Linear-Time Computation of Local Periods

Jean-Pierre Duval [a]

[a] *LIFAR, Faculté des Sciences et Techniques, Université de Rouen, 76821 Mont-Saint-Aignan Cedex, France*

Roman Kolpakov [b,1]

[b] *Complexity Theory and Algorithmics Group, Department of Computer Science, The University of Liverpool, Chadwick Building, Peach Street, Liverpool L69 7ZF, UK*

Gregory Kucherov [c]

[c] *INRIA/LORIA, 615 rue du Jardin Botanique, B.P. 101, 54602 Villers-lès-Nancy, France*

Thierry Lecroq [d]

[d] *ABISS, Faculté des Sciences et Techniques, Université de Rouen, 76821 Mont-Saint-Aignan Cedex, France*

Arnaud Lefebvre [e]

[e] *UMR 6037/ABISS, Faculté des Sciences et Techniques, Université de Rouen, 76821 Mont-Saint-Aignan Cedex, France*

Abstract

We present a linear-time algorithm for computing *all* local periods of a given word. This subsumes (but is substantially more powerful than) the computation of the (global) period of the word and on the other hand, the computation of a critical factorization, implied by the Critical Factorization Theorem.

Key words: word, period, local period, algorithm, complexity, string matching

Email addresses: `Jean-Pierre.Duval@univ-rouen.fr` (Jean-Pierre Duval), `R.Kolpakov@csc.liv.ac.uk` (Roman Kolpakov), `Gregory.Kucherov@loria.fr` (Gregory Kucherov), `Thierry.Lecroq@univ-rouen.fr` (Thierry Lecroq), `Arnaud.Lefebvre@univ-rouen.fr` (Arnaud Lefebvre).

[1] On leave from the French-Russian Institute for Informatics and Applied Mathe-

1 Introduction

Periodicities in words have been classically studied in word combinatorics and are at the core of many fundamental results [Lot83,CK97,Lot02]. Besides, notions and techniques related to periodic structures in words find their applications in different areas: data compression [Sto88], molecular biology [Gus97], as well as for designing more efficient string search algorithms [GS83,CP91,CR95].

In this paper, we concentrate, from the algorithmic perspective, on the important notion of *local periods*, that characterize a local periodic structure at each location of the word [Duv98,DMR01]. In informal terms, the local period at a given position is the size of the smallest *square* centered at this position. The reader can refer to [FS98,GS04] for recent studies on squares also called tandem repeats in the context of computational biology. An importance of local periods is evidenced by the fundamental Critical Factorization Theorem [Lot83,CK97,Lot02] that asserts that there exists a position in the word (and a corresponding factorization), for which the local period is equal to the global period of the word. The Critical Factorization Theorem has many important consequences and has been recently subject of new research developments [MRS95].

Designing efficient algorithms for computing different periodic structures in words has been for a long time an active area of research. It is well-known that the (global) period of a word can be computed in linear time, using the Knuth-Morris-Pratt string matching method [KMP77,CR94] (see also [CG00]). On the other hand, in [CP91] it has been shown that a critical factorization of a word w can be constructed in linear time, by computing the maximal suffix of w for the lexicographical order and the maximal suffix for the reverse lexicographical order. In the same work, the factorization has then been used to design a new string matching algorithm. As another example, the Critical Factorization Theorem was used in [JJB96] to design an efficient approximation algorithm for the shortest superstring problem.

In this paper, we show how to compute *all* local periods in a word of length n in time $O(n)$ assuming an alphabet of constant size. This is substantially more powerful than linear-time computations of a critical factorization and of the global period: indeed, once all local periods have been computed, the global period is simply the maximum of all local periods, and each such maximal value corresponds to a distinct critical factorization.

Note that a great deal of work has been done on finding periodicities occurring in a word (see [KK99] for a survey). However, none of them allows to compute

matics, Moscow University, Russia.

all local periods in linear time. The reason is that most of those algorithms are intrinsically super-linear, which can be explained by the fact that they tend, explicitly or implicitly, to enumerate all squares in the word, the number of which can be super-linear. The closest result is the one of [Kos94] which claims a linear-time algorithm for finding, for each position i of the word, the smallest square starting at i. The approach is based on a sophisticated analysis of the suffix tree. The absence of a complete proof prevents the comprehension of the algorithm in full details; however, to the best of our understanding, this approach cannot be applied to finding local periods.

Here we design a linear-time algorithm for finding all local periods, based on several different string matching techniques. Some of those techniques (s-factorization, Main-Lorentz extension functions) have already been successfully used for several repetition finding problems [Cro83], [ML84], [Mai89], [KK99], [KK00] and [KK01]. In particular, in [KK99], it has been shown that all *maximal repetitions* can be found in linear time, providing an exhaustive information about the periodic structure of the word. However, here again, a direct application of this approach to finding local periods leads to a super-linear algorithm. We then propose a non-trivial modification of this approach, that allows to find a subclass of local periods in linear time.

