Algorithmica (1995) 13: 405-425

Algorithmica

Squares, Cubes, and Time–Space Efficient String Searching

M. Crochemore[1] and W. Rytter[2]

Abstract. We address several technical problems related to the time–space optimal string-matching algorithm of Galil and Seiferas (called the GS algorithm). This algorithm contains a parameter k on which the complexity depends and that originally satisfies $k \geq 4$. We show that $k = 3$ is the least integer for which the GS algorithm works. This value of the parameter k also minimizes the time of the search phase of the string-searching algorithm. With the parameter $k = 2$ we consider a simpler version of the algorithm working in linear time and logarithmic space. This algorithm is based on the following fact: any word of length n starts by less than $\log_\Phi n$ squares of primitive prefixes. Fibonacci words have a logarithmic number of square prefixes. Hence, the combinatorics of prefix squares and cubes is essential for string-matching with small memory.

We give a time–space optimal sequential computation of the period of a word based on the GS algorithm. The latter corrects the algorithm given in [GS2] for the computation of periods. We present an optimal parallel algorithm for pattern preprocessing. This paper also provides a cleaner version and a simpler analysis of the GS algorithm.

Key Words. Analysis of algorithms, Algorithms on strings, Pattern matching, String matching, Periods of words, Parallel algorithms

1. Introduction.

This paper discusses the string-matching problem: finding all the occurrences of a pattern within a text.

The algorithm of Galil and Seiferas [GS2] was the first time–space optimal algorithm for the string-searching problem, and its existence even disproved a conjecture stated by the authors [GS1]. The algorithm works in time linear in the size of input words and requires only a constant additional memory space. Later we refer to this algorithm as the GS algorithm. The basic parameter of the algorithm is an integer k. As is written on p. 281 of [GS2]: "the constant of proportionality in our algorithms' worst-case running times will be proportional to k, so there is a practical reason to keep k small." The GS algorithm was originally designed with $k \geq 4$. Recently, two other time–space optimal string-searching algorithms were discovered, see [CP] and [C]. However, despite this we believe that the GS algorithm is of so great importance that it deserves further investigation. In this paper we show that the GS algorithm also works with $k = 3$.

[1] Institut Gaspard Monge, Université de Marne la Vallée, 2 rue de la Butte Verte, F-93160 Noisy le Grand, France. Work by this author was partially supported by PRC "Mathématiques-Informatique," by GDR "Informatique et Génome," and by NATO Grant CRG 900293.

[2] Institute of Informatics, Warsaw University, ul. Banacha 2, 00913 Warsaw 59, Poland. Work by this author was supported by Grant KBN 2-11-90-91-01.

Received August 14, 1991; revised December 11, 1992. Communicated by C. K. Wong.

At the same time, we present a complete proof of the result which provides a cleaner version and a simpler analysis of the GS algorithm.

The GS algorithm can be treated as a space-efficient implementation of the algorithm of Knuth, Morris, and Pratt [KMP]. The KMP algorithm essentially works in stages composed of a left-to-right scan of the pattern against the text followed by a shift of the pattern to the right. The main operation in one stage consists of making a suitable safe shift of the pattern. We refer the reader to [KMP] and [A] for the definition of the shift function and the failure function related to the algorithm. More recently, Simon [S] has improved on the KMP algorithm, using a clever implementation of the automaton underlying it. In [C] a time–space optimal string-searching algorithm is presented that can also be considered as an efficient implementation of the KMP algorithm.

Space-efficient implementations are based on properties of the periods of prefixes $p[1..i]$ of the pattern p. If we know that the periods of these segments are always large with respect to i, then it is easy to reduce space, memorizing a single constant of proportionality. If the period of a prefix is small, then the pattern starts with a segment, which repeats in the pattern several times, say k times. Such a segment is defined later in this paper as a *highly repeating prefix* (HRP). The main parameter of the HRP is the number k which essentially describes the continuation of the HRP as a period. If $k = 2$, then the HRP yields a square prefix, if $k = 3$ it leads to a cube prefix. Hence, the combinatorics of cubes and squares in strings plays an important role in the problem. In particular, if the string is cube-free or square-free, the string searching is much simpler (no pattern preprocessing is even needed).

We show that the number of HRPs of a word x is $O(\log|x|)$. The result is very easy to prove for $k \geq 3$, but the proof becomes very intricate for $k = 2$. In this case the bound on the maximum number of HRPs is shown to be less than $\log_\Phi|x|$, where Φ is the golden ratio. We even exhibit the words that reach the upper bound. As a coincidence, the bound $\log_\Phi|x|$ is the same as the bound on the number of certain periods of the entire word x that come in the analysis of the KMP algorithm (see [KMP]).

We further examine the problem of computing all the periods of a word. A linear-time algorithm may be derived from the KMP algorithm. A time–space optimal algorithm to compute the periods of a word is given in [C]. We show that the GS algorithm can also be adapted for this purpose, provided the parameter k is chosen large enough ($k \geq 7$). Thus, this leads to a new time–space optimal algorithm for computing the periods of a word. It also corrects a flaw in [GS2].

The preprocessing phase of the GS algorithm consists of a decomposition of the pattern that we call a *k-perfect decomposition.* We show that preprocessing can be efficiently computed in parallel. To do so, we assume that $k \geq 4$, and we show how to compute a 4-perfect decomposition of the pattern in time $\log^2 n$ with $n/\log^2 n$ processors in the CRCW PRAM model. The complexities becomes respectively $\log^3 n$ and $n/\log^3 n$ in the CREW PRAM model.

