Longest Palindromic Substring in Sublinear Time

Panagiotis Charalampopoulos
Reichman University, Herzliya, Israel

Solon P. Pissis
CWI, Amsterdam, The Netherlands
Vrije Universiteit, Amsterdam, The Netherlands

Jakub Radoszewski
Institute of Informatics, University of Warsaw, Poland

Abstract

We revisit the classic algorithmic problem of computing a longest palidromic substring. This problem is solvable by a celebrated $\mathcal{O}(n)$-time algorithm [Manacher, *J. ACM* 1975], where n is the length of the input string. For small alphabets, $\mathcal{O}(n)$ is not necessarily optimal in the word RAM model of computation: a string of length n over alphabet $[0, \sigma)$ can be stored in $\mathcal{O}(n \log \sigma / \log n)$ space and read in $\mathcal{O}(n \log \sigma / \log n)$ time. We devise a simple $\mathcal{O}(n \log \sigma / \log n)$-time algorithm for computing a longest palindromic substring. In particular, our algorithm works in sublinear time if $\sigma = 2^{o(\log n)}$. Our technique relies on periodicity and on the $\mathcal{O}(n \log \sigma / \log n)$-time constructible data structure of Kempa and Kociumaka [STOC 2019] that answers longest common extension queries in $\mathcal{O}(1)$ time.

2012 ACM Subject Classification Theory of computation → Pattern matching

Keywords and phrases string algorithms, longest palindromic substring, longest common extension

Digital Object Identifier 10.4230/LIPIcs.CPM.2022.20

Funding *Panagiotis Charalampopoulos*: Supported by the Israel Science Foundation, grant no. 810/21.
Solon P. Pissis: Supported by the PANGAIA and ALPACA projects that have received funding from the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreements No 872539 and 956229, respectively.
Jakub Radoszewski: Supported by the Polish National Science Center, grant no. 2018/31/D/ST6/03991.

1 Introduction

We start with some basic definitions and notation. Let $S = S[0] \cdots S[n-1]$ be a *string* of length $n = |S|$ over an alphabet Σ of σ *letters*. We consider throughout an integer alphabet $\Sigma = [0, \sigma) \subseteq [0, n)$. The *empty string* is the unique string of length 0. For any two positions i and $j \geq i$ of S, $S[i \,.\,.\, j]$ is the *fragment* of S starting at position i and ending at position j; it is represented in $\mathcal{O}(1)$ space by i and j. The fragment $S[i \,.\,.\, j]$ is an *occurrence* of the underlying *substring* $P = S[i] \cdots S[j]$; we say that P occurs at *position* i in S. A fragment $S[i \,.\,.\, j]$ can be equivalently written as $S[i \,.\,.\, j+1)$, $S(i-1 \,.\,.\, j]$, or $S(i-1 \,.\,.\, j+1)$. A *prefix* of S is a fragment of the form $S[0 \,.\,.\, j]$ and a *suffix* of S is a fragment of the form $S[i \,.\,.\, n)$. A substring of S is *proper* when it does not equal S. By ST we denote the *concatenation* of two strings S and T. We denote the reverse string of S by S^R, i.e., $S^R = S[n-1] \cdots S[0]$. A palindrome is a symmetric word that reads the same backward and forward. Formally, a string S is said to be a *palindrome* if and only if $S = S^R$.

In this work, we consider the classic algorithmic problem of computing a longest palindromic substring.

LONGEST PALINDROMIC SUBSTRING **Input:** A string S of length n over an integer alphabet $[0,\sigma)$ with $\sigma \le n$. **Output:** Positions $i, j \in [0,n)$ such that $S[i\,..\,j]$ is a longest palindromic substring of S.

LONGEST PALINDROMIC SUBSTRING can be solved in $\mathcal{O}(n)$ time by Manacher's celebrated algorithm [34, 3], by Jeuring's algorithm [31] or by Gusfield's simple algorithm, which uses longest common extension queries [28]. Other settings in which the problem has been studied include the compressed setting, where the input string is given as a straight-line program [36], the streaming setting [26], the dynamic setting, where the string undergoes updates [1, 2], and a semi-dynamic setting [23]. Le Gall and Seddighin [24] have recently presented a strongly sublinear-time quantum algorithm for the problem and a quantum lower bound.

The detection of palindromes is a well-studied problem with a lot of variants [29, 30, 19, 25, 5, 22, 12, 41, 40, 39] arising out of different practical scenarios. For instance, in computational biology, palindromes are found in both prokaryotic and eukaryotic genomes and they have been linked with countless possible functions. They play an important role in the regulation of gene activity and other cell processes because these are often observed near promoters, introns, and specific untranslated regions; for more details see [38, 13, 18, 42, 43, 35].