2 Local periods: preliminaries

Consider a word $w = a_1 ... a_n$ over a finite alphabet. $|w|$ denotes the length of w, and w^R stands for the reverse of w, that is $a_n a_{n-1} \dots a_1$. $w[i..j]$, for $1 \leq i, j \leq n$, denotes the subword $a_i ... a_j$ provided that $i \leq j$, and the empty word otherwise. A position i in w is an integer number between 0 and n, associated with the factorization $w = uv$, where $|u| = i$.

A *square* s is a word of the form tt (i.e. a word of even length with two equal halves). t is called the *root* of s, and $|t|$ is called its *period*.

Definition 1 *Let $w = uv$, and $|u| = i$. We say that a non-empty square tt* is centered at position i of w *(or* matches w at central position i*) iff the following conditions hold:*

(i) t is a suffix of u, or u is a suffix of t,
(ii) t is a prefix of v, or v is a prefix of t.

In the case when t is a suffix of u and t is a prefix of v, we have a square occurring inside w. We call it an *internal square* (see Figure 1). If v is a proper prefix of t (respectively, u is a proper suffix of t), the square is called *right external* (respectively, *left external*) (see Figures 2 and 3).

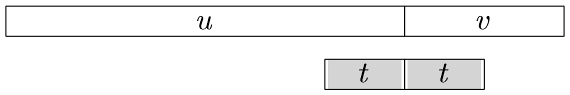

Fig. 1. Internal square

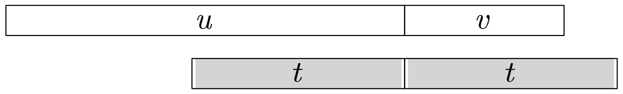

Fig. 2. Right external square

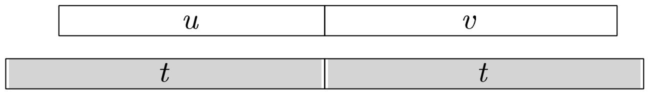

Fig. 3. Right and left external square

Definition 2 *The smallest non-empty square centered at a position i of w is called the* minimal local *square (hereafter simply* minimal*, for shortness). The* local period *at position i of w, denoted $LP_w(i)$, is the period of the minimal square centered at this position.*

Note that for each position i of w, $LP_w(i)$ is well-defined, and $1 \leq LP_w(i) \leq |w|$.

Any word w of length n has the *(global) period* $per(w)$, which is the minimal integer p such that $w[i] = w[i+p]$ whenever $1 \leq i, i+p \leq |w|$. Equivalently, $per(w)$ is the smallest positive integer p such that words $w[1..n-p]$ and $w[p+1..n]$ are equal. The critical factorization theorem is a fundamental result relating local and global periods [CV78,Duv79,Lot83,CK97,Lot02]:

Theorem 1 (Critical Factorization Theorem) *For each word w there exists a position i (and the corresponding factorization $w = uv$, $|u| = i$, $0 \leq i \leq |w|$) such that $LP_w(i) = per(w)$. Moreover, such a position exists among any $per(w)$ consecutive positions of w.*

Apart from its combinatorial consequences, an interesting feature of the critical factorization is that it can be computed very efficiently, in a time linear in the word length [CP91]. This can be done, for example, using the suffix tree construction [CR94]. On the other hand, it is well-known that the (global) period of a word can be computed in linear time, using, for example, the Knuth-Morris-Pratt technique [CR94] (see also [CG00]).

In this paper, we show how to compute *all* local periods in a word in linear time. This computation is much more powerful than that of a critical factorization or the global period: once all local periods are computed, the global period is equal to the maximum among them, and each such maximal local period corresponds to a critical factorization of the word.

The method we propose consists of two parts. We first show, in Section 3, how to compute all *internal* minimal squares. Then, in Section 4 we show how to compute left and right external minimal squares, in particular for those positions for which no internal square has been found. Both computations will be shown to be linear-time, and therefore computing all local periods can be done within linear time too.

3 Computing internal minimal squares

Finding internal minimal squares amounts to computing, for each position of the word, the smallest square centered at this position and occurring entirely inside the word, provided that such a square exists. Thus, throughout this section we will be considering only squares occurring inside the word and therefore, for the sake of brevity, omit the adjective "internal".

The problem of finding squares and, more generally, finding repetitions occurring in a given word has been studied for a long time in the string matching area, we refer to [KK99] for a survey. A natural idea is then to apply one of those methods in order to compute *all* squares and then select, for each central position, the smallest one. A direct application of this approach, however, cannot result in a linear-time algorithm, for the reason that the overall number of squares in a word can be as big as $\Theta(n \log n)$ (see [Cro81]). Therefore, manipulating the set of all squares explicitly is prohibitive for our purpose. In [KK99], *maximal repetitions* have been studied, which are maximally extended runs of consecutive squares. Importantly, the set of maximal repetitions encodes the whole set of squares, while being only of linear size.