The paper is organized as follows. The time–space optimal string-searching algorithm is presented in Section 2. It is based on the decomposition theorem proved in Section 3. Sections 4 and 5 are devoted to the case $k = 2$. The former

shows that the proof of Section 3 cannot be extended to this case, and the latter provides the accurate upper bound on the number of squares, prefixes of a word. The time–space optimal computation of periods is given in Section 6. The last section presents the parallel algorithm for the decomposition of a word.

Throughout this paper we consider a text t and a pattern p. Words t and p are on the same alphabet A. The empty word of A^* is denoted by ε.

2. Highly Repeating Prefixes. String-searching algorithms repeatedly perform a series of scans of the pattern against the text, and shifts of the pattern to the right. We are first interested in algorithms that perform left-to-right scans, as the KMP algorithm does. In these algorithms the current situation is when a mismatch is met during left-to-right scanning. The next step is then a shift of the pattern to the right. At this point, the algorithm has discovered inside the text t an occurrence of a prefix y of the pattern p followed by a letter b (see Figure 1). If y is the pattern itself, then an occurrence of the pattern is found. If not, ya is a prefix of p for some letter $a \neq b$.

The shift that comes next must if possible keep a prefix of the pattern matching the text. This is realized by both the KMP algorithm and Simon's algorithm. It is easy to see the following fact:

Let $x = vz$ with v nonempty. Then z is a prefix of x
iff x is a prefix of some power v^e of v.

When x is prefix of v^k for some nonempty prefix v of x, v is called a *prefix period* of x, and the integer $|v|$ is called a *period* of x. In other words, two letters occurring in x at distance $|v|$ coincide. The word z of the above fact, which is both a proper prefix and a suffix of x, is called a *border* of x. Borders and periods are in one-to-one correspondence. Finally, we denote by $\mathrm{per}(x)$ the smallest period of x.

The base of many combinatorial properties of repetitions in words is given by the well-known periodicity lemma of Fine and Wilf (see [L]). Its weak version can be formulated as follows:

Let p and q be two periods of a word x. If $p + q \leq |x|$,
then $\gcd(p, q)$ is also a period of x.

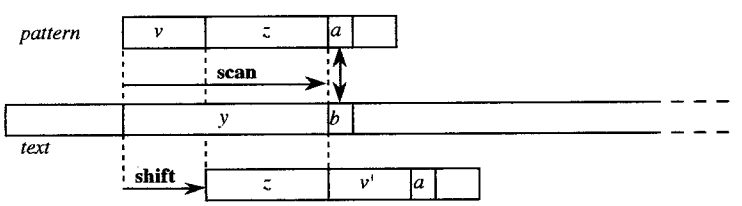

Fig. 1. Left-to-right scan: $|v|$ is a period of $y = vz$.

The KMP string-matching algorithm is based on a linear-time computation of periods of all prefixes of pattern p, and, since all these periods (or associated borders) can be pairwise distinct, the space complexity of the algorithm is inherently $O(|p|)$.

The idea used in GS algorithm to save on space is to eliminate some prefix periods of the pattern, namely its large periods. When the prefix y of Figure 1 has a large period, the algorithm shifts the pattern to the right a number of places less than the shortest period of y. This kind of shift is not optimal but no occurrence of the pattern in the text is missed, and this avoids memorizing large periods.

Large periods are defined relative to an integer k that is always considered as greater than 1 in the following. It is greater than 3 in [GS2]. We show that the same approach works for $k = 3$ in a later section.

Recall that a word v is said to be *primitive* if it is not a power of another word, that is, $v = u^i$ implies both $u = v$ and $i = 1$. Note that the empty word is not primitive.

We now introduce the basic notions. Let v^k be a prefix of x with v a primitive word ($k > 1$). The word v is called a *k-highly repeating prefix* of x, a k-HRP of x, or even an HRP of x when k is clear from the context. When v is a k-HRP of x, the prefix v^2 of x has the smallest period $|v|$. Thus, we can consider the longest prefix z of x which has the prefix period v. Then *the scope of* v is the interval of integers $[L, R]$ defined by

$$L = |v^2| \quad \text{and} \quad R = |z|.$$

Note that, by definition, any prefix of x whose length falls inside the scope of v has the prefix period v. Some shorter prefixes may also have the same period, but we do not deal with them. Figure 2 shows the structure of scopes of k-HRPs of a word x.

Let us take $k = 2$ and look at the Fibonacci word Fib_9 (recall that the sequence of Fibonacci words is defined by $\text{Fib}_1 = b$, $\text{Fib}_2 = a$, and $\text{Fib}_i = \text{Fib}_{i-1}\text{Fib}_{i-2}$ for $i > 2$). Figure 3 displays the scopes of its 2-HRPs.

The following lemma shows that all the scopes of HRPs of a word x are pairwise disjoint (see Figure 2), Lemma 2 gives a lower bound on periods of prefixes whose length does not belong to any scope of an HRP. These two lemmas provide the basic elements for a proof of Algorithm SIMPLE-TEXT-SEARCH (shown in Figure 4).

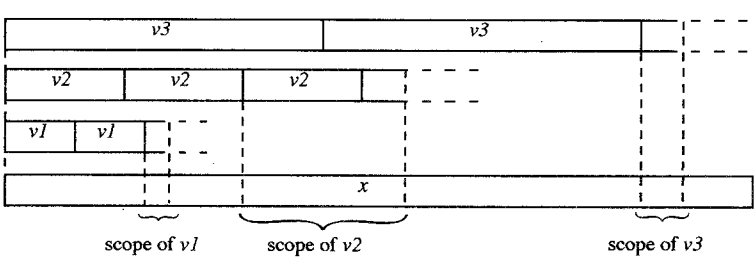

Fig. 2. Scopes of highly repeating prefixes.

