Longest α-Gapped Repeat and Palindrome

Paweł Gawrychowski[1] and Florin Manea[2(✉)]

[1] Institute of Informatics, University of Warsaw, Warsaw, Poland
gawry@mimuw.edu.pl
[2] Department of Computer Science, Kiel University, Kiel, Germany
flm@informatik.uni-kiel.de

Abstract. We propose an efficient algorithm finding, for a word w and an integer $\alpha > 0$, the longest word u such that w has a factor uvu, with $|uv| \leq \alpha|u|$ (i.e., the longest α-gapped repeat of w). Our algorithm runs in $\mathcal{O}(\alpha n)$ time. Moreover, it can be easily adapted to find the longest u such that w has a factor $u^R vu$, with $|uv| \leq \alpha|u|$ (i.e., the longest α-gapped palindrome), again in $\mathcal{O}(\alpha n)$ time.

1 Introduction

Gapped repeats and palindromes have been investigated for a long time (see, e.g., [2,4,6,8,12–14] and the references therein), with motivation coming especially from the analysis of DNA and RNA structures, where tandem and interspersed repeats as well as hairpin structures play important roles in revealing structural and functional information of the analysed genetic sequence (see, e.g., [2,8,12] and the references therein).

Following [12,13], we analyse gapped repeats uvu or palindromes $u^R vu$ where the length of the gap v is upper bounded by the length of the arm u multiplied by some factor. More precisely, in [13], the authors investigate *α-gapped repeats*: words uvu with $|uv| \leq \alpha|u|$. Similarly, [12] deals with *α-gapped palindromes*, i.e., words $u^R vu$ with $|uv| \leq \alpha|u|$. For $\alpha = 2$, these structures are called long armed repeats (or pairs) and palindromes, respectively; for $\alpha = 1$, they are squares and palindromes of even length, respectively. Intuitively, one is interested in repeats or palindromes whose arms are roughly close one to the other; therefore, the study of α-gapped repeats and palindromes was rather focused on the cases with small α. Here, we address the general case, of searching in a word w α-gapped repeats or palindromes for $\alpha \leq |w|$.

In [12] the authors propose an algorithm that, given a word of length n, finds the set S of all its factors which are maximal α-gapped palindromes (i.e., the arms cannot be extended to the right or to the left) in $\mathcal{O}(\alpha^2 n + |S|)$ time. No upper bound on the possible size of the set S was given in [12], but, following the ideas of [13], we conjecture it is $\mathcal{O}(\alpha^2 n)$.

The algorithms of [2] can be directly used to find in $\mathcal{O}(n \log n + |S|)$ time the set S of all the factors of a word of length n which are maximal α-gapped repeats.

The work of Florin Manea was supported by the DFG grant 596676.

A. Kosowski and I. Walukiewicz (Eds.): FCT 2015, LNCS 9210, pp. 27–40, 2015.
DOI: 10.1007/978-3-319-22177-9_3

Note that all the square factors of a word are maximal α-gapped repeats, with empty gap; thus, there are words for which $|S|$ is $\Omega(n \log n)$ (see, e.g., [3]). In [13] the size of the set of maximal α-gapped repeats with non-empty gap is shown to be $\mathcal{O}(\alpha^2 n)$, and can be computed in $\mathcal{O}(\alpha^2 n)$ time for integer alphabets.

A classical problem for palindromes asks to find the longest palindromic factor of a word [15]. Inspired by this, we address the following problems.

Problem 1. Given a word w and $\alpha \leq |w|$, find the longest word u such that w has a factor uvu with $|uv| \leq \alpha|u|$ (the longest α-gapped repeat of w).

Problem 2. Given a word w and $\alpha \leq |w|$, find the longest word u such that w has a factor $u^R vu$ with $|uv| \leq \alpha|u|$ (the longest α-gapped palindrome of w).

In this paper, we present a solution of Problem 1, working in $\mathcal{O}(\alpha n)$ time, and explain briefly how our algorithms can be adapted to solve Problem 2 within the same time complexity Our approach is more efficient than producing first the list of all maximal α-gapped repeats or palindromes and then returning the one with the longest arms; the algorithms of [13] (for repeats) and [12] (for palindromes), which produce such lists, are slower by an α factor than ours. Our solutions are based on a careful combinatorial analysis (e.g., in each solution we find separately the repeats or palindromes with periodic arms and those with aperiodic arms, respectively) as well as on the usage of several word-processing data-structures (e.g., suffix arrays, dictionary of basic factors). They are essentially different from the approaches of [2] (which is based on an efficient processing of the suffix tree of the input word) and from those in [12–14] (which have as crucial idea the construction and analysis of the LZ-factorisation of the word).

This extended abstract is structured as follows. After giving a series of basic facts regarding combinatorics on words and data structures, we develop the tools we need in our solutions. We then give a solution for Problem 1 and briefly point how it can be adapted to solve Problem 2.

2 Preliminaries

Definitions. The computational model we use to design and analyse our algorithms is the standard unit-cost RAM (Random Access Machine) with logarithmic word size, which is generally used in the analysis of algorithms. In the upcoming algorithmic problems, we assume that the words we process are sequences of integers (called letters, for simplicity). In general, if the input word has length n then we assume its letters are in $\{1, \ldots, n\}$, so each letter fits in a single memory-word. This is a common assumption in stringology (see, e.g., the discussion in [9]). Also, all logarithms appearing here are in base 2; we denote by $\log n$ the value $\lfloor \log_2 n \rfloor$. While we do not assume to be able to compute the value of $\log x$ in constant time, for some $x \leq n$, we note that one can compute in $\mathcal{O}(n)$ time all the values $\log x$ with $x \leq n$; therefore, we assume that in the algorithms addressed by Remark 2, or Lemmas 6 and 7, on an input of length n, we implicitly compute all the values $\log x$ with $x \leq n$.