Our approach here is to use the technique of computing maximal repetitions in order to retrieve squares which are minimal for some position. To present the algorithm in full details, we first need to describe the techniques used in [Mai89,KK99] for computing maximal repetitions.

3.1 s-factorization, Main-Lorentz extension functions, and computing repetitions

In this section we recall basic ideas, methods and tools underlying our approach.

The s-factorization [Cro83] is a special decomposition of the word. It is closely related to the Lempel-Ziv factorization (implicitly) defined by the well-known Lempel-Ziv compression method. The idea of defining the s-factorization is to proceed from left to right and to find, at each step, the longest factor which has another copy on the left. Alternatively, the Lempel-Ziv factorization considers the shortest factor which does not appear to the left (i.e. extends by one letter the longest factor previously defined). We refer to [Gus97] for a discussion on these two variants of factorization. A salient property of both factorizations is that they can be computed in linear time [RPE81] in the case of constant alphabet.

In their original definition, both of these factorizations allow an overlap between a factor and its left copy. However, we can restrict this and require the copy to be non-overlapping with the factor. This yields a *factorisation with non-overlapping copies* (see [KK01]). Computing the s-factorization (or Lempel-Ziv factorization) with non-overlapping copies can still be done in linear time.

In this work we will use the s-factorization with non-overlapping copies:

Definition 3 *The s-factorization of w with non-overlapping copies is the factorization $w = f_1 f_2 \dots f_m$, where f_i's are defined inductively as follows:*

(i) $f_1 = w[1]$,
(ii) assume we have computed $f_1 f_2 \dots f_{i-1}$ ($i \geq 2$), and let $w[j]$ be the letter immediately following $f_1 f_2 \dots f_{i-1}$ (i.e. $j = |f_1 f_2 \dots f_{i-1}| + 1$). If $w[j]$ does not occur in $f_1 f_2 \dots f_{i-1}$, then $f_i = w[j]$, otherwise f_i is the longest subword starting at position j, which has another occurrence in $f_1 f_2 \dots f_{i-1}$.

Note however that the choice of the factorization definition is guided by the simplicity of algorithm design and presentation clarity, and is not unique.

Our second tool is Main-Lorentz extension functions [ML84]. In its basic form, the underlying problem is the following. Assume we are given two words w_1, w_2 and we want to compute, for each position i of w_2, the longest prefix of w_1 which occurs at position i in w_2. This computation can be done in time $O(|w_1| + |w_2|)$ [ML84]. Note that w_1 and w_2 can be the same word, and that if we reverse w_1 and w_2, we come up with the symmetric computation of longest suffixes of $w_2[1..i]$ which are suffixes of w_1.

We now recall how Main-Lorentz extension functions are used for finding repetitions. The key idea is illustrated by the following problem. Assume we have two words $w_1 = w_1[1..m]$ and $w_2 = w_2[1..n]$ and consider their concatenation $w = w_1 w_2$. Assume we want to find all squares of w which cross the boundary between w_1 and w_2, i.e. squares which start at some position $\leq m$ and end at some position $> m$ in w (start and end positions of a square are the positions of respectively its first and last letter). First, we divide all such squares into two categories – those centered at a position $< m$ and those centered at a position $\geq m$ – and by symmetry, we concentrate on the squares centered at

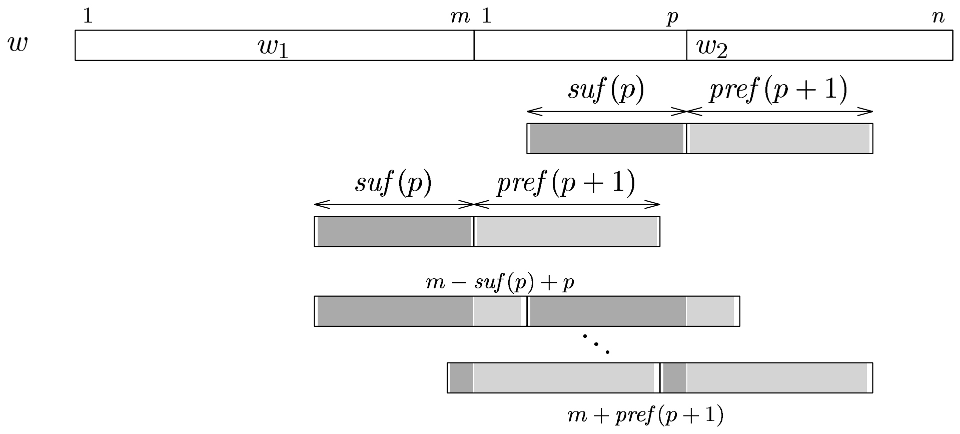

Fig. 4. If $\mathit{suf}(p)+\mathit{pref}(p+1) \geq p$ then there is a run of squares centered at positions from $m - \mathit{suf}(p) + p$ to $m + \mathit{pref}(p + 1)$.

a position $\geq m$ only. We then compute the following extension functions:

- $\mathit{pref}(i)$, $2 \leq i \leq n+1$ defined by $\mathit{pref}(i) = \max\{j \mid w_2[1..j] = w_2[i..i+j-1]\}$ for $2 \leq i \leq n$, and $\mathit{pref}(n + 1) = 0$,
- $\mathit{suf}(i)$, $1 \leq i \leq n$ defined by $\mathit{suf}(i) = \max\{j \mid w_1[m - j + 1..m] = w[m + i - j + 1..m + i]\}$.

