Maximal Palindromic Factorization

Ali Alatabbi[1,⋆], Costas S. Iliopoulos[2], and M. Sohel Rahman[1,2,⋆⋆]

[1] Department of Informatics, King's College London, London WC2R 2LS, United Kingdom
[2] AℓEDA Group, Department of CSE, BUET, Dhaka-1000, Bangladesh

Abstract. A palindrome is a symmetric string, phrase, number, or other sequence of units sequence that reads the same forward and backward.
We present an algorithm for maximal palindromic factorization of a finite string by adapting an Gusfield algorithm [15] for detecting all occurrences of maximal palindromes in a string in linear time to the length of the given string then using the breadth first search (BFS) to find the maximal palindromic factorization set.
A factorization $\mathcal{F}$ of s with respect to $\mathcal{S}$ refers to a decomposition of s such that $s = s_{i_1} s_{i_2} \cdots s_{i_\ell}$ where $s_{i_j} \in \mathcal{S}$ and ℓ is minimum. In this context the set $\mathcal{S}$ is referred to as the factorization set. In this paper, we tackle the following problem. Given a string s, find the maximal palindromic factorization of s, that is a factorization of s where the factorization set is the set of all center-distinct maximal palindromes of a string s $\mathcal{MP}(s)$.

Keywords: palindromes, factorization, graph search

1 Introduction

A palindrome is a symmetric word that reads the same backward and forward. The detection of palindromes is a classical and well-studied problem in computer science, language theory and algorithm design with a lot of variants arising out of different practical scenarios. String and sequence algorithms related to palindromes have long drawn attention of stringology researchers [1,12,17,22,25,26,27,29]. Interestingly, in the seminal Knuth-Morris-Pratt paper presenting the well-known string matching algorithm [19], a problem related to palindrome recognition was also considered. In word combinatorics, for example, studies have investigated the inhabitation of palindromes in Fibonacci words or Sturmian words in general [10], [11], [14].

Manacher discovered an on-line sequential algorithm that finds all *initial* palindromes in a string [25]. A string $X[1 \ldots n]$ is said to have an initial palindrome of length k if the prefix $S[1 \ldots k]$ is a palindrome. Gusfield gave a linear-time algorithm to find all *maximal* palindromes (a notion we define shortly) in a string [16]. Porto and Barbosa gave an algorithm to find all *approximate* palindromes in a string [29]. Matsubara et al. solved in [27] the problem of finding all palindromes in SLP (Straight Line Programs)-compressed strings. Additionally, a number of problems on variants of palindromes have also been investigated in the literature [17,4,22]. Very recently, I *et al.* [18] worked on pattern matching problems and Chowdhury et al. [6] studied the longest common subsequence problem involving palindromes.

In this paper, we present a linear-time algorithm for computing the *maximal palindromic factorization (MPF)* of a string, that is the smallest set (minimum number of palindromic factors), such that the string is covered by that set of factors with no

⋆ Partially supported by a Commonwealth Fellowship.
⋆⋆ ⋆, Ali Alatabbi.

Ali Alatabbi, Costas S. Iliopoulos, M. Sohel Rahman: Maximal Palindromic Factorization, pp. 70–77.
Proceedings of PSC 2013, Jan Holub and Jan Žďárek (Eds.), ISBN 978-80-01-05330-0 © Czech Technical University in Prague, Czech Republic

overlaps. This problem was very recently posed as an open problem in Stringology at [30].