Our Model and Result

The main contribution of our work is to improve on the existing linear-time solutions to LONGEST PALINDROMIC SUBSTRING in the word RAM model of computation when the input string is given in a packed representation. Let us now describe this model in more detail.

We assume the unit-cost word RAM model with word size $w = \Theta(\log n)$ and a standard instruction set including arithmetic operations, bitwise Boolean operations, and shifts. We count the space complexity of our algorithms in machine words used by the algorithm. The *packed representation* of a string S over an integer alphabet $[0,\sigma)$ is a list obtained by storing $\Theta(\log_\sigma n)$ letters per machine word thus representing S in $\mathcal{O}(|S|/\log_\sigma n)$ machine words. If S is given in the packed representation we simply say that S is a *packed string*.

We prove the following result.

▶ **Theorem 1.** *LONGEST PALINDROMIC SUBSTRING can be solved in $\mathcal{O}(n/\log_\sigma n)$ time, if the input is given in a packed representation.*

In Section 2 we provide the necessary background. In Section 3 we recall the linear-time algorithm for solving LONGEST PALINDROMIC SUBSTRING by Gusfield [28]. We provide our sublinear-time algorithm in Section 4 and conclude in Section 5.

Other Related Work

A large body of work exploits bit-level parallelism in the word RAM model to speed-up string matching algorithms; see [4, 37, 21, 9, 6, 10, 7, 14, 27, 8, 11, 15] and references therein.

2 Preliminaries

Palindromes. Let S be a string of length n. If $S[i\,..\,j]$, $0 \le i \le j < n$, is a palindrome, the number $\frac{i+j}{2}$ is called the *center* of $S[i\,..\,j]$ and the number $\frac{j-i+1}{2}$ is called the *radius* of $S[i\,..\,j]$. A palindromic fragment $S[i\,..\,j]$ of S 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 of S can be a fragment of another palindrome of S and that the longest palindrome in S must be maximal.

Periodicity. A positive integer p is called a *period* of a string S if $S[i] = S[i+p]$ for all $i \in [0, |S| - p)$. We refer to the smallest period as *the period* of the string, and denote it by $\mathsf{per}(S)$. A string S is called *periodic* if $2 \cdot \mathsf{per}(S) \leq |S|$. A *border* of a nonempty string S is a proper substring of S that occurs both as a prefix and as a suffix of S. A string S has a period p if and only if it has a border of length $|S| - p$.

▶ **Lemma 2** (Periodicity Lemma (weak version) [20]). *If a string S has periods p and q such that $p + q \leq |S|$, then $\gcd(p, q)$ is also a period of S.*

Let $\mathcal{B}(S)$ denote the set of lengths of borders of S. The following characterization of long borders of a string is generally known; cf. [17]. We give a proof of the lemma for completeness.

▶ **Lemma 3.** *Assume that a string S of length n is periodic with smallest period p. Then $\mathcal{B}(S) \cap [p, n] = \{n - kp : k \in \mathbb{Z}^+\} \cap [p, n]$.*

Proof. ($\subseteq$) If $b \in \mathcal{B}(S)$, then $q = n - b$ is a period of S. As p is a period of S as well, if $b \geq p$, then by the Periodicity Lemma $\gcd(p, q)$ is also a period of S. This means that p divides q, as otherwise $\gcd(p, q)$ would have been a period of S smaller than p, which is impossible.

($\supseteq$) For each integer $k \in [0, n/p)$, the string S has a period kp and hence a border of length $n - kp$. ◀

Longest Common Extension. An important building block of our technique is a so-called longest common extension data structure, first used by Landau and Vishkin in their textbook solution for approximate pattern matching with at most k mismatches [33]. Let us denote the lengths of the longest common prefix and the longest common suffix of two strings U and V by $\mathsf{LCP}(U, V)$ and $\mathsf{LCS}(U, V) = \mathsf{LCP}(U^R, V^R)$ respectively. Given a string S, it is often useful to have a data structure that can efficiently return $\mathsf{LCP}(S[i \,..\, n), S[j \,..\, n))$ or $\mathsf{LCS}(S[0 \,..\, i], S[0 \,..\, j])$; we collectively call such queries *longest common extension (LCE) queries*. Kempa and Kociumaka presented an optimal LCE data structure for packed strings.

▶ **Theorem 4** ([32, Theorem 5.4]). *Given a packed representation of a string $S \in [0, \sigma)^n$, LCE queries on S can be answered in $\mathcal{O}(1)$ time after $\mathcal{O}(n/\log_\sigma n)$-time preprocessing.*

3 LCE-based Linear-Time Algorithm

We describe the linear-time algorithm given by Gusfield for Longest Palindromic Substring [28]. Gusfield's algorithm is based on the following simple fact – its proof follows by the definition of palindromes and by the definition of $\mathsf{LCP}(U, V)$ for two strings U, V.