Then there exists a square with period p iff

$$\mathit{suf}(p) + \mathit{pref}(p + 1) \geq p \qquad (1)$$

[Mai89]. This gives a key of the algorithm: we first compute values $\mathit{pref}(p)$ and $\mathit{suf}(p)$ for all possible p, which takes time $O(m + n)$. Then we simply check for each p inequality (1) – each time it is verified, we witness new squares of period p. More precisely, whenever the inequality is verified we have identified, in general, a series (*run*) of squares centered at each position from the interval $[m - \mathit{suf}(p) + p..m + \mathit{pref}(p + 1)]$ (see Figure 4). This run is a *maximal repetition* in w (see [KK99]). Formally, this maximal repetition may contain squares centered at positions $< m$ (if $\mathit{suf}(p) > p$), and squares starting at positions $> m$ (if $\mathit{pref}(p + 1) > p - 1$). Therefore, if we want only squares centered at positions $\geq m$ and starting at positions $\leq m$ (as it will be our case in the next Section), we have to restrict the interval of centers to $[\max\{m - \mathit{suf}(p) + p, m\}.. \min\{m + \mathit{pref}(p + 1), m + p\}]$. Clearly, verifying inequality (1) takes a constant time and the whole computation can be done in time $O(n)$.

To find, in linear time, all squares in a word (and not only those which cross a given position), we have to combine the factorization and extension function techniques. In general terms, the idea is the following: we compute the *s*-

factorization and process factors one-by-one from left to right. For each factor f_r, we consider separately those squares which occur completely inside f_r, and those ending in f_r and crossing the boundary with f_{r-1}. The squares of the first type are computed using the fact that f_r has a copy on the left – we can then retrieve those squares from this copy in time $O(|f_r|)$. The squares of the second type are computed using the extension function technique sketched above, together with an additional lemma asserting that those squares cannot extend to the left of f_r by more than $|f_r| + 2|f_{r-1}|$ letters [Mai89]. Therefore, finding all these squares, in form of runs, takes time $O(|f_{r-1}| + |f_r|)$. The whole word can then be processed in time $O(n)$. The reader is referred to [Mai89,KK99] for full details.

This general approach, initiated in [Cro83,Mai89] has been applied successfully to various repetition finding problems [KK99,KK00,KK01]. In this work we show that it can be also applied to obtain a linear-time algorithm for computing internal local periods. This gives yet another illustration of the power of the approach.

3.2 Finding internal minimal squares

We are now ready to present a linear-time algorithm for computing all internal minimal squares in a given word w.

First, we compute, in linear time, the s-factorization of w with non-overlapping copies and we keep, for each factor f_r, a reference to its non-overlapping left copy. The algorithm processes all factors from left to right and computes, for each factor f_r, all minimal squares ending in this factor. For each minimal square found, centered at position i, the corresponding value $LP_w(i)$ is set. After the whole word has been processed, positions i for which values $LP_w(i)$ have not been assigned are those positions for which no internal square centered at i exists. For those positions, minimal squares are external, and they will be computed at the second stage, presented in Section 4.

Let $f_r = w[m+1..m+\ell]$ be the current factor, and let $w[p+1..p+\ell]$ be its left copy (note that $p+\ell \leq m$). If for some position $m+i$, $1 \leq i < \ell$, the minimal square centered at $m+i$ occurs entirely inside the factor, that is $LP_w(m+i) \leq \min\{i, \ell - i\}$, then $LP_w(m+i) = LP_w(p+i)$. Note that $LP_w(p+i)$ has been computed before, as the minimal square centered at $p+i$ ends before the beginning of f_r. Based on this, we retrieve, in time $O(|f_r|)$, all values $LP_w(m+i)$ which correspond to squares occurring entirely inside f_r. It remains to find those values $LP_w(m+i)$ which correspond to minimal squares that end in f_r and extend to the left beyond the border between f_r and f_{r-1}.

To do this, we use the technique of computing squares described in the previous section. The idea is to compute all candidate squares and test which of them are minimal. However, this should be done carefully: as mentioned earlier, this can break down the linear time bound, because of a possible super-linear number of all squares. The main trick is to keep squares in runs and to show that there is only a linear number of individual squares which need to be tested for minimality.