Generic factorization process plays an important role in String Algorithms. The obvious advantage of such process is that when processing a string online, the work done on an element of the factorization can usually be skipped because already done on its previous occurrence [8]. A typical application of this concept resides in algorithms to compute repetitions in strings, such as Kolpakov and Kucherov algorithm for reporting all maximal repetitions [21], Lyndon factorization [28], have been applied in: string matching [9,2], the Burrows-Wheeler Transform [3] and LempelZiv factorization [32] have been applied in: data compression [7,13] and indeed it seems to be the only technique that leads to linear-time algorithms independently of the alphabet size [8]. Words with palindromic structure are important in DNA and RNA sequences, Biologists believe that palindromes play an important role in regulation of gene activity and other cell processes because these are often observed near promoters, introns and specific untranslated regions. Palindromic structure in DNA and RNA sequences reflects the capacity of molecules to fold [20], i.e. to form double-stranded stems, which insures a stable state of those molecules with low free energy. Identifying palindromes could help in advancing the understanding of genomic instability [5], [24], [31]. Finding common palindromes in two gene sequences can be an important criterion to compare them, and also to find common relationships between them. However, in those applications, the reversal of palindromes should be combined with the complementarity concept on nucleotides, where c is complementary to g and a is complementary to t (or to u, in case of RNA). Moreover, gapped palindromes are biologically meaningful, i.e. contain a spacer between left and right copies (see [20]). Therefore, detecting palindromes in DNA sequences is one of the challenging problems in computational biology. Researchers have also shown that based on palindrome frequency, DNA sequences can be discriminated to the level of species of origin [23]. So, finding common palindromes in two DNA sequences can be an important criterion to compare them, and also to find common relationships between them.
The rest of the paper is organized as follows. In Section 2 we give some definitions and introduce the notations used in the rest of the paper. In Section 3, we describe our algorithm for computing the maximal palindromic factorization of a given string. Finally, We will prove correctness of the algorithm and analyze its running time in Section 4 and we briefly conclude in Section 5 with some future proposals.

2 Notation and terminology

A string or sequence is a succession of zero or more symbols from an alphabet Σ of cardinality σ, where σ expresses the number of distinct characters in the alphabet. The empty string is the empty sequence (of zero length) and is denoted by ϵ. The set of all strings over the alphabet Σ including ϵ is denoted by Σ^*. The set of all non-empty strings over the alphabet Σ is denoted by Σ^+. $\Sigma^* = \Sigma^+ \cup \epsilon$. A string s of length $|s| = n$ is represented by $s[1 \dots n]$. The i-th symbol of s is denoted by $s[i]$. A string y is a factor of s if $s = xyz$ for $x, z \in \Sigma^*$; it is a prefix of s if x is empty and a suffix of s if z is empty. We denote by $s[i \dots j]$ the factor of s that starts at position i and ends at position j. We denote by $\tilde{s}$ the reversal of s, i.e., $\tilde{s} = s[n]\ s[n-1]\ \cdots\ s[1]$.

A palindrome is a symmetric string that reads the same forward and backward. More formally, s is called a palindrome if and only if $s = \tilde{s}$. The empty string ϵ is assumed to be a palindrome. Also note that a single character is a palindrome

by definition. The following is another (equivalent) definition of a palindrome which indicates that palindrome can be of both odd and even length. A string s is a palindrome if $s = xa\tilde{x}$ where x is a string and a is either a single character or the empty string ϵ. Clearly, if a is a single character, then s is a palindrome having odd length; otherwise, it is of even length.

The radius of a palindrome s is $\frac{|s|}{2}$. In the context of a string, if we have a substring that is a palindrome, we often call it a palindromic substring. Given a string s of length n, suppose $s[i \dots j]$,with $1 \leq i \leq j \leq n$ is a palindrome, i.e., $s[i \dots j]$ is a palindromic substring of s. Then, the center of the palindromic substring $s[i \dots j]$ is $\lfloor \frac{i+j}{2} \rfloor$. A palindromic substring $s[i \dots j]$ is called the maximal palindrome at the center $\lfloor \frac{i+j}{2} \rfloor$ if no other palindromes at the center $\lfloor \frac{i+j}{2} \rfloor$ have a larger radius than $s[i \dots j]$, i.e., if $s[i-1] \neq s[j+1]$, where $i = 1$, or $j = n$. A maximal palindrome $s[i \dots j]$ is called a suffix (prefix) palindrome of s if and only if $j = n$ ($i = 1$). We denote by $(c, r)_s$ the maximal maximal palindromic factor of a string s whose center is c and radius is r; we usually drop the subscript and use (c, r) when the string s is clear from the context. The set of all center-distinct maximal palindromes of a string s is denoted by $\mathcal{MP}(s)$. Further, for the string s, we denote the set of all *prefix palindromes* (*suffix palindromes*) as $\mathcal{PP}(s)$ ($\mathcal{SP}(s)$). We use the following result from [25,16].

Theorem 1 ([25,16]). *For any string s of length n, $\mathcal{MP}(s)$ can be computed in $O(n)$ time.*

In what follows, we assume that the elements of $\mathcal{MP}(s)$ are sorted in increasing order of centers c. Actually, the algorithm of [25] computes the elements of $\mathcal{MP}(s)$ in this order. Clearly, the set $\mathcal{PP}(s)$ and $\mathcal{SP}(s)$ can be computed easily during the computation of $\mathcal{MP}(s)$.