▶ **Fact 5.** *Let S be a string of length n. $S[i \,..\, j]$ is a palindrome of odd length with center $c = \frac{j+i}{2}$ if and only if $\mathsf{LCP}(S[c+1 \,..\, n), (S[0 \,..\, c-1])^R) \geq \frac{j-i}{2}$. $S[i \,..\, j]$ is a palindrome of even length with center $c = \frac{i+j}{2}$ if and only if $\mathsf{LCP}(S[\lceil c \rceil \,..\, n), (S[0 \,..\, \lfloor c \rfloor])^R) \geq \frac{j-i+1}{2}$.*

Thus, after constructing an LCE data structure for string $T = SS^R$, it suffices to perform $\mathcal{O}(n)$ LCP queries: one for each integer or half-integer possible center in $[0, n)$. By using any LCE data structure, which is constructible in $\mathcal{O}(n)$ time and answers LCP queries in $\mathcal{O}(1)$ time, such as the one by Landau and Vishkin [33], we obtain a linear-time solution to Longest Palindromic Substring; in fact this algorithm computes all maximal palindromes.

4 Computing a Longest Palindromic Substring in Sublinear Time

The main goal of this section is to prove Theorem 1; namely, to design an algorithm for LONGEST PALINDROMIC SUBSTRING that works in $\mathcal{O}(n/\log_\sigma n)$ time. Recall that our input is a string S of length n over alphabet $[0,\sigma)$. Let us set $\ell' = \max(1, \lfloor \frac{1}{8}\log_\sigma n \rfloor)$ and $\ell = 4\ell'$. Intuitively, ℓ' and ℓ correspond to lengths of chunks and extended chunks of S, respectively. Our algorithm proceeds with processing every chunk separately. We assume that $n \geq 8$.

Preprocessing. We compute the radii of maximal palindromes with each possible center for every distinct length-ℓ string over $[0,\sigma)$. The number of such length-ℓ strings is $\sigma^\ell = \sigma^{4\ell'} = \mathcal{O}(\sqrt{n})$ and each of them can be stored in one machine word. All the radii can be computed using Manacher's algorithm [34] in $\mathcal{O}(\ell)$ time per string, which takes $\mathcal{O}(\ell\sqrt{n}) = o(n/\log_\sigma n)$ time overall. In the end of the preprocessing step, we store, in an $\mathcal{O}(\sqrt{n})$-sized array, for each length-ℓ string X, a constant amount of data:

(a) a longest palindrome in X;
(b) a longest palindrome in X that has its center in $[\ell/2-\ell', \ell/2-\frac{1}{2}]$; and
(c) the two longest prefix palindromes of X, if they exist.

Algorithm. The precomputed data allows us to compute the longest palindrome in the length-ℓ prefix and in the length-ℓ suffix of S. This will account for the longest palindrome with the center in the first and last $\ell/2$ positions of S. Let us partition $S[\ell/2 \,..\, n-\ell/2)$ into chunks of length ℓ'; if the final chunk has length smaller than ℓ', we complete it to a length-ℓ' string by taking letters of S preceding it. Our goal is to compute, for each chunk C, the longest palindrome in the whole string S with a center in C; let us note that this palindrome may be much longer than chunk C, as its length may even be $\Theta(n)$. We assume that a chunk $S[i \,..\, i+\ell')$ includes all centers in $[i, i+\ell'-\frac{1}{2}]$, consistently with Item b above.

For each chunk C, we consider the length-ℓ fragment X (*extended chunk*) of S such that C is the second quarter of X, i.e., C is a suffix of $X[0 \,..\, \ell/2)$. Let $\mathcal{P}_X$ denote the set of maximal palindromes in S with centers in C that either exceed X or are prefixes of X (inspect Figure 1 for an illustration). We will show that the longest palindrome in $\mathcal{P}_X$ can be computed in the time required to answer $\mathcal{O}(1)$ LCP queries on substrings of SS^R.

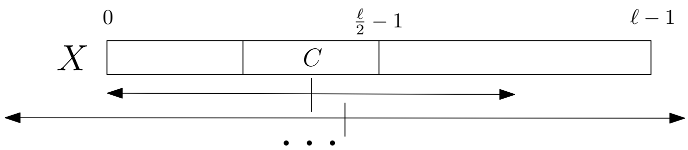

Figure 1 Two of the possible palindromes from the set $\mathcal{P}_X$.