As in [Mai89], we divide all squares under consideration into those which are centered inside f_r and those centered to the left of f_r. The two cases are symmetrical and therefore we concentrate on those squares centered inside f_r thus at positions $m..m + \ell - 1$. In addition, we are interested in squares starting at positions less than or equal to m and ending inside f_r. We compute all such squares *in the increasing order of periods*. For each $p = 1..\ell - 1$ we compute the run of all squares of period p centered at positions belonging to the interval $[m..m + \ell - 1]$, starting at a position less than or equal to m, and ending inside f_r, as explained in Section 3.1. Assume we have computed a run of such squares of period p, and assume that $q < p$ is the maximal period for which squares have been previously found. If $p \geq 2q$, then we check each square of the run whether it is minimal or not by checking the value $LP_w(i)$. If this square is not minimal, then its center i has been already assigned a value $LP_w(i)$. Indeed, if a smaller square centered at i exists, it has necessarily been already computed by the algorithm (recall that squares are computed in the increasing order of periods), and therefore a positive value $LP_w(i)$ has been set before. If no value $LP_w(i)$ has yet been assigned, then we have found the minimal square centered at i. Since there are at most p considered squares of period p (their centers belong to the interval $[m..m + p - 1]$), checking all of them takes at most $2(p - q)$ individual checks (as $q \leq p/2$ and $p - q \geq p/2$).

Now assume $p < 2q$. Consider a square $s_q = w[j - q + 1..j + q]$ of period q and center j, which has been previously found by the algorithm (square of period q in Figure 5). We now prove that we need to check for minimality only those squares s_p of period p which have their center k verifying one of the following inequalities:

$$|k - j| \leq p - q, \text{ or} \qquad (2)$$
$$k \geq j + q \qquad (3)$$

In words, k is located either within distance $p - q$ from j, or beyond the end of square s_q.

Lemma 1 *Let $s_p = w[k - p + 1..k + p]$ be the minimal square centered at some position k. Let $s_q = w[j - q + 1..j + q]$ be another square with $q < p$. Then one of inequalities (2),(3) holds.*

Proof By contradiction, assume that neither of them holds. Consider the case

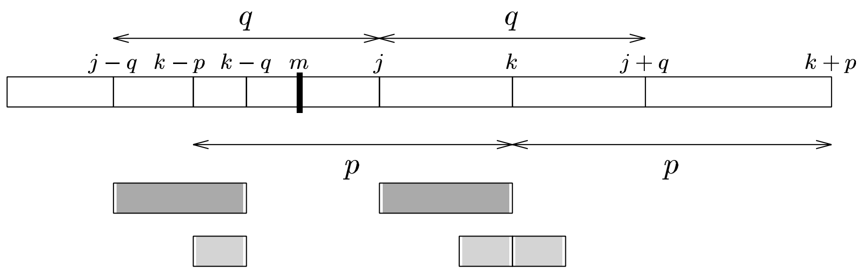

Fig. 5. Case where neither of inequalities (2),(3) holds (subcase $k > j$). m is the border between the factor f_r and the factor f_{r-1}.

$k > j$, case $k < j$ is symmetric. The situation with $k > j$ is shown in Figure 5.

Now observe that word $w[j+1..k]$ has a copy $w[j-q+1..k-q]$ (shown in dark gray in Figure 5) and that its length is $(k-j)$. Furthermore, since $k-j > p-q$ (as inequality (2) does not hold), this copy overlaps by $p-q$ letters with the left root of s_p. Consider this overlap $w[k-p+1..k-q]$ (light gray in Figure 5). It has a copy $w[k+1..k+(p-q)]$ and another copy $w[k-(p-q)+1..k]$ (see Figure 5). We thus have a smaller square centered at k, which proves that square s_p cannot be minimal. $\square$

Therefore, we need to check for minimality only those squares s_p which verify, with respect to s_q, one of inequalities (2),(3). Note that there are at most $2(p-q)$ squares s_p verifying (2), and at most $p-q$ squares s_p verifying (3), the latter because s_p must start before the current factor, i.e. $k \leq m+p$. We conclude that there are at most $3(p-q)$ squares of period p to check for minimality, among all squares found for period p. Summing up the number of all individual checks results in a telescopic sum, and we obtain that processing all squares centered in the current factor can be done in time $O(|f_r|)$.

A similar argument applies to the squares centered on the left of f_r. Note that after processing f_r, all minimal squares ending in f_r have been computed.

To summarize, we need to check for minimality only $O(|f_{r-1}|+|f_r|)$ squares, among those crossing the border between f_r and f_{r-1}, each check taking a constant time. We also need $O(|f_r|)$ time to compute minimal squares occurring inside f_r. Processing f_r takes then time $O(|f_{r-1}|+|f_r|)$ overall, and processing the whole word takes time $O(n)$.