Let V be a finite alphabet; V^* is the set of all finite words over V. The *length* of a word $w \in V^*$ is denoted by $|w|$. The *empty word* is denoted by λ. A word $u \in V^*$ is a *factor* of $v \in V^*$ if $v = xuy$, for some $x, y \in V^*$; we say that u is a *prefix* of v, if $x = \lambda$, and a *suffix* of v, if $y = \lambda$. We denote by $w[i]$ the symbol at position i in w, and by $w[i..j]$ the factor of w starting at position i and ending at position j, consisting of the catenation of the symbols $w[i], \ldots, w[j]$, where $1 \leq i \leq j \leq n$; we define $w[i..j] = \lambda$ if $i > j$. The powers of a word w are defined recursively by $w^0 = \lambda$ and $w^n = ww^{n-1}$ for $n \geq 1$. If w cannot be expressed as a nontrivial power (i.e., w is not a repetition) of another word, then w is *primitive*. A *period* of a word w over V is a positive integer p such that $w[i] = w[j]$ for all i and j with $i \equiv j \pmod{p}$; if p is a period of w, then w is called p-periodic. Let $per(w)$ be the smallest period of w. A word w with $per(w) \leq \frac{|w|}{2}$ is called run; a run $w[i..j]$ (so, $p = per(w[i..j]) \leq \frac{j-i+1}{2}$) is maximal if and only if it cannot be extended to the left or right to get a word with period p, i.e., $i = 1$ or $w[i-1] \neq w[i+p-1]$, and, $j = n$ or $w[j+1] \neq w[j-p+1]$. In [11] it is shown that the number of maximal runs of a word is linear and their list (with a run $w[i..j]$ represented as the triple $(i, j, per(w[i..j]))$ can be computed in linear time (see [1] for an algorithm constructing all maximal runs of a word, without producing its LZ-factorisation).

We now give the basic definitions of the data structures we use. For a word u, $|u| = n$, over $V \subseteq \{1, \ldots, n\}$ we build in $\mathcal{O}(n)$ time its suffix tree and suffix array, as well as LCP-data structures, allowing us to retrieve in constant time the length of the longest common prefix of any two suffixes $u[i..n]$ and $u[j..n]$ of u, denoted $LCP_u(i, j)$ (the subscript u is omitted when there is no danger of confusion). See, e.g., [8,9], and the references therein.

Note that, given a word w of length n and $\ell < n$ we can use one LCP query to compute the longest prefix w' of w which is ℓ-periodic: $|w'| = \ell + LCP(1, \ell+1)$. In our solution for Problem 2, we construct LCP data structures for the word $v = ww^R$; this takes $\mathcal{O}(|w|)$ time. To check whether $w[i..j]$ occurs at position ℓ in w (respectively, $w[i..j]^R$ occurs at position ℓ in w) we check whether $\ell+(j-i) \leq n$ and $LCP(i, \ell) \geq j - i + 1$ (respectively, $LCP(\ell, 2|w| - j + 1) \geq j - i + 1$).

Given a word w, its dictionary of basic factors (introduced in [5]) is a structure that labels the factors of the form $w[i..i + 2^k - 1]$ (called basic factors), for $k \geq 0$ and $1 \leq i \leq n - 2^k + 1$, such that every two equal basic factors get the same label and the label of a basic factor can be retrieved in $\mathcal{O}(1)$ time. The dictionary of basic factors of a word of length n is constructed in $\mathcal{O}(n \log n)$ time.

Preliminary Results and Basic Tools. In the following we introduce our basic tools, of both algorithmic and combinatorial nature.

The first lemma concerns overlapping maximal runs in a word.

Lemma 1. *The overlap of two maximal runs u and u' with the same period (i.e., per(u)=per(u')) is shorter than the period.*

A consequence of this lemma is that no position of w is contained in more than two maximal runs having the same period. Also, for $\ell, p \geq 1$, a factor of length ℓp of w contains at most $\ell - 1$ maximal runs of period p; this holds because

the overlap of each two consecutive (when ordered w.r.t. their starting position) runs of that factor is shorter than $p-1$.

The following lemma can be shown using standard tools.

Lemma 2. *Let w be a word of length n. We can preprocess the word w in $\mathcal{O}(n \log n)$ time to construct data structures that allow us to answer in $\mathcal{O}(\log n)$ time queries asking for the length of the period of the factors of w.*

For a word u, let $lper(u)$ be the lexicographically minimal factor x of u of length equal to $per(u)$; clearly, $lper(u)$ is primitive for all u. For a word w and a factor x of w, let L_x be the list of the maximal runs $v = w[i..j]$ of w such that $lper(v) = x$, ordered with respect to the starting positions of these runs.

Lemma 3. *Let w be a word of length n. We can compute in $\mathcal{O}(n)$ time the lists L_x for all $x \in H_w$, where H_w is the set of the factors x of w with $L_x \neq \emptyset$.*

During the computation of the lists L_x we can also compute for each $v \in L_x$ the position where x occurs firstly in v. Accordingly, each list L_x is stored in a structure where it is identified by the starting position of the first occurrence of x in the first run of L_x and by $|x|$. However, for simplicity, we keep writing L_x.

Remark 1. Assume that w is a word and let v be a factor of w with $per(v) = p \leq \frac{|v|}{2}$. Let z be a factor of length $\ell|v|$ of w. Each occurrence of v in z is part of a maximal run of period p; moreover, if v occurs in a maximal run of period p at the position i of that run, it will also occur at positions $i-p$ and $i+p$, provided that $i-p$ and $i+p+|v|-1$ fall inside the run, respectively. So, we can represent succinctly the occurrences of v in z by returning succinct representations of the maximal runs u contained in z, of length at least $|v|$, with $lper(v) = lper(u)$; for each of these runs we also store the first occurrence of v in the run. Finally, note that there are at most $2\ell - 1$ such runs contained in z.

Similarly, for a word w of length n, we note that a basic factor $w[i..i+2^k-1]$ occurs either at most twice in a factor $w[j..j+2^{k+1}-1]$ or its occurrences are part of a run of period $per(w[i..i+2^k-1]) \leq 2^{k-1}$ (so, the positions where $w[i..i+2^k-1]$ occurs in $w[j..j+2^{k+1}-1]$ form an arithmetic progression of ratio $per(w[i..i+2^k-1])$, see [10]). Hence, the occurrences of $w[i..i+2^k-1]$ in $w[j..j+2^{k+1}-1]$ can be represented in a compact manner, just like before. To the same end, for an integer $c \geq 2$, the occurrences of the basic factor $w[i..i+2^k-1]$ in $w[j..j+c2^k-1]$ can be presented in a compact manner: the positions (at most c) of the separate occurrences of $w[i..i+2^k-1]$ (that is, occurrences that do not form a run) and/or at most c maximal runs determined by the overlapping occurrences of $w[i..i+2^k-1]$. We can show the following result.