Suppose, we are given a set of strings $\mathcal{S} = \{s_1, s_2, \dots, s_k\}$, such that s_i is a substring of s and $1 \leq i \leq k$. A factorization $\mathcal{F}$ of s with respect to $\mathcal{S}$ refers to a decomposition of s such that $s = s_{i_1} s_{i_2} \cdots s_{i_\ell}$ where $s_{i_j} \in \mathcal{S}$ and ℓ is minimum. In this context the set $\mathcal{S}$ is referred to as the factorization set. In this paper, we tackle the following problem.

Problem 2. (Maximal Palindromic Factorization (MPF)) Given a string s, find the maximal palindromic factorization of s, that is a factorization of s where the factorization set is $\mathcal{MP}(s)$.

3 The Algorithm

In this section we present an algorithm to compute the maximal palindromic factorization of a given string s. We first present some notions required to present our algorithm. First of all, recall that we use $\mathcal{MP}(s)$ to denote the set of center distinct maximal palindromes of s. We further extend this notation as follows. We use $\mathcal{MP}(s)[i]$, where $1 \leq i \leq n$ to denote the set of maximal palindromes with center i.

Proposition 3. *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, we 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 4. *The set $\mathcal{MPL}(s)$ (Equation 1) can be computed in linear time from the set $\mathcal{MP}(s)$.*

Now we 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 we have $\mathcal{MPL}(s)$ computed.

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

One can observe, from 3, 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, we define a directed graph $\mathcal{G}_s = (\mathcal{V}, \mathcal{E})$ as follows. We have $\mathcal{V} = \{i \mid 1 \leq i \leq n\}$ 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. Now we can present the steps of our algorithm for computing the maximal palindromic factorization of a given string s of length n. The steps 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 $P \equiv n \rightsquigarrow v$ such that v is the end position of a palindrome belonging to $\mathcal{PP}(s)$. Suppose $P \equiv \langle n = p_k, p_{k-1}, \ldots, p_2, p_1 = v \rangle$.
7: Return $s = s[1..p_1]\ s[p_1+1..p_2]\ \cdots\ s[p_{k-1}+1..p_k]$.

4 Analysis

We now have the following theorem which proves the correctness of MPF Algorithm.

Theorem 5 (Correctness and Running time). *Given a string s of length n, MPF Algorithm correctly computes the maximal palindromic factorization of s in $O(n)$ time.*

Proof. **Correctness:**
We first focus on an edge $(i,j) \in \mathcal{E}$ of the graph $\mathcal{G}_s$ constructed at Step 4 of the algorithm. By definition, this means the following:

1. There is a maximal palindrome pal_i having length ℓ_i (say) ending at position i.
2. There is a maximal palindrome pal_j having length ℓ_j (say) ending at position j.
3. $i > j$.
4. $i - \ell_i = j$.

Since, by definition, each directed edge $(i, j) \in \mathcal{E}$ is such that $i > j$, so, for a path $P \equiv \langle p_k, p_{k-1}, \ldots, p_2, p_1 \rangle$ in $\mathcal{G}_s$, we always have $p_k > p_{k-1} > \cdots > p_1$. A path $P \equiv \langle p_k, p_{k-1}, \ldots, p_2, p_1 \rangle$ can be seen as corresponding to a substring of s formed by concatenation of maximal palindromes as follows. Each edge $(p_i, p_{i-1}) \in P$ corresponds to a palindromic substring $s[p_{i-1}]s[p_{i-1}+1]s[p_{i-1}+2]\cdots s[p_i]$.

Hence, following the definition of the edges, it is clear that any path would correspond to a substring of s formed by concatenation of consecutive palindromic substrings.

In Step 5, a breadth first tree is constructed from $\mathcal{G}_s$ considering the vertex n as the source. A breadth first tree gives the shortest path from the source (in this case, n) to any other node. Now, in Step 6, MPF Algorithm identifies the set of shortest paths (say, $SPath$) between n and j such that j corresponds to a maximal palindromic prefix of s. Now the maximum palindromic factorization must contain exactly one palindrome from $\mathcal{PP}(s)$ and exactly one palindrome from $\mathcal{SP}(s)$, where ℓ is minimum. Hence, it is easy to realize that the shortest one among the paths in $SPath$ corresponds to the maximal palindromic factorization. This completes the correctness proof.