Theorem 2 *In a word of length n, all internal minimal squares can be computed in time $O(n)$.*

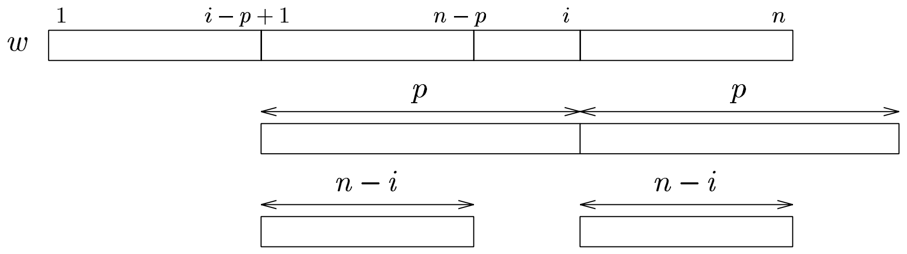

Fig. 6. If there is a right-external square of period p centered at position i then $suf(n-p) \geq n-i$ (case where $i > n-p$).

4 Computing external minimal squares

The algorithm of the previous section allows to compute all *internal* minimal squares of a word. Here we show how to compute *external* minimal squares for those positions which do not have internal squares centered at them.

Consider a word $w[1..n]$. We first consider squares which are right-external but not left-external. Those squares are centered at positions in the right half of the word. The case of squares which are left-external but not right-external is symmetrical.

For each position i in the right half of w, we compute a value $RS(i)$ equal to the minimal period of a right-external and not left-external square centered at i, provided such a square exists. We show that all values $RS(i)$ can be computed in linear time using longest extension functions that we already used in Section 3.2 for computing minimal internal squares.

Here we need the following longest extension function: For each position i of w, define $suf(i) = \max\{j \mid w[n-j+1..n] = w[i-j+1..i]\}$. This is the same function suf as the one defined in Section 3.1 except that it is defined on one word w instead of two words. Similar to Section 3.1, all values $suf(i)$ can be computed in linear time.

Consider now a right-external square of period p centered at some position $i \in [\lceil n/2 \rceil..n-1]$, where $n-i < p \leq i$. Observe that $w[i-p+1..n-p] = w[i+1..n]$. This implies that $suf(n-p) \geq n-i$ (see Figure 6). Conversely, if for some $p \in [1..n-1]$, $suf(n-p) > 0$, then there exists a family of squares of period p centered at positions $i \in [n - suf(n-p)..n-1]$ (see Figure 7). For $i > n-p$, the square is right-external, otherwise it is internal.

This implies the following algorithm for computing minimal right-external squares. Compute suf for all positions of w. For each $j \in [1..n]$, set $suf'(j) = suf(j)$ if $suf(j) < n-j$, and $suf'(j) = n-j-1$ otherwise. For each center position $i \in [\lceil n/2 \rceil..n-1]$, we need to compute the minimal p such that

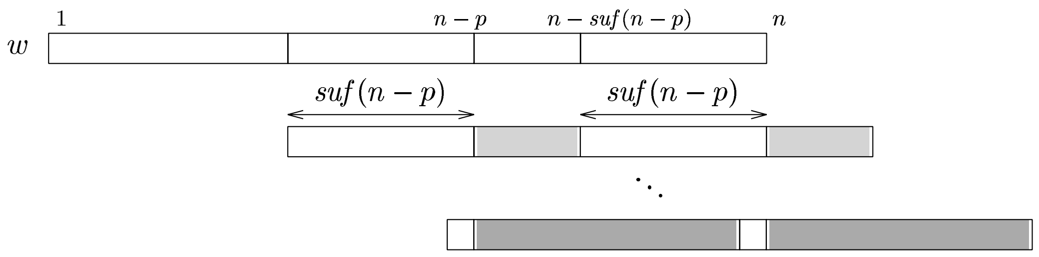

Fig. 7. If $suf(n-p) > 0$ then there is a run of square of period p centered at positions from $n - suf(n-p)$ to $n-1$.

$suf'(n-p) \geq n-i$.

Consider all pairs $(j, suf'(j))$ for $j \in [1..n]$. If for some pair $(j, suf'(j))$, there exists a pair $(j', suf'(j'))$ such that $suf'(j') \geq suf'(j)$ and $j' > j$, then $(j, suf'(j))$ carries no useful information for computing minimal right-external squares (see Figure 8). We then delete all such pairs $(j, suf'(j))$ from consideration by looping through all j from n to 1 and deleting those for which the value $suf'(j)$ is smaller than or equal to $\max_{j'>j}\{suf'(j')\}$. We then sort the remaining pairs $(j, suf'(j))$ in the decreasing order of $suf'(j)$. Using bucket sort, this can be done in $O(n)$ time and space.