Using the packed representation of S, we can recover the string X packed into one machine word in constant time with word RAM operations. Using the precomputed data for X, we know the at most two longest prefix palindromes P_1, P_2 of X; we assume that $|P_1| > |P_2|$. If any P_i, $i \in \{1,2\}$, satisfies $|P_i| \leq \ell - 2\ell'$, we discard it, as the center of the occurrence of this palindrome as a prefix of X does not lie in C. Let $\mathcal{Q}_X$ be the set of palindromes which are prefixes of X of length greater than $\ell - 2\ell'$. Let us note that each of P_1 and P_2 that was not discarded belongs to $\mathcal{Q}_X$. Each palindrome $P \in \mathcal{P}_X$ has a subpalindrome (palindromic substring) $P' \in \mathcal{Q}_X$ with the same center. If P_1 does not exist, then $\mathcal{P}_X = \emptyset$. If P_1 exists but P_2 does not, then $|\mathcal{P}_X| = 1$. In this case, we can apply Fact 5 to compute the only palindrome in $\mathcal{P}_X$ from P_1 using one LCP query on suffixes of SS^R.

Finally, we consider the case where both P_1 and P_2 exist. Here we use the following well-known property of palindromes.

▶ **Lemma 6** (cf. [19, Lemma 3]). *Let U be a proper prefix of a palindrome V. Then $|V|-|U|$ is a period of V if and only if U is a palindrome. In particular, $\mathsf{per}(V)=|V|-|U|$ if and only if U is the longest palindromic proper prefix of V.*

Let $p := |P_1| - |P_2|$. By Lemma 6, $p = \mathsf{per}(P_1)$. We can check how far this periodicity extends on both sides by using two LCE queries. Namely, if $X = S[i\,..\,i+\ell)$, the maximal fragment with period p that contains P_1 is $S[i-a\,..\,i+b)$, where $a := \mathsf{LCS}(S[0\,..\,i), S[0\,..\,i+p))$ and $b := p+\mathsf{LCP}(S[i\,..\,n), S[i+p\,..\,n))$. We next provide a characterization of the lengths of the palindromes in $\mathcal{Q}_X$.

▶ **Lemma 7.** *Let P_1 be the longest prefix palindrome in $\mathcal{Q}_X$. Further, let $p = \mathsf{per}(P_1)$. The set of lengths of prefix palindromes in $\mathcal{Q}_X$ is $L := \{|P_1| - kp : k \geq 0\} \cap (\ell - 2\ell', \ell]$.*

Proof. ($\subseteq$) Let $Q \in \mathcal{Q}_X$. We have $p < \ell - (\ell - 2\ell') = 2\ell'$ and $|Q| > \ell - 2\ell'$, so $|Q| - p > \ell - 4\ell' = 0$. By Lemma 6, Q is a border of P_1, so by Lemma 3, $|Q| \in L$.

($\supseteq$) For each k such that $|P_1| - kp \in L$, the string P_1 has a period kp, hence a border of length $|P_1| - kp$. This border is a palindrome by Lemma 6. ◀

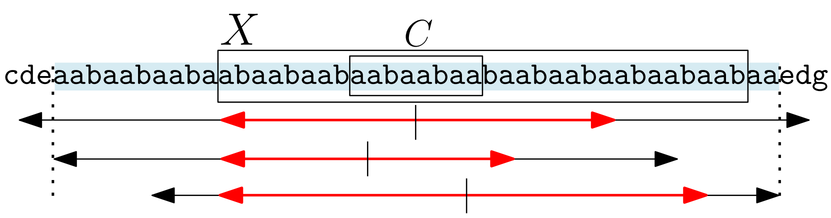

Figure 2 Configuration in Lemmas 7 and 8 for $\ell' = 8$ and $\ell = 4\ell' = |X|$. The prefix palindromes in $\mathcal{Q}_X$ are denoted by red arrows; the maximal palindromes in $\mathcal{P}_X$ are denoted by black arrows. The fragment $S[i-a\,..\,i+b)$ is shaded in blue.

The periodicity of the elements of $\mathcal{Q}_X$ enables an efficient computation of the longest palindrome in $\mathcal{P}_X$. For intuition, consider answering $\mathsf{LCP}(S[\lceil c\rceil\,..\,n), (S[0\,..\,\lfloor c\rfloor])^R)$ (see Fact 5), for some half-integer $c \in [i-a\,..\,i+b) \setminus \mathbb{Z}$, by comparing pairs of letters: either we reach $i-a$ and $i+b-1$ at the same time, which can happen for at most a single value of c, namely for $(i-a+i+b-1)/2 = (2i-a+b-1)/2$, or we reach one of the two endpoints first, in which case we get a mismatch (inspect Figure 2).