Running time:
In Step 1 the computation of $\mathcal{MP}(s)$ can be done using the algorithm of [25] in $O(n)$ time. Also, $\mathcal{PP}(s)$ and $\mathcal{SP}(s)$ can be computed easily while computing $\mathcal{MP}(s)$. The computation of $\mathcal{MPL}(s)$ and $\mathcal{U}(s)$ in Step 2 and Step 3 can be done in linear time once $\mathcal{MP}(s)$ is computed.

Now construction of the graph $\mathcal{G}_s$ is done in Step 4. There are in total n number of vertices is $\mathcal{G}_s$. The number of edges $|\mathcal{E}|$ of $\mathcal{G}_s$ depends on $\mathcal{U}(s)$. But it is easy to realize that the summation of the number of elements in all the positions of $\mathcal{U}(s)$ cannot exceed the total number of maximal palindromes. Now, since there can be at most $2n + 1$ centers, there can be just as many maximal palindromes in s. Therefore we have $|\mathcal{E}| = O(n)$.

Hence, the graph construction (Step 4) as well as the breadth first search (Step 5) can be done in $O(|\mathcal{V}| + |\mathcal{E}|) = O(n)$ time. Finally, the identification of the desired path in Step 6 can also be done easily if we do some simple bookkeeping during the breadth first search because we already have computed the sets $\mathcal{PP}(s)$ and $\mathcal{SP}(s)$ in Step 1. Hence the total running time of the algorithm is $O(n)$. And this completes the proof. □

4.1 An Illustrative Example

Suppose we are given a string $s = abbcbbcbbbcbb$. We will proceed as follows:

First we compute the set $\mathcal{MP}(s)$. For example, at position $i = 9$ there are 2 palindromes of lengthes 2 and 9 centered at position 9 of s.

Secondly, we compute the set $\mathcal{MPL}(s)$. For example, at position $i = 9$ there are 3 palindromes of lengthes 2, 5 and 8 ending at position 9 of s.

Finally, we compute $\mathcal{U}(s)$ (Table 1 shows full steps for $s = abbcbbcbbbcbb$).

Now, we can construct the graph $\mathcal{G}_s$ easily as shown in Figure 1. For example, we can see that from vertex $i = 9$ we have 3 directed edges, namely, $(9, 7), (9, 4)$ and

$(9, 1)$. Our desired shortest path is $P = \langle 13, 4, 3, 1\rangle$ (corresponding edges are shown as dashed edges). So, the maximal palindromic factorization of $s = abbcbbcbbbcbb$ is as follows:

$$s[1..1]s[2..3]s[4..4]s[5..13] = a\ bb\ c\ bbcbbbcbb\,.$$

i	$\mathcal{MP}[i]$	$\mathcal{MPL}[i]$	$\mathcal{U}[i]$
1	$\mathcal{MP}[1] = \{(1,1)\}$	$\mathcal{MPL}[1] = \{1\}$	$\mathcal{U}[1] = \{0\}$
2	$\mathcal{MP}[2] = \{(2,2)\}$	$\mathcal{MPL}[2] = \{.\}$	$\mathcal{U}[2] = \{.\}$
3	$\mathcal{MP}[3] = \{(3,1)\}$	$\mathcal{MPL}[3] = \{2\}$	$\mathcal{U}[3] = \{1\}$
4	$\mathcal{MP}[4] = \{(4,5)\}$	$\mathcal{MPL}[4] = \{.\}$	$\mathcal{U}[4] = \{.\}$
5	$\mathcal{MP}[5] = \{(5,8)\}$	$\mathcal{MPL}[5] = \{.\}$	$\mathcal{U}[5] = \{.\}$
6	$\mathcal{MP}[6] = \{(6,1)\}$	$\mathcal{MPL}[6] = \{5\}$	$\mathcal{U}[6] = \{1\}$
7	$\mathcal{MP}[7] = \{(7,5)\}$	$\mathcal{MPL}[7] = \{.\}$	$\mathcal{U}[7] = \{.\}$
8	$\mathcal{MP}[8] = \{(8,2)\}$	$\mathcal{MPL}[8] = \{.\}$	$\mathcal{U}[8] = \{.\}$
9	$\mathcal{MP}[9] = \{(9,2)(9,9)\}$	$\mathcal{MPL}[9] = \{2,5,8\}$	$\mathcal{U}[9] = \{7,4,1\}$
10	$\mathcal{MP}[10] = \{(10,1)\}$	$\mathcal{MPL}[10] = \{2\}$	$\mathcal{U}[10] = \{8\}$
11	$\mathcal{MP}[11] = \{(11,5)\}$	$\mathcal{MPL}[11] = \{.\}$	$\mathcal{U}[11] = \{.\}$
12	$\mathcal{MP}[12] = \{(12,2)\}$	$\mathcal{MPL}[12] = \{.\}$	$\mathcal{U}[12] = \{.\}$
13	$\mathcal{MP}[13] = \{(13,1)\}$	$\mathcal{MPL}[13] = \{1,2,5,9\}$	$\mathcal{U}[13] = \{12,11,8,4)\}$