We now set the values $RS(i)$ as follows. For the first element $(j_0, suf'(j_0))$ of the list, we set $RS(i) = 0$ for all $i \in [\lceil n/2 \rceil..n-suf'(j_0)-1]$. We then scan through the ordered list of pairs and for each element $(j, suf'(j))$, look at the next element $(j', suf'(j'))$, $suf'(j) > suf'(j')$. For all $i \in [n-suf'(j)..n-suf'(j')-1]$, set $RS(i) = n-j$. We then have the following

Lemma 2 *For each $i \in [\lceil n/2 \rceil..n-1]$, $RS(i)$ is the smallest period of a right-external square centered at i if such a square exists, and $RS(i) = 0$ otherwise.*

We now turn to squares that are both right-external and left-external. Consider such a square of period p, centered at some position i. Observe that $w[1..n-p] = w[p+1..n]$. Therefore, there exists a *border* of w of size $n-p < n/2$. The border of maximal size corresponds to right-external and left-external squares of minimal period. On the other hand, this period is equal to the (global) period $per(w)$ of w. Therefore, all minimal squares which are both right-external and left-external have the period equal to $per(w)$. On the other hand, it is well known that $per(w)$ can be easily computed in linear time [MP70,KMP77].

To conclude, all external minimal squares can be computed in time $O(n)$, for those positions which do not have internal squares centered in them. We then obtain an $O(n)$ algorithm for computing all minimal squares: first, using the algorithm of Section 3 we compute all internal minimal squares and then, using Lemma 2 and the above remark, we compute the minimal external squares for

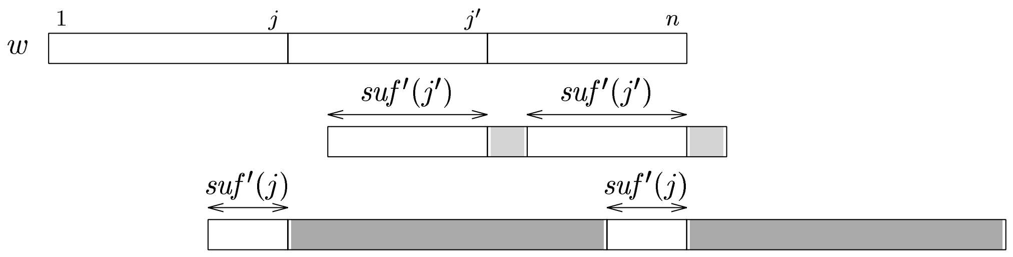

Fig. 8. If $j < j'$ and $suf'(j) \leq suf'(j')$ then $(j, suf'(j))$ is useless for computing minimal right-external squares.

those positions for which no internal square has been found at the first stage. This proves the main result.

Theorem 3 *In a word of length n, all local periods $LP_w(i)$ can be computed in time $O(n)$.*

5 Conclusions

We presented an algorithm that computes all local periods in a word in time linear in the length of the word. This computation provides an exhaustive information about the local periodic structure of the word. According to the Critical Factorization Theorem, the (global) period of the word is simply the maximum among all local periods. Therefore, as a case application, our algorithm allows to find *all* possible critical factorizations of the word.

The main difficulty was to extract all shortest local squares without having to process all individual squares occurring in the word, which would break down the linear time bound. This made impossible an off-the-shelf use of existing repetition-finding algorithms, and necessitated a non-trivial modification of existing methods.

An interesting research direction would be to study the combinatorics of possible sets of local periods, in a similar way as it was done for the structure of all (global) periods [GO81,RR01]. The results presented in this paper might provide an initial insight for such a study.

Acknowledgments GK, TL and AL have been supported by the french *Action Spécifique "Algorithmes et Séquences"* of *CNRS*. JPD, TL and AL have been supported by the NATO grant PST.CLG.977017. Part of this work has been done during the stay of RK at LORIA in summer 2002, supported by INRIA.

References

[CV78] Y. Cesari and M. Vincent. Une caractérisation des mots périodiques. *Comptes Rendus de l'Académie des Sciences Paris*, 286(A):1175-1177, 1978.

[CK97] C. Choffrut and J. Karhumäki. Combinatorics of words. In G. Rozenberg and A. Salomaa, editors, *Handbook on Formal Languages*, volume I, pages 329–438, Springer Verlag, Berlin-Heidelberg-New York, 1997.

[Cro81] M. Crochemore. An optimal algorithm for computing the repetitions in a word. *Information Processing Letters*, 12(5):244–250, 1981.

[Cro83] M. Crochemore. Recherche linéaire d'un carré dans un mot. *Comptes Rendus de l'Académie des Sciences Paris Série I Mathématiques*, 296:781–784, 1983.

[CP91] M. Crochemore and D. Perrin. Two-way string matching. *Journal of the ACM*, 38(3):651–675, 1991.

[CR94] M. Crochemore and W. Rytter. *Text algorithms.* Oxford University Press, 1994.

[CR95] M. Crochemore and W. Rytter. Squares, cubes, and time-space efficient string searching. *Algorithmica*, 13(5):405–425, 1995.

[CG00] A. Czumaj and L. Gąsieniec. On the complexity of determining the period of a string. In R. Giancarlo and D. Sankoff, editors, *Proceedings of the 11th Annual Symposium on Combinatorial Pattern Matching (CPM)*, Montréal, Canada, number 1848 in Lecture Notes in Computer Science, pages 412–422, Springer-Verlag, Berlin, 2000.