▶ **Lemma 8.** *Let P_1 be the longest prefix palindrome in $\mathcal{Q}_X$. Further let $P_1 = S[i\,..\,i+|P_1|)$, $p = \mathsf{per}(P_1)$, $a = \mathsf{LCS}(S[0\,..\,i), S[0\,..\,i+p))$ and $b = p + \mathsf{LCP}(S[i\,..\,n), S[i+p\,..\,n))$. For palindromes $P \in \mathcal{P}_X$ and $Q \in \mathcal{Q}_X$ with the same center c, either $|Q| = b-a$ or $|P| = \min(|Q|+2a,\, 2b-|Q|)$.*

Proof. Let us first consider the case where Q and P are even-length palindromes. Note that $\lceil c\rceil = i + |Q|/2$ and recall that $|Q| \geq 2p$. Let $F := S[\lceil c\rceil\,..\,\lceil c\rceil + p)$. We have

$$\lambda := \mathsf{LCP}((S[0\,..\,\lceil c\rceil))^R, F^n) = |S[i-a\,..\,\lceil c\rceil)| = \lceil c\rceil - i + a = |Q|/2 + a, \text{ and}$$

$$\rho := \mathsf{LCP}(S[\lceil c\rceil\,..\,n), F^n) = |S[\lceil c\rceil\,..\,i+b)| = i + b - \lceil c\rceil = b - |Q|/2.$$

Then, if $\lambda \neq \rho$, we have

$$|P| = 2 \cdot \mathsf{LCP}((S[0\,..\,\lceil c \rceil))^R, S[\lceil c \rceil\,..\,n)) = 2 \cdot \min\{\lambda, \rho\} = \min\{|Q| + 2a, 2b - |Q|\}.$$

Else, we have $\lambda = \rho \Leftrightarrow |Q|/2 + a = b - |Q|/2 \Leftrightarrow |Q| = b - a$.

The proof for the case where Q and P are odd-length palindromes is similar, but we include it for completeness. In this case, $c = i + (|Q| - 1)/2$. Let $F := S[c\,..\,c + p)$. We have

$$\lambda := \mathsf{LCP}((S[0\,..\,c])^R, F^n) = |S[i - a\,..\,c]| = c - i + a + 1 = |Q|/2 + 1/2 + a, \text{ and}$$

$$\rho := \mathsf{LCP}(S[c\,..\,n), F^n) = |S[c\,..\,i + b)| = i + b - c = b - |Q|/2 + 1/2.$$

Then, if $\lambda \neq \rho$, we have

$$|P| = 2 \cdot \mathsf{LCP}((S[0\,..\,c])^R, S[c\,..\,n)) - 1 = 2 \cdot \min\{\lambda - 1/2, \rho - 1/2\} = \min\{|Q| + 2a, 2b - |Q|\}.$$

Else, we have $\lambda = \rho \Leftrightarrow |Q| = b - a$ as before. ◀

We use Lemma 8 to compute the longest palindrome in $\mathcal{P}_X$ as follows. For two palindromes $Q \in \mathcal{Q}_X$ and $P \in \mathcal{P}_X$ with the same center such that $|Q| \neq b - a$, either $|Q| < b - a \Leftrightarrow |Q| + 2a < 2b - |Q|$ and hence $|P| = |Q| + 2a$ due to Lemma 8 or $|Q| > b - a \Leftrightarrow |Q| + 2a > 2b - |Q|$ and hence $|P| = 2b - |Q|$ due to Lemma 8. Thus, it suffices to consider only three palindromes in $\mathcal{Q}_X$. Specifically, with the characterization of Lemma 7 we compute in $\mathcal{O}(1)$ time: the longest palindrome Q_1 in $\mathcal{Q}_X$ of length smaller than $b - a$; the shortest palindrome Q_2 in $\mathcal{Q}_X$ of length greater than $b - a$; and check if there is a palindrome Q_3 in $\mathcal{Q}_X$ of length $b - a$. Finally we pick the longest of the following palindromes from $\mathcal{P}_X$:

- The palindrome P_I corresponding to Q_1 if Q_1 exists; the length of P_I is $|Q_1| + 2a$ due to the formula from Lemma 8.
- The palindrome P_{II} corresponding to Q_2 if Q_2 exists; the length of P_{II} is $2b - |Q_2|$ due to the formula from Lemma 8.
- The palindrome P_{III} corresponding to Q_3 if Q_3 exists; the center of P_{III} is $i + (|Q_3| - 1)/2$ and hence the length of P_{III} can be computed using one LCP query on suffixes of SS^R due to Fact 5.

Thus we have proved the following lemma.

