8. *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 [3] to find the 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 $2n-2$ 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 the factorization. We formally present the above as Algorithm SMPF for computing a smallest maximal z-palindromic factorization and obtain the following result.

Theorem 9. *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 $\mathcal{O}(nz)$.*

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 $r \geq \frac{1}{2}$ 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 $P \approx \langle n-1 = p_\ell, p_{\ell-1}, \ldots, p_2, p_1, p_0 = -1 \rangle$;
  Return $X[0\,..\,p_1], X[p_1+1\,..\,p_2], \ldots, X[p_{\ell-1}+1\,..\,p_\ell]$;

7 Conclusion

In this paper we present a review of recent advances of palindromic factorization.

References

1. M. Adamczyk, M. Alzamel, P. Charalampopoulos, C. S. Iliopoulos, and J. Radoszewski. Palindromic decompositions with gaps and errors. *arXiv preprint arXiv:1703.08931*, 2017.
2. A. Alatabbi, C. S. Iliopoulos, and M. S. Rahman. Maximal palindromic factorization. In *Stringology*, pages 70–77, 2013.
3. A. Alatabbi, C. S. Iliopoulos, and M. S. Rahman. Maximal palindromic factorization. In *Proceedings of the Prague Stringology Conference 2013*, pages 70–77, Czech Technical University in Prague, Czech Republic, 2013.
4. M. Alzamel, J. Gao, C. S. Iliopoulos, C. Liu, and S. P. Pissis. Efficient computation of palindromes in sequences with uncertainties. *accepted at Mining Humanistic Data Workshop*, 2017.
5. C. Barton, T. Kociumaka, C. Liu, S. P. Pissis, and J. Radoszewski. Indexing Weighted Sequences: Neat and Efficient. *CoRR*, abs/1704.07625, 2017.
6. G. Fici, T. Gagie, J. Kärkkäinen, and D. Kempa. A subquadratic algorithm for minimum palindromic factorization. *J. Discr. Alg.*, 28(C):41–48, Sept. 2014.
7. D. Gusfield. *Algorithms on Strings, Trees, and Sequences: Computer Science and Computational Biology.* Cambridge University Press, New York, NY, USA, 1997.