[Duv79] J.-P. Duval. Périodes et Répétitions des Mots du Monoide Libre. *Theoretical Computer Science*, 9:17–26, 1979.

[Duv98] J.-P. Duval. Périodes locales et propagation de périodes dans un mot. *Theoretical Computer Science*, 204(1-2):87–98, 1998.

[DMR01] J.-P. Duval, F. Mignosi and A. Restivo. Recurrence and periodicity in infinite words from local periods. *Theoretical Computer Science*, 262(1):269–284, 2001.

[FS98] A.S. Fraenkel, and J. Simpson. How many squares can a string contain ? *Journal of Combinatorial Theory,* Series A, 82:112–120, 1998.

[GS83] Z. Galil and J. Seiferas. Time-space optimal string matching. *Journal of Computer and System Sciences*, 26(3):280–294, 1983.

[GO81] L.J. Guibas and A.M. Odlyzko. Periods in strings. *Journal of Combinatorial Theory,* Series A, 30:19–42, 1981.

[Gus97] D. Gusfield. *Algorithms on Strings, Trees, and Sequences. Computer Science and Computational Biology.* Cambridge University Press, 1997.

[GS04] D. Gusfield, and J. Stoye. Linear-time algorithms for finding and representing all tandem repeats in a string. *Journal of Computer and System Sciences*, to appear.

[JJB96] T. Jiang, Z. Jiang and D. Breslauer. Rotation of periodic strings and short superstrings. In N. Ziviani, R. Baeza-Yates and K. Guimarães, editors, *Proceedings of the 3rd South American Workshop on String Processing (WSP)*, Recife, Brazil, pages 115–125, Carleton University Press, 1996.

[KMP77] D. E. Knuth, J. H. Morris, and V. R. Pratt. Fast pattern matching in strings. *SIAM Journal of Computing*, 6(2):323–350, 1977.

[KK99] R. Kolpakov and G. Kucherov. Finding maximal repetitions in a word in linear time. In *Proceedings of the 1999 Symposium on Foundations of Computer Science (FOCS)*, New York, pages 596–604, IEEE Computer Society, October 17-19 1999.

[KK00] R. Kolpakov and G. Kucherov. Finding repeats with fixed gap. In *Proceedings of the 7th International Symposium on String Processing and Information Retrieval (SPIRE)*, A Coruña, Spain, pages 162–168, IEEE Computer Society, Septembre 2000.

[KK01] R. Kolpakov and G. Kucherov. Finding Approximate Repetitions under Hamming Distance. In F. Meyer auf der Heide, editor, *Proceedings of the 9th European Symposium on Algorithms (ESA)*, Aarhus, Denmark, volume 2161 of Lecture Notes in Computer Science, pages 170–181, Springer-Verlag, Berlin, August 2001.

[Kos94] S. R. Kosaraju. Computation of squares in string. In M. Crochemore and D. Gusfield, editors, *Proceedings of the 5th Annual Symposium on Combinatorial Pattern Matching (CPM)*, Asilomar, California, number 807 in Lecture Notes in Computer Science, pages 146–150, Springer Verlag, Berlin, 1994.

[Lot83] M. Lothaire. *Combinatorics on Words*, volume 17 of *Encyclopedia of Mathematics and Its Applications.* Addison Wesley, 1983.

[Lot02] M. Lothaire. *Algebraic Combinatorics on Words.* Cambridge University Press, 2002.

[Mai89] M. G. Main. Detecting leftmost maximal periodicities. *Discrete Applied Mathematics*, 25(1-2):145–153, 1989.

[MP70] J. H. Morris, Jr and V. R. Pratt. A linear pattern-matching algorithm. Report, University of California, Berkeley, Number 40, 1970.

[ML84] M.G. Main and R.J. Lorentz. An $O(n \log n)$ algorithm for finding all repetitions in a string. *Journal of Algorithms*, 5(3):422–432, 1984.

[MRS95] F. Mignosi, A. Restivo and S. Salemi. A periodicity theorem on words and applications. In J. Wiedermann and P. Hájek, editors, *Proceedings of the 20th International Symposium on Mathematical Foundations of Computer*

Science (MFCS), Prague, Czech Republic, volume 969 of Lecture Notes in Computer Science, pages 337–348, Springer Verlag, Berlin, 1995.

[RR01] E. Rivals and S. Rahmann. Combinatorics of periods in strings. In J. van Leuween, P. Orejas and P.G. Spirakis, editors, *Proceedings of the 28th International Colloquium on Automata, Languages and Programming (ICALP)*, Heraklion, Crete, Greece, volume 2076 of Lecture Notes in Computer Science, pages 615–626, Springer Verlag, Berlin, 2001.

[RPE81] M. Rodeh, V.R. Pratt and S. Even. Linear algorithm for data compression via string matching. *Journal of the ACM*, 28(1):16–24, 1981.

[Sto88] J.A. Storer. *Data Compression: Methods and Theory.* Computer Science Press, Rockville, MD, 1988.