▶ **Lemma 9.** *The longest palindrome in $\mathcal{P}_X$ can be computed in the time required to answer $\mathcal{O}(1)$ LCP queries on suffixes of SS^R.*

For each chunk C (we have $\mathcal{O}(n/\ell)$ of them), we take the longer palindrome of the one computed by an application of Lemma 9 and the longest palindrome stored in Item b for the corresponding substring X. Over all chunks, using Theorem 4 to answer LCE queries in $\mathcal{O}(1)$ time, the algorithm requires time $\mathcal{O}(n/\log_\sigma n)$, and we thus obtain Theorem 1.

5 Final Remarks

We have shown an $\mathcal{O}(n/\log_\sigma n)$-time algorithm for computing a longest palindromic substring of a string of length n over alphabet $[0, \sigma)$. Our algorithm can be easily modified to compute the number of all palindromic fragments of the string within the same time complexity.

We anticipate that our technique will be applicable in many other problems on strings, which currently admit only linear-time solutions. For instance, our approach applied to the prefix array of a string [16] can be used to compute the longest repeating prefix of a string of length n over alphabet $[0, \sigma)$, still in $\mathcal{O}(n/\log_\sigma n)$ time.

References

1 Amihood Amir and Itai Boneh. Dynamic palindrome detection. *CoRR*, abs/1906.09732, 2019. arXiv:1906.09732.

2 Amihood Amir, Panagiotis Charalampopoulos, Solon P. Pissis, and Jakub Radoszewski. Dynamic and internal longest common substring. *Algorithmica*, 82(12):3707–3743, 2020. doi:10.1007/s00453-020-00744-0.

3 Alberto Apostolico, Dany Breslauer, and Zvi Galil. Parallel detection of all palindromes in a string. *Theoretical Computer Science*, 141(1):163–173, 1995. doi:10.1016/0304-3975(94)00083-U.

4 Ricardo A. Baeza-Yates. Improved string searching. *Software: Practice and Experience*, 19(3):257–271, 1989. doi:10.1002/spe.4380190305.

5 Hideo Bannai, Travis Gagie, Shunsuke Inenaga, Juha Kärkkäinen, Dominik Kempa, Marcin Piatkowski, and Shiho Sugimoto. Diverse palindromic factorization is NP-complete. *International Journal of Foundations of Computer Science*, 29(2):143–164, 2018. doi:10.1142/S0129054118400014.

6 Djamal Belazzougui. Worst case efficient single and multiple string matching in the RAM model. In *Combinatorial Algorithms – 21st International Workshop, IWOCA 2010*, volume 6460 of *Lecture Notes in Computer Science*, pages 90–102. Springer, 2010. doi:10.1007/978-3-642-19222-7_10.

7 Oren Ben-Kiki, Philip Bille, Dany Breslauer, Leszek Gasieniec, Roberto Grossi, and Oren Weimann. Optimal packed string matching. In *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2011*, volume 13 of *LIPIcs*, pages 423–432. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2011. doi:10.4230/LIPIcs.FSTTCS.2011.423.

8 Oren Ben-Kiki, Philip Bille, Dany Breslauer, Leszek Gasieniec, Roberto Grossi, and Oren Weimann. Towards optimal packed string matching. *Theoretical Computer Science*, 525:111–129, 2014. doi:10.1016/j.tcs.2013.06.013.

9 Philip Bille. Fast searching in packed strings. In *Combinatorial Pattern Matching, 20th Annual Symposium, CPM 2009*, volume 5577 of *Lecture Notes in Computer Science*, pages 116–126. Springer, 2009. doi:10.1007/978-3-642-02441-2_11.

10 Philip Bille. Fast searching in packed strings. *Journal of Discrete Algorithms*, 9(1):49–56, 2011. doi:10.1016/j.jda.2010.09.003.

11 Philip Bille, Inge Li Gørtz, and Frederik Rye Skjoldjensen. Deterministic indexing for packed strings. In *28th Annual Symposium on Combinatorial Pattern Matching, CPM 2017*, volume 78 of *LIPIcs*, pages 6:1–6:11. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2017. doi:10.4230/LIPIcs.CPM.2017.6.

12 Kirill Borozdin, Dmitry Kosolobov, Mikhail Rubinchik, and Arseny M. Shur. Palindromic length in linear time. In *28th Annual Symposium on Combinatorial Pattern Matching, CPM 2017*, volume 78 of *LIPIcs*, pages 23:1–23:12. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2017. doi:10.4230/LIPIcs.CPM.2017.23.

13 Václav Brázda, Martin Bartas, Jiří Lỳsek, Jan Coufal, and Miroslav Fojta. Global analysis of inverted repeat sequences in human gene promoters reveals their non-random distribution and association with specific biological pathways. *Genomics*, 112(4):2772–2777, 2020. doi:10.1016/j.ygeno.2020.03.014.