Lemma 4. *Given a word w of length n and a number $c \geq 2$, we can preprocess w in time $\mathcal{O}(n \log n)$ such that given any basic factor $y = w[i..i+2^k-1]$ and any factor $z = w[j..j+c2^k-1]$, with $k \geq 0$, we can compute in $\mathcal{O}(\log\log n + c)$ time a succinct representation of all the occurrences of y in z.*

The following results are shown using tools developed in [7].

Lemma 5. *Given a word v, $|v| = \alpha \log n$, we can process v in time $\mathcal{O}(\alpha \log n)$ time such that given any basic factor $y = v[j \cdot 2^k + 1..(j+1)2^k]$, with $j, k \geq 0$ and $j2^k + 1 > (\alpha - 1)\log n$, we can find in $\mathcal{O}(\alpha)$ time $\mathcal{O}(\alpha)$ bit sets, each storing $\mathcal{O}(\log n)$ bits, characterising all the occurrences of y in v.*

Note that each of the bit sets produced in the above lemma can be stored in a constant number of memory words in our model of computation. Essentially, this lemma states that we can obtain in $\mathcal{O}(\alpha \log n)$ time a representation of size $\mathcal{O}(\alpha)$ of all the occurrences of y in v.

Remark 2. By the previous lemma, given a word v, $|v| = \alpha \log n$, and a basic factor $y = v[j \cdot 2^k + 1..(j+1)2^k]$, with $j, k \geq 0$ and $j2^k + 1 > (\alpha - 1)\log n$, we can produce $\mathcal{O}(\alpha)$ bit sets, each containing exactly $\mathcal{O}(\log n)$ bits, characterising all the occurrences of y in v. Let us also assume that we have access to all values $\log x$ with $x \leq n$. Now, using the bit-sets encoding the occurrences of y in v and given a factor z of v, $|z| = c|y|$ for some $c \geq 1$, we can obtain in $\mathcal{O}(c)$ time the occurrences of y in z: the positions (at most c) where y occurs outside a run and/or at most c maximal runs containing the occurrences of y. Indeed, the main idea is to select by bitwise operations on the bit-sets encoding the factors of v that overlap z the positions where y occurs (so the positions with an 1). For each two consecutive such occurrences of y we detect whether they are part of a run in v (by *LCP*-queries on v) and then skip over all the occurrences of y from that run (and the corresponding parts of the bit-sets) before looking again for the 1-bits in the bit-sets.

3 Our Solution

We now give the solution to Problem 1. Our solution has two major steps. In the first one, described in Lemma 6, we compute the longest α-gapped repeat uvu with u periodic, i.e., $per(u) \leq \frac{|u|}{2}$. In the second one, described in Lemma 7, we compute the longest α-gapped repeat uvu with u aperiodic, i.e., $per(u) > \frac{|u|}{2}$. Then, we return the one repeat of these two that has the longest arms.

We first show how to find the longest α-gapped uvu repeat with u periodic.

Lemma 6. *Given a word w of length n and $\alpha \leq n$, the longest α-gapped repeat uvu with u periodic contained in w can be found in $\mathcal{O}(\alpha n)$ time.*

Proof. For the simplicity of the exposure, let $c = \alpha - 1$. We want to find a repeat $u_1vu_2 = uvu$ with $u_1 = u_2 = u$ periodic and $|v| \leq c|u|$. As u is periodic, both its occurrences u_1 and u_2 must be contained in maximal runs of w having the same *lper*. Accordingly, let y_1 and y_2 denote the maximal runs containing u_1 and u_2; we have, $lper(y_1) = lper(y_2) = x$.

The outline of our algorithm is the following. For each x, we have three cases to analyse. In the first y_1 and y_2 denote the same run, and in the second y_1 and y_2 are overlapping runs with the same period. In both cases, we can easily find the α-gapped repeat u_1vu_2 with u_1 contained in y_1 and u_2 contained in y_2.

The last case occurs when y_1 and y_2 do not overlap. Then, it is harder to find the repeat u_1vu_2 we look for. In this case, we try each possibility for y_2 and, at a very intuitive level, the restriction on the gap suggests that we do not have to look for y_1 among too many runs occurring to the left of y_2, and having the same *lper* as y_2: either they are not long enough to be worth considering as a possible place for the left arm of the longest α-gapped repeat with the right arm in y_2 or the gap between them and y_2 is too large. We exploit this and are able to find efficiently the repeat u_1vu_2 with the longest arms. With care, the analysis of the first two cases takes constant time per run, respectively, per pair of overlapping runs, while in the last case, for each run y_2 we can check whether it is the place of the right arm of the repeat we look for in amortised $\mathcal{O}(c)$ time. This adds up to $\mathcal{O}(cn) = \mathcal{O}(\alpha n)$ time. We now give more details for each case.

In the first case (see Fig. 1), u_1 and u_2 occur in the same run, so $y_1 = y_2 = y$. Let us first assume that in u_1vu_2 we have $v \neq \lambda$. Then u_1 must be a prefix of y and u_2 a suffix of y. Otherwise, we could extend both u_1 and u_2 with at least one symbol to get an α-gapped repeat with longer arm. For instance, when u_1 is not a prefix of y, then we extend both u_1 and u_2 with a letter to the left, the last letter of v becoming now part of the new u_2. This contradicts the fact that u_1vu_2 was the longest α-gapped repeat. Now, if $y = t^\ell t'$ is the maximal run, we get that $u = t^{\ell'}t'$ where $\ell' = \left\lfloor \frac{\ell-1}{2} \right\rfloor$. Further, assume that $v = \lambda$, so u_1vu_2 is in fact a square uu. Like before, we have that uu is either the longest square prefix of y or its longest square suffix; these can be immediately computed, and we just return the longest of them. This way, we can compute the longest α-gapped repeat uvu with both occurrences of u contained in the same maximal run y of w.