Table 1. Steps for computing $\mathcal{U}(s)$ and $\mathcal{MPL}(s)$ for $s = abbcbbcbbbcbb$

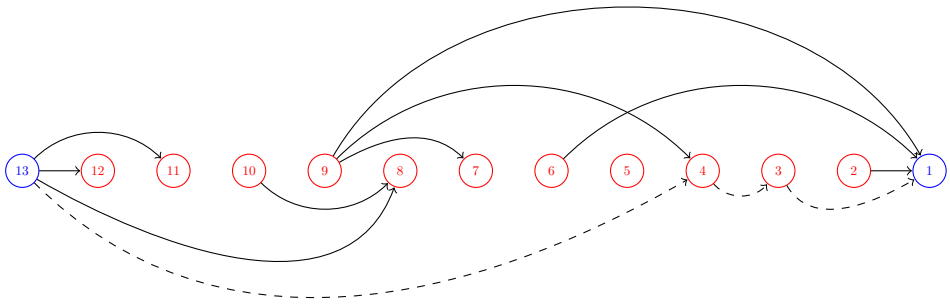

Figure 1. The graph $\mathcal{G}_s$ for $s = abbcbbcbbbcbb$

5 Conclusion

In this paper, we answer a recent question raised during StringMasters, Verona, Italy - 2013: does there exist an algorithm to compute the maximal palindromic factorization of a finite string? Namely, given a finite string, find the smallest set (minimum number of palindromic factors), such that the string is covered by that set of factors with no overlaps. We answer the previous question affirmatively by providing a linear-time algorithm that computes the *maximal palindromic factorization (MPF)* of a string (the algorithms is evaluated with respect to the length of the given string).

An immediate target will be extending the algorithm presented in 3 to biological palindromes, where the word reversal is defined in conjunction with the complementarity of nucleotide letters: $c \leftrightarrow g$ and $a \leftrightarrow t$ (or $a \leftrightarrow u$, in case of RNA). The proposed algorithm can be extended to find *maximal distinct palindromic factorization set*. We will focus on this problem in a future work. Also we will work on studying palindromic cover of string and how can it be modeled using graphs.

Acknowledgements

The authors are grateful to the anonymous reviewers for their helpful comments. This research was carried out when Rahman was visiting King's College London as a Commonwealth Fellow.

References

1. D. Breslauer and Z. Galil: *Finding all periods and initial palindromes of a string in parallel.* Algorithmica, 14 October 1995, pp. 355–366.
2. D. Breslauer, R. Grossi, and F. Mignosi: *Simple real-time constant-space string matching*, in CPM, R. Giancarlo and G. Manzini, eds., vol. 6661 of Lecture Notes in Computer Science, Springer, 2011, pp. 173–183.
3. M. Burrows and D. Wheeler: *A block-sorting lossless data compression algorithm*, tech. rep., Digital SRC Research Report 124, 1994.
4. K.-Y. Chen, P.-H. Hsu, and K.-M. Chao: *Identifying approximate palindromes in run-length encoded strings*, in Proceedings of 21st International Symposium, ISAAC 2010, Jeju, Korea, December 15-17, 2010, 2010, pp. 339–350.
5. C. Choi: *DNA palindromes found in cancer.* Genome Biology, 6 2005.
6. S. R. Chowdhury, M. M. Hasan, S. Iqbal, and M. S. Rahman: *Computing a longest common palindromic subsequence*, to appear in Fundamenta Informaticae.
7. M. Crochemore, J. Désarménien, and D. Perrin: *A note on the Burrows-Wheeler Transformation.* CoRR, abs/cs/0502073 2005.
8. M. Crochemore, L. Ilie, and W. F. Smyth: *A simple algorithm for computing the Lempel Ziv factorization*, in DCC, IEEE Computer Society, 2008, pp. 482–488.
9. M. Crochemore and D. Perrin: *Two-way string matching.* J. ACM, 38(3) 1991, pp. 651–675.
10. X. Droubay: *Palindromes in the Fibonacci word.* Inf. Process. Lett., 55(4) 1995, pp. 217–221.
11. X. Droubay and G. Pirillo: *Palindromes and Sturmian words.* Theoretical Computer Science, 223 1999, pp. 73–85.
12. Z. Galil: *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.
13. J. Y. Gil and D. A. Scott: *A bijective string sorting transform.* CoRR, abs/1201.3077 2012.
14. A. Glen: *Occurrences of palindromes in characteristic Sturmian words.* Theor. Comput. Sci., 352(1–3) 2006, pp. 31–46.
15. D. Gusfield: *Algorithms on strings, trees, and sequences: computer science and computational biology*, Cambridge University Press, New York, NY, USA, 1997.