14 Dany Breslauer, Leszek Gasieniec, and Roberto Grossi. Constant-time word-size string matching. In *Combinatorial Pattern Matching – 23rd Annual Symposium, CPM 2012*, pages 83–96, 2012. doi:10.1007/978-3-642-31265-6_7.

15 Panagiotis Charalampopoulos, Tomasz Kociumaka, Solon P. Pissis, and Jakub Radoszewski. Faster algorithms for longest common substring. In *29th Annual European Symposium on Algorithms, ESA 2021*, volume 204 of *LIPIcs*, pages 30:1–30:17. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPIcs.ESA.2021.30.

16 Maxime Crochemore, Christophe Hancart, and Thierry Lecroq. *Algorithms on strings*. Cambridge University Press, 2007.

17 Maxime Crochemore, Costas S. Iliopoulos, Tomasz Kociumaka, Marcin Kubica, Jakub Radoszewski, Wojciech Rytter, Wojciech Tyczyński, and Tomasz Waleń. The maximum number of squares in a tree. In *Combinatorial Pattern Matching – 23rd Annual Symposium, CPM 2012*, pages 27–40, 2012. doi:10.1007/978-3-642-31265-6_3.

18 Michaela Čutová, Jacinta Manta, Otília Porubiaková, Patrik Kaura, Jiří Šťastnỳ, Eva B. Jagelská, Pratik Goswami, Martin Bartas, and Václav Brázda. Divergent distributions of inverted repeats and G-quadruplex forming sequences in Saccharomyces cerevisiae. *Genomics*, 112(2):1897–1901, 2020. doi:10.1016/j.ygeno.2019.11.002.

19 Gabriele Fici, Travis Gagie, Juha Kärkkäinen, and Dominik Kempa. A subquadratic algorithm for minimum palindromic factorization. *Journal of Discrete Algorithms*, 28:41–48, 2014. doi:10.1016/j.jda.2014.08.001.

20 Nathan J. Fine and Herbert S. Wilf. Uniqueness theorems for periodic functions. *Proceedings of the American Mathematical Society*, 16(1):109–114, 1965. URL: http://www.jstor.org/stable/2034009.

21 Kimmo Fredriksson. Shift-or string matching with super-alphabets. *Information Processing Letters*, 87(4):201–204, 2003. doi:10.1016/S0020-0190(03)00296-5.

22 Mitsuru Funakoshi and Takuya Mieno. Minimal unique palindromic substrings after single-character substitution. In *String Processing and Information Retrieval – 28th International Symposium, SPIRE 2021*, pages 33–46, 2021. doi:10.1007/978-3-030-86692-1_4.

23 Mitsuru Funakoshi, Yuto Nakashima, Shunsuke Inenaga, Hideo Bannai, and Masayuki Takeda. Computing longest palindromic substring after single-character or block-wise edits. *Theoretical Computer Science*, 859:116–133, 2021. doi:10.1016/j.tcs.2021.01.014.

24 François Le Gall and Saeed Seddighin. Quantum meets fine-grained complexity: Sublinear time quantum algorithms for string problems. In *13th Innovations in Theoretical Computer Science Conference, ITCS 2022*, volume 215 of *LIPIcs*, pages 97:1–97:23. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPIcs.ITCS.2022.97.

25 Pawel Gawrychowski, Tomohiro I, Shunsuke Inenaga, Dominik Köppl, and Florin Manea. Tighter bounds and optimal algorithms for all maximal α-gapped repeats and palindromes – finding all maximal α-gapped repeats and palindromes in optimal worst case time on integer alphabets. *Theory of Computing Systems*, 62(1):162–191, 2018. doi:10.1007/s00224-017-9794-5.

26 Pawel Gawrychowski, Oleg Merkurev, Arseny M. Shur, and Przemyslaw Uznański. Tight tradeoffs for real-time approximation of longest palindromes in streams. *Algorithmica*, 81(9):3630–3654, 2019. doi:10.1007/s00453-019-00591-8.

27 Emanuele Giaquinta, Szymon Grabowski, and Kimmo Fredriksson. Approximate pattern matching with k-mismatches in packed text. *Information Processing Letters*, 113(19-21):693–697, 2013. doi:10.1016/j.ipl.2013.07.002.

28 Dan Gusfield. *Algorithms on Strings, Trees, and Sequences – Computer Science and Computational Biology*. Cambridge University Press, 1997. doi:10.1017/cbo9780511574931.

29 Tomohiro I, Shunsuke Inenaga, and Masayuki Takeda. Palindrome pattern matching. *Theoretical Computer Science*, 483:162–170, 2013. doi:10.1016/j.tcs.2012.01.047.