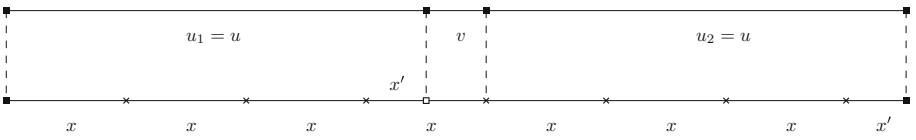

Fig. 1. An α-gapped repeat $u_1vu_2 = uvu$ inside a run $x^k x'$.

In the second case (see Fig. 2), y_1 and y_2 are distinct runs that overlap. By Lemma 1, the length of the overlap between these two runs is at most $|x|$; also, y_1 and y_2 should occur on consecutive positions in the list L_x. We first assume that $v \neq \lambda$. If u_1 is not a prefix of y_1, then u_2 should be a prefix of y_2, or we could extend both u_1 and u_2 to the left and get a longer α-gapped repeat (with a shorter gap v). So assume that u_2 is a prefix of y_2, and let x' be the prefix of length $|x|$ of y_2. It is clear that u_2 is obtained by taking the longest word of period $|x|$ that starts with x' occurring both in y_2 and in y_1, such that it ends in y_1 before the position where y_2 starts. From the computation of L_x we have the first occurrence of x in both y_1 and y_2, so we can also obtain the first occurrence of x' in y_1; consequently, we can also compute the aforementioned longest word. This concludes the case when u_1 is not a prefix of y_1. The case when u_1 is a prefix of y_1 can be treated in a very similar fashion. This way, we get the longest

α-gapped repeat u_1vu_2 with u_1 contained in y_1, u_2 contained in y_2, and $v \neq \lambda$. Further, we consider the case when $v = \lambda$, so $u_1u_2 = uu$ is a square. If neither u_1 nor u_2 is a prefix of y_1 and y_2, respectively, then we can shift the entire square u_1u_2 to the left with one symbol, to get a new square with the same length of the arm. We can repeat this process until one of u_1 or u_2 becomes a prefix of its corresponding maximal run. Now, just like before, we can determine the longest square uu such that either uu starts on the same position as y_1 or the second u starts on the same position as y_2, and this gives us one of the longest squares uu such that the first u is contained in y_1 and the second in y_2. Hence, we obtained the longest α-gapped repeat u_1u_2 with u_1 contained in y_1 and u_2 in y_2, where y_1 and y_2 overlap.

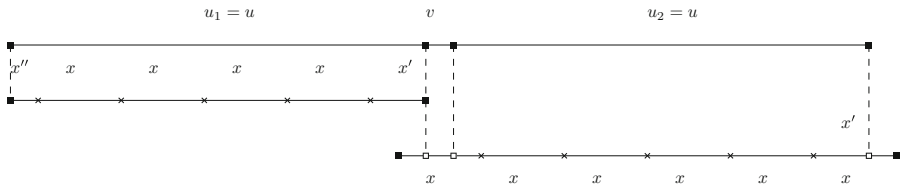

Fig. 2. An α-gapped repeat $u_1vu_2 = uvu$ with u_1 in $y_1 = x''x^{k_1}x'$, u_2 in $y_2 = x^{k_2}$, and y_1 and y_2 overlap.

In the final case (see Fig. 3), the runs y_1 and y_2 do not overlap. We first analyse the ways the arms of the longest α-gapped repeat uvu, namely u_1 and u_2, may occur in the runs y_1 and y_2, respectively.

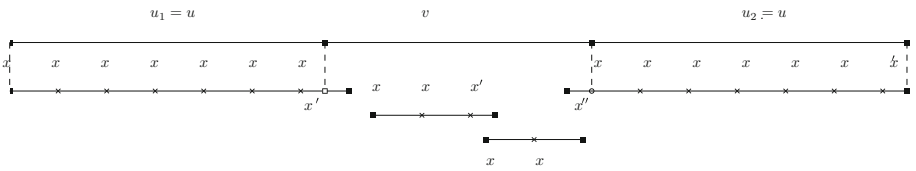

Fig. 3. An α-gapped repeat $u_1vu_2 = uvu$ with u_1 in $y_1 = x^{k_1}$ and u_2 in $y_2 = x''x^{k_2}x'$. Between y_1 and y_2 there are some runs with the same *lper*: $y_3 = x^2x'$, $y_4 = x^2$.

Firstly, assume that y_1 equals some factor of y_2. Then, either $u_1 = y_1 (= u_2)$ or u_1 is a suffix of y_1 and u_2 is a prefix of y_2. Indeed, if we cannot construct an α-gapped repeat u_1vu_2 with $u_1 = u_2 = y_1$ because the gap is too long, then any repeat u_1vu_2 with u_1 prefix of y_1 and u_2 suffix of y_2 is not α-gapped either. Moreover, unless $u_1 = y_1$, then u_1 should be a suffix of y_1 and u_2 a prefix of y_2; otherwise, a longer repeat could be constructed. Finally, if y_1 equals a factor of y_2 but we cannot construct an α-gapped repeat u_1vu_2 as above with $u_1 = u_2 = y_1$, then we will not be able to construct an α-gapped repeat u_1vu_2 with u_1 being the run y_1 and u_2 contained in some run occurring to the right of y_2.

Secondly, the case when y_2 equals a factor of y_1 is similar. Either $u_1 = u_2 = y_2$ or u_1 is a suffix of y_1 and u_2 is a prefix of y_2. Moreover, if y_2 is contained in y_1 but we cannot construct an α-gapped repeat $u_1 v u_2$ as above with $u_1 = u_2 = y_2$, then we will not be able to construct an α-gapped repeat $u_1' v u_2$ with u_2 being part of the run y_2 and $u_1' = y_2$ contained in any run occurring to the left of y_1.

Finally, assume that neither y_1 equals a factor of y_2, nor y_2 a factor of y_1. This means that the difference between the $|y_1|$ and $|y_2|$ is rather small; anyway, smaller than $|x|$. Now, there are two possibilities. The first one is when u_1 is a prefix of y_1 with $|y_1| - |u_1| < |x|$ and u_2 a suffix of y_2 with $|y_2| - |u_2| < |x|$. The second one is when u_1 is a suffix of y_1 with $|y_1| - |u_1| < |x|$ and u_2 a prefix of y_2 with $|y_2| - |u_2| < |x|$. In both cases, we get $\max\{|y_1|, |y_2|\} \leq |u| + |x|$ and $\min\{|y_1|, |y_2|\} \geq |u|$.