16. D. Gusfield: *Algorithms on Strings, Trees, and Sequences: Computer Science and Computational Biology*, Cambridge University Press, New York, 1997.
17. P.-H. Hsu, K.-Y. Chen, and K.-M. Chao: *Finding all approximate gapped palindromes*, in Proceedings of 20th International Symposium, ISAAC 2009, Honolulu, Hawaii, USA, December 16-18, 2009., 2009, pp. 1084–1093.
18. T. I, I. Shunsuke, and T. Masayuki: *Palindrome pattern matching*, in Proceedings of 22nd Annual Symposium, CPM 2011, Palermo, Italy, June 27-29, 2011., 2011, pp. 232–245.
19. D. E. Knuth, J. H. M. Jr., and V. R. Pratt: *Fast pattern matching in strings.* SIAM Journal of Computing, 6(2) 1977, pp. 323–350.
20. R. Kolpakov and G. Kucherov: *Searching for gapped palindromes.*
21. R. Kolpakov and G. Kucherov: *On maximal repetitions in words.* J. Discrete Algorithms, 1 1999, pp. 159–186.
22. R. Kolpakov and G. Kucherov: *Searching for gapped palindromes.* Theoretical Computer Science, November 2009, pp. 5365–5373.
23. E. Lamprea-Burgunder, P. Ludin, and P. Mser: *Species-specific typing of dna based on palindrome frequency patterns.* DNA Research, 18 2011, pp. 117–124.
24. J. Lange, H. Skaletsky, S. K. M. van Daalen, S. L. Embry, C. M. Korver, L. G. Brown, R. D. Oates, S. Silber, S. Repping, and D. C. Page: *Isodicentric y chromosomes and sex disorders as byproducts of homologous recombination that maintains palindromes.* Cell, 138 September 2009, pp. 855–869.
25. G. Manacher: *A new linear-time on-line algorithm for finding the smallest initial palindrome of a string.* Journal of the ACM, 22 July 1975, pp. 346–351.
26. T. Martínek and M. Lexa: *Hardware acceleration of approximate palindromes searching*, in Proceedings of The International Conference on Field-Programmable Technology, 2008, pp. 65–72.
27. W. Matsubara, S. Inenaga, A. Ishino, A. Shinohara, T. Nakamura, and K. Hashimoto: *Efficient algorithms to compute compressed longest common substrings and compressed palindromes.* Theoretical Computer Science, 410(8–10) March 2009, pp. 900–913.
28. G. Melançon: *Lyndon factorization of infinite words.*, in STACS, C. Puech and R. Reischuk, eds., vol. 1046 of Lecture Notes in Computer Science, Springer, 1996, pp. 147–154.
29. A. H. L. Porto and V. C. Barbosa: *Finding approximate palindromes in strings.* Pattern Recognition, 2002.
30. *StringMasters, Verona, Italy*, 2013.
31. H. Tanaka, D. A. Bergstrom, M.-C. Yao, and S. J. Tapscott: *Large dna palindromes as a common form of structural chromosome aberrations in human cancers.* Human Cell, 19(1) 2006, pp. 17–23.
32. J. Ziv and A. Lempel: *A universal algorithm for sequential data compression.* IEEE Transactions on Information Theory, 23(3) 1977, pp. 337–343.