30 Tomohiro I, Shiho Sugimoto, Shunsuke Inenaga, Hideo Bannai, and Masayuki Takeda. Computing palindromic factorizations and palindromic covers on-line. In *Combinatorial Pattern Matching – 25th Annual Symposium, CPM 2014*, volume 8486 of *Lecture Notes in Computer Science*, pages 150–161. Springer, 2014. doi:10.1007/978-3-319-07566-2_16.

31 Johan Jeuring. The derivation of on-line algorithms, with an application to finding palindromes. *Algorithmica*, 11(2):146–184, 1994. doi:10.1007/BF01182773.

32 Dominik Kempa and Tomasz Kociumaka. String synchronizing sets: sublinear-time BWT construction and optimal LCE data structure. In *51st Annual ACM SIGACT Symposium on Theory of Computing, STOC 2019*, pages 756–767. ACM, 2019. doi:10.1145/3313276.3316368.

33 Gad M. Landau and Uzi Vishkin. Efficient string matching with k mismatches. *Theoretical Computer Science*, 43:239–249, 1986. `doi:10.1016/0304-3975(86)90178-7`.

34 Glenn K. Manacher. A new linear-time "on-line" algorithm for finding the smallest initial palindrome of a string. *Journal of the ACM*, 22(3):346–351, 1975. `doi:10.1145/321892.321896`.

35 Fernando Martínez-Alberola, Eva Barreno, Leonardo M Casano, Francisco Gasulla, Arantzazu Molins, Patricia Moya, María González-Hourcade, and Eva M. Del Campo. The chloroplast genome of the lichen-symbiont microalga Trebouxia sp. Tr9 (Trebouxiophyceae, Chlorophyta) shows short inverted repeats with a single gene and loss of the rps4 gene, which is encoded by the nucleus. *Journal of Phycology*, 56(1):170–184, 2020. `doi:10.1111/jpy.12928`.

36 Wataru Matsubara, Shunsuke Inenaga, Akira Ishino, Ayumi Shinohara, Tomoyuki Nakamura, and Kazuo Hashimoto. Efficient algorithms to compute compressed longest common substrings and compressed palindromes. *Theoretical Computer Science*, 410(8-10):900–913, 2009. `doi:10.1016/j.tcs.2008.12.016`.

37 Gonzalo Navarro and Mathieu Raffinot. A bit-parallel approach to suffix automata: Fast extended string matching. In *Combinatorial Pattern Matching, 9th Annual Symposium, CPM 1998*, volume 1448 of *Lecture Notes in Computer Science*, pages 14–33. Springer, 1998. `doi:10.1007/BFb0030778`.

38 Christopher E. Pearson, Haralabos Zorbas, Gerald B. Price, and Maria Zannis-Hadjopoulos. Inverted repeats, stem-loops, and cruciforms: significance for initiation of DNA replication. *Journal of Cellular Biochemistry*, 63(1):1–22, 1996. `doi:10.1002/(SICI)1097-4644(199610)63:1<1::AID-JCB1>3.0.CO;2-3`.

39 Mikhail Rubinchik and Arseny M. Shur. Counting palindromes in substrings. In *String Processing and Information Retrieval – 24th International Symposium, SPIRE 2017*, volume 10508 of *Lecture Notes in Computer Science*, pages 290–303. Springer, 2017. `doi:10.1007/978-3-319-67428-5_25`.

40 Mikhail Rubinchik and Arseny M. Shur. EERTREE: an efficient data structure for processing palindromes in strings. *European Journal of Combinatorics*, 68:249–265, 2018. `doi:10.1016/j.ejc.2017.07.021`.

41 Mikhail Rubinchik and Arseny M. Shur. Palindromic k-factorization in pure linear time. In *45th International Symposium on Mathematical Foundations of Computer Science, MFCS 2020*, volume 170 of *LIPIcs*, pages 81:1–81:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020. `doi:10.4230/LIPIcs.MFCS.2020.81`.

42 Xin Tao, Shaochun Yuan, Fan Chen, Xiaoman Gao, Xinli Wang, Wenjuan Yu, Song Liu, Ziwen Huang, Shangwu Chen, and Anlong Xu. Functional requirement of terminal inverted repeats for efficient ProtoRAG activity reveals the early evolution of V(D)J recombination. *National Science Review*, 7(2):403–417, 2020. `doi:10.1093/nsr/nwz179`.

43 Ran Zhou, David Macaya-Sanz, Craig H. Carlson, Jeremy Schmutz, Jerry W. Jenkins, David Kudrna, Aditi Sharma, Laura Sandor, Shengqiang Shu, Kerrie Barry, et al. A willow sex chromosome reveals convergent evolution of complex palindromic repeats. *Genome Biology*, 21(1):1–19, 2020. `doi:10.1186/s13059-020-1952-4`.