Now we can describe how to compute the longest α-gapped repeat $u_1 v u_2$, with u_1 and u_2 occurring in non-overlapping runs. Unlike the previous case, here it may be the case that y_1 and y_2 are not consecutive runs of L_x. So, we consider each run $y_2 \in L_x$ at a time. For the current y_2 we maintain a set M of possible runs that may contain the left arm u_1 of the longest α-gapped repeat. The main point of our solution is that, at each step, the current M (corresponding to the current run y_2) is computed efficiently from the set M corresponding to the run of L_x previously considered in the role of y_2. Then, we use M to find the longest α-gapped repeat we can construct with the right arm in y_2, and, finally, eliminate from M the runs that are useless for the rest of the computation. That is, when we reached y_2, we know already that some of the runs occurring before it are too short to be able to contain the left arm of the longest α-gapped repeat; also, some runs may not be too short to produce the repeat with the longest arms, but are not long enough to ensure that any repeat with its left arm in them and the right arm in one of the runs occurring to the right of y_2 would be α-gapped. These runs should not be considered further in the computation, so they are discarded. We execute this process for all $y_2 \in L_x$.

More precisely, our algorithm runs as follows. We assume that before analysing $y_2 \in L_x$ we found the longest α-gapped repeat $u_1 v u_2$, with both u_1 and u_2 contained in two non-overlapping runs which occur before y_2 in L_x, so we processed them already. Let $\ell = |u_1|$ and $k = \left\lfloor \frac{\ell}{|x|} \right\rfloor$. We now consider the run $y_2 \in L_x$, and assume that $|y_2| > \ell$; clearly, shorter runs could not contain the right arm of a repeat with arms longer than u_1. By the same reason, we do not have to consider any run that contains at most $k-2$ full occurrences of x as a place for the left arm of the repeat: the length of this arm would be less than $k|x| \leq \ell$. Also, we do not have to consider a run y_1 with at most $k+1$ occurrences of x as a possible place for the left arm of the repeat unless it is one of the rightmost $6c$ runs occurring in L_x that end before y_2 begins and have at least $k-1$ occurrences of x. Indeed, if y_1 is not one of these $6c$ runs, then the gap between the end of y_1 and the start of y_2 would be, if $k \geq 3$, longer than $6c(k-2)|x|$, or, if $k = 2$, longer than $6c|x|$. This is greater than $(k+3)c|x| \geq c|y_1|$. Therefore, any repeat with the left arm in such a run y_1 and right arm in the considered run y_2 would not be α-gapped.

So, we only need to consider for some y_2 the rightmost $6c$ runs of L_x which end before y_2 and have at least $k-1$ occurrences of x as well as the runs occurring before (i.e., to the left of) these $6c$ runs, that contain at least $k+2$ occurrences of x. These runs are stored in the structure M. First, we check which is the α-gapped repeat with longest arm that can be constructed with the left arm in one of the aforementioned $6c$ runs and the right one in y_2. Then we try to do the same thing for the other runs of M. Let y_1 be a run with at least $k+2$ occurrences of x.

It might be the case that y_1 equals a factor of y_2. We produce the longest α-gapped repeat with left arm in y_1 and right arm in y_2. In the case when this repeat is y_1vy_1, we store it if its arm y_1 is longer than the arm of the current longest α-gapped repeat, and then discard y_1: we already obtained the longest repeat whose left arm could be in y_1, so it bears no importance for the rest of the computation. We then check the next run that occurs to the left of y_1 and is contained in M, and so on. If we cannot obtain y_1vy_1, we discard y_1 because it will never lead to another α-gapped repeat again, as explained in the discussion on how an α-gapped repeat with arms in non-overlapping runs may be placed inside these runs, and, again, continue our search. If y_2 is contained in y_1, we either obtain the α-gapped repeat y_2vy_2 and stop analysing y_2, as it already produced the longest α-gapped repeat it can ever produce, or we obtain only a shorter α-gapped repeat and we stop analysing y_2 because it will never produce another α-gapped repeat with a run occurring to the left of y_1: the rest of the runs in M are simply to far away from y_2.

If neither y_1 equals a factor of y_2 nor y_2 equals a factor of y_1, we proceed as follows. Let u_1vu_2 be the longest α-gapped repeat with the arms contained in y_1 and y_2, respectively. Assume that $\left\lceil \frac{|u|}{|x|} \right\rceil = m$; then $\max\{|y_1|, |y_2|\} \leq (m+1)|x|$ and $\min\{|y_1|, |y_2|\} \geq (m-1)|x|$. It follows that we do not have to consider many more runs to the left of y_1 as a possible place for the left arm of the longest α-gapped repeat with the right arm in y_2. Indeed, it is enough to consider, besides y_1, the next (rightmost) $8c$ runs occurring to the left of y_1 in L_x and having at least $m-2$ occurrences of x. All the other runs we meet while traversing L_x to select these $8c$ runs are discarded: they are too short to produce a repeat with a longer arm than what we already have found. The runs occurring to the left of these $8c$ runs could not produce an α-gapped repeat with the right arm in y_2, as the gap between them and y_2 would too long. Indeed, the gap between any of them and y_2 is at least $8c(m-3)|x| + 6c|x|$, if $m \geq 4$, or $14c|x|$, otherwise; this is, anyway, greater than $c|y_2|$. Therefore, after checking the selected $8c$ runs as a possible place for the left arm of the longest α-gapped repeat, we are sure to have obtained the longest α-gapped repeat with the right arm in y_2.

Further, we have to update M so that it becomes ready to be used when considering a new run of L_x in the role of y_2. Assume the current longest α-gapped repeat has the arm length ℓ' and let $k' = \left\lfloor \frac{\ell'}{|x|} \right\rfloor$. From the runs of M (so, excluding the ones we already discarded), we first store the rightmost $6c$ runs with at least $k'-1$ occurrences of x. Then we also discard from the rest of M all the runs with at most $k'+1$ occurrences of x, occurring to the right of the

leftmost run of L_x considered in our search. Once M is cleaned up like this, if the next run we need to consider in the role of y_2 (so, which has length greater than ℓ') is y', we add to M, in the order of their starting positions, the runs with at least $k'-1$ occurrences of x and ending between the starting position of y_2 and the starting position of y'. These runs are added one by one, such that the set of the rightmost $6c$ runs of M with at least $k'-1$ occurrences of x is correctly maintained. When a run with at most $k'+1$ occurrences of x is no longer among the last $6c$ runs of M, we just discard it. We then process the updated M with y' in the role of y_2.

In this way, we compute for each x the longest α-gapped repeat u_1vu_2 with $u_1 = u_2$ periodic, contained in non-overlapping runs with $lper = x$. We do this for all possible values of x, and, alongside the analysis from the cases discussed at the beginning of this proof, we get the longest α-gapped repeat u_1vu_2 with $u_1 = u_2$ periodic.

The complexity of this algorithm is $\mathcal{O}(\alpha n)$. Indeed, in the first case we spend constant time per each run. In the second case we need constant time to analyse each pair of consecutive runs from the list L_x, for each x. This adds up to $\mathcal{O}(n)$ as the number of runs in a word is linear. In the last case, the time needed to process a run $y_2 \in L_x$ is proportional to the number of elements discarded from the structure M during this processing plus the number of elements inserted in M before considering the next run of L_x, to which we add an extra $\mathcal{O}(c) = \mathcal{O}(\alpha)$ processing time. As each run of L_x is added at most once in M, and then removed once, the total number of deletions and insertions we make is $\mathcal{O}(n)$. In total, the analysis of the third case takes, as claimed, $\mathcal{O}(\alpha n)$. This concludes our proof. □

Next we show how to find the longest α-gapped repeats uvu with u aperiodic.

Lemma 7. *Given a word w of length n and an integer $\alpha \leq n$, the longest α-gapped repeat uvu with u aperiodic, contained in w, can be found in $\mathcal{O}(\alpha n)$ time.*

Proof. Here we cannot use the runs structure of the input word to guide our search for the arms of the longest α-gapped repeat. So we need a new approach.

Informally, this new approach works as follows (see also Fig. 4). For each k, we try to find the longest α-gapped repeat $u_1vu_2 = uvu$, with $u_1 = u_2 = u$ aperiodic, and $2^{k+1} \log n \leq |u| \leq 2^{k+2} \log n$. In each such repeat, the right arm u_2 must contain a factor (called k-block) z, of length $2^k \log n$, starting on a position of the form $j2^k \log n + 1$. So, we try each such factor z, fixing in this way a range of the input word where u_2 could appear. Now, u_1 must also contain a copy of z. However, it is not mandatory that this copy of z occurs nicely aligned to its original occurrence; from our point of view, this means the copy of z does not necessarily occur on a position of the form $i \log n + 1$. But, it is not hard to see that z has a factor y of length $2^{k-1} \log n$, starting in its first $\log n$ positions and whose corresponding occurrence in u_1 should start on a position of the form $i \log n + 1$. Further, we can use the fact that u_1vu_2 is α-gapped and apply Lemma 4 to a suitable encoding of the input word to locate in constant time for each y starting in the first $\log n$ positions of z all possible occurrences of y on

a position of the form $i \log n + 1$, occurring not more than $(8\alpha + 2)|y|$ positions to the left of z. Intuitively, each occurrence of y found in this way fixes a range where u_1 might occur in w, such that $u_1 v u_2$ is α-gapped. So, around each such occurrence of y (supposedly, in the range corresponding to u_1) and around the y from the original occurrence of z we try to effectively construct the arms u_1 and u_2, respectively, and see if we get the α-gapped repeat. In the end, we just return the longest repeat we obtained, going through all the possible choices for z and the corresponding y's. We describe in the following an $\mathcal{O}(\alpha n)$ time implementation of this approach.

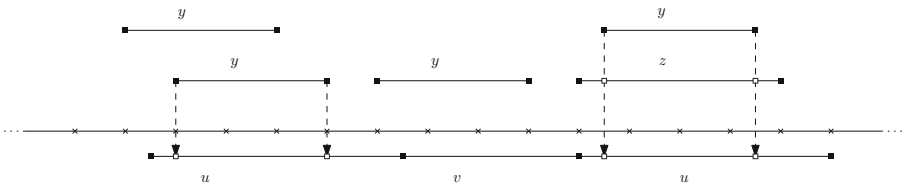

Fig. 4. Segment of w, split into blocks of length $\log n$. In this segment, z is a k-block of length $2^k \log n$. For each factor y, of length $2^{k-1} \log n$, occurring in the first $\log n$ symbols of z (not necessarily a sequence of blocks), we find the occurrences of y that correspond to sequences of 2^{k-1} blocks, and start at most $(8\alpha+2)|y| = (4\alpha+1) \cdot 2^k \log n$ symbols (or, alternatively, $(4\alpha + 1) \cdot 2^k$ blocks) to the left of the considered z. These y factors may appear as runs or as separate occurrences. Some of them can be extended to form an α-gapped repeat $u_1 v u_2 = uvu$ such that the respective occurrence of y has the same offset in u_1 as the initial y in u_2.

The first step of the algorithm is to construct a word w', of length $\frac{n}{\log n}$, whose symbols, called *blocks*, encode $\log n$ consecutive symbols of w grouped together. Basically, now we have two versions of the word w: the original one, and the one where it is split in blocks. Clearly, the blocks can be encoded into numbers between 1 and n in linear time, so we can construct in $\mathcal{O}(n)$ time the suffix arrays and *LCP*-data structures for both w and w'. We can also build in $\mathcal{O}(n)$ time the data structures of Lemma 4 for the word w'.

Now, we try to find the longest α-gapped repeat $u_1 v u_2 = uvu$ of w, with $u_1 = u_2 = u$ aperiodic, and $2^{k+1} \log n \leq |u| \leq 2^{k+2} \log n$, for each $k \geq 1$ if $\alpha > \log \log n$ or $k \geq \log \log n$, otherwise. Let us consider now one such k. We split again the word w, this time in factors of length $2^k \log n$, called *k-blocks*. For simplicity, assume that each split is exact.

Clearly, if an α-gapped repeat $u_1 v u_2$ like above exists, then u_2 contains at least one of the k-blocks. Consider such a k-block z and assume it is the leftmost k-block of u_2. On the other hand, u_1 contains at least $2^{k+1} - 1$ consecutive blocks from w', so there should be a factor y of w corresponding to 2^{k-1} of these $(2^{k+1} - 1)$ blocks which is also a factor of z, and starts on one of the first $\log n$ positions of z. Now, for each k-block z and each y, with $|y| = 2^{k-1} \log n$ and starting in its prefix of length $\log n$, we check whether there are occurrences of y in w ending before z that correspond to exactly 2^{k-1} consecutive blocks of

w' (one of them should be the occurrence of y in u_1); note that the occurrence of y in z may not necessarily correspond to a group of 2^{k-1} consecutive blocks, but the one from u_1 should. As u_1vu_2 is α-gapped and $|u_1| \leq 2^{k+2} \log n$, then the occurrence of y from u_1 starts at most $(4\alpha + 1)2^k \log n$ symbols before z (as $|u_1v| \leq \alpha|u_2| \leq \alpha 2^{k+2} \log n$, and z occurs with an offset of at most $2^k \log n$ symbols in u_2). So, the block-encoding of the occurrence of the factor y from the left arm u_1 should occur in a factor of $(4\alpha+1)2^k$ blocks of w', to the left of the blocks corresponding to z.

For the current z and an y as above, we check whether there exists a factor y' of w' whose blocks correspond to y, by binary searching the suffix array of w' (using LCP-queries on w to compare the factors of $\log n$ symbols of y and the blocks of w', at each step of the search). If not, we try another possible y. If yes, using Lemma 4 for w', we retrieve (in $\mathcal{O}(\log\log|w'| + \alpha)$ time) a representation of the occurrences of y' in the range of $(4\alpha + 1)2^k$ blocks of w' occurring before the blocks of z; this range corresponds to a range of length $(4\alpha+1)2^k \log n$ of w.

If y' is aperiodic then there are only $\mathcal{O}(\alpha)$ such occurrences. Each factor of w corresponding to one of these occurrences might be the occurrence of y from u_1, so we try to extend both the factor corresponding to the respective occurrence of y' from w' and the factor y from z in a similar way to the left and right to see whether we obtain the longest α-gapped repeat. If y' is periodic (so, y is periodic as well), Remark 1 shows that the representation of its occurrences consists of $\mathcal{O}(\alpha)$ separate occurrences and $\mathcal{O}(\alpha)$ runs in which y' occurs. The separate occurrences are treated as above. Each run r' of w' where y' occurs is treated differently, depending on whether its corresponding run r from w (made of the blocks corresponding to r') supposedly starts inside u_1, ends inside u_1, or both starts and ends inside u_1. We can check each of these three cases separately, each time trying to establish a correspondence between r and the run containing the occurrence of y from z, which should also start, end, or both start or end inside u_2, respectively. Then we define u_1 and u_2 as the longest equal factors containing these matching runs on matching positions. Hence, for each separate occurrence of y' or run of such occurrences, we may find an α-gapped repeat in w; we just store the longest. This whole process takes $\mathcal{O}(\alpha)$ time.

If $\alpha > \log\log n$, we run this algorithm for all $k \geq 1$ and find the longest α-gapped repeat uvu, with u aperiodic, and $4\log n \leq |u|$, in $\mathcal{O}(\alpha n)$ time.

If $\alpha \leq \log\log n$, we run this algorithm for all $k \geq \log\log n$ and find the longest α-gapped repeat uvu, with u aperiodic, and $2^{\log\log n+1} \log n \leq |u|$, in $\mathcal{O}(\alpha n)$ time. If our algorithm did not find such a repeat, we should look for α-gapped repeats with shorter arm. Now, $|u|$ is upper bounded by $2^{\log\log n+1} \log n = 2(\log n)^2$, so $|uvu| \leq \ell_0$, for $\ell_0 = \alpha \cdot 2(\log n)^2 + 2(\log n)^2 = (2\alpha + 2)(\log n)^2$. Such an α-gapped repeat uvu is, thus, contained in (at least) one factor of length $2\ell_0$ of w, starting on a position of the form $1 + m\ell_0$ for $m \geq 0$. So, we take the factors $w[1 + m\ell_0..(m + 2)\ell_0]$ of w, for $m \geq 0$, and apply for each such factor, separately, the same strategy as above. The total time needed to do that is $\mathcal{O}\left(\alpha\ell_0 \frac{n}{\ell_0}\right) = \mathcal{O}(\alpha n)$. Hence, we found the longest α-gapped repeats uvu, with u aperiodic, and $2^{\log\log(2\ell_0)+1} \log(2\ell_0) \leq |u|$. If our search was still fruitless, we

need to search α-gapped repeats with $|u| \leq 2^{\log\log(2\ell_0)+1}\log(2\ell_0) \leq 16\log n$ (a rough estimation, based on the fact that $\alpha \leq \log\log n$).

So, in both cases, $\alpha > \log\log n$ or $\alpha \leq \log\log n$, it is enough to find the longest α-gapped repeats with $|u| \leq 16\log n$. The right arm u_2 of such a repeat is contained in a factor $w[m\log n+1..(m+17)\log n]$ of w, while u_1 surely occurs in a factor $x = w[m\log n-16\alpha\log n+1..(m+17)\log n]$ (or, if $m\log n-16\alpha\log n+1 \leq 0$, then in a factor $x = w[1..(m+17)\log n]$); in total, there are $\mathcal{O}(n/\log n)$ such x factors. In each of these factors, we look for α-gapped repeats $u_1vu_2 = uvu$ with $2^{k+1} \leq |u| \leq 2^{k+2}$, where $0 \leq k \leq \log\log n + 2$ (the case $|u| < 2$ is trivial), and u_2 occurs in the suffix of length $17\log n$ of this factor. Moreover, u_2 contains a factor y of the form $x[j2^k + 1..(j+1)2^k]$. Using Lemma 5 and Remark 2, for each such possible y occurring in the suffix of length $17\log n$ of x, we assume it is the one contained in u_2 and we produce in $\mathcal{O}(\alpha)$ time a representation of the $\mathcal{O}(\alpha)$ occurrences of y in the factor of length $(4\alpha+1)|y|$ preceding y. One of these should be the occurrence of y from u_1. Similarly to the previous cases, we check in $\mathcal{O}(\alpha)$ time which is the longest α-gapped repeat obtained by pairing one of these occurrences to y, and extending them similarly to the left and right. The time needed for this is $\mathcal{O}(\alpha\log n)$ per each of the $\mathcal{O}(\frac{n}{\log n})$ factors x defined above. This adds up to an overall complexity of $\mathcal{O}(\alpha n)$, again.

This was the last case we needed to consider. In conclusion, we can find the longest α-gapped repeat uvu, with u aperiodic, in $\mathcal{O}(\alpha n)$ time. □

Lemmas 6 and 7 lead to the following theorem.

Theorem 1. *Problem 1 can be solved in $\mathcal{O}(\alpha n)$ time.*

To solve Problem 2 we construct *LCP*-structures for ww^R (allowing us to test efficiently whether a factor $w[i..j]^R$ occurs at some position ℓ in w) and a mapping connecting the lists L_x, of maximal runs y with $lper(y) = x$, to the lists L'_x of maximal runs y in w^R with $lper(y) = x$. Using the same strategy as in the case of repeats we can solve Problem 2 in $\mathcal{O}(\alpha n)$ time.

We first look for α-gapped palindromes with periodic arm, and then for such palindromes with aperiodic arms. The main difference is that, when looking for α-gapped palindrome u^Rvu with u contained in or containing a part of a run from L_x, for some x, we get that u^R is in contained in or contains part of a run from L'_x, respectively.

Basically, when we search u^Rvu with the longest periodic u and $|uv| \leq \alpha|u|$, we choose a maximal run y, with $lper(y) = x$, as the possible place for the right arm of the gapped palindrome; then, we only have to check (counting in an amortised setting) the rightmost $\mathcal{O}(\alpha)$ runs of L'_x that end before y as the possible place of u^R. When we search the longest α-gapped palindrome u^Rvu with u aperiodic, we split again w in blocks and k-blocks, for each $k \leq \log|w|$, to check in whether there exists such an u^Rvu with $2^k \leq |u| \leq 2^{k+1}$. This search is conducted pretty much as in the case of repeats, only that now when we fix some factor y of u, we have to look for the occurrences of y^R in the factor of length $\mathcal{O}(\alpha|y|)$ preceding it; the *LCP*-structures for ww^R are useful for this. The following results follows.

Theorem 2. *Problem 2 can be solved in $\mathcal{O}(\alpha n)$ time.*

References

1. Bannai, H., Tomohiro, I., Inenaga, S., Nakashima, Y., Takeda, M., Tsuruta, K.: A new characterization of maximal repetitions by lyndon trees. In: Proceedings of the SODA, pp. 562–571 (2015)
2. Brodal, G.S., Lyngsø, R.B., Pedersen, C.N.S., Stoye, J.: Finding maximal pairs with bounded gap. In: Crochemore, M., Paterson, M. (eds.) CPM 1999. LNCS, vol. 1645, pp. 134–149. Springer, Heidelberg (1999)
3. Crochemore, M.: An optimal algorithm for computing the repetitions in a word. Inf. Process. Lett. **12**(5), 244–250 (1981)
4. Crochemore, M., Iliopoulos, C.S., Kubica, M., Rytter, W., Waleń, T.: Efficient algorithms for two extensions of LPF table: the power of suffix arrays. In: van Leeuwen, J., Muscholl, A., Peleg, D., Pokorný, J., Rumpe, B. (eds.) SOFSEM 2010. LNCS, vol. 5901, pp. 296–307. Springer, Heidelberg (2010)
5. Crochemore, M., Rytter, W.: Usefulness of the Karp-Miller-Rosenberg algorithm in parallel computations on strings and arrays. Theoret. Comput. Sci. **88**(1), 59–82 (1991). http://dx.doi.org/10.1016/0304-3975(91)90073-B
6. Crochemore, M., Tischler, G.: Computing longest previous non-overlapping factors. Inf. Process. Lett. **111**(6), 291–295 (2011)
7. Gawrychowski, P.: Pattern matching in Lempel-Ziv compressed strings: fast, simple, and deterministic. In: Demetrescu, C., Halldórsson, M.M. (eds.) ESA 2011. LNCS, vol. 6942, pp. 421–432. Springer, Heidelberg (2011)
8. Gusfield, D.: Algorithms on strings, trees, and sequences: computer science and computational biology. Cambridge University Press, New York (1997)
9. Kärkkäinen, J., Sanders, P., Burkhardt, S.: Linear work suffix array construction. J. ACM **53**, 918–936 (2006)
10. Kociumaka, T., Radoszewski, J., Rytter, W., Waleń, T.: Efficient data structures for the factor periodicity problem. In: Calderón-Benavides, L., González-Caro, C., Chávez, E., Ziviani, N. (eds.) SPIRE 2012. LNCS, vol. 7608, pp. 284–294. Springer, Heidelberg (2012)
11. Kolpakov, R., Kucherov, G.: Finding maximal repetitions in a word in linear time. In: Proceedings of the FOCS, pp. 596–604 (1999)
12. Kolpakov, R., Kucherov, G.: Searching for gapped palindromes. Theor. Comput. Sci. **410**(51), 5365–5373 (2009)
13. Kolpakov, R., Podolskiy, M., Posypkin, M., Khrapov, N.: Searching of gapped repeats and subrepetitions in a word. In: Kulikov, A.S., Kuznetsov, S.O., Pevzner, P. (eds.) CPM 2014. LNCS, vol. 8486, pp. 212–221. Springer, Heidelberg (2014)
14. Kolpakov, R.M., Kucherov, G.: Finding repeats with fixed gap. In: Proceedings of the SPIRE, pp. 162–168 (2000)
15. Manacher, G.K.: A new linear-time on-line algorithm for finding the smallest initial palindrome of a string. J. ACM **22**(3), 346–351 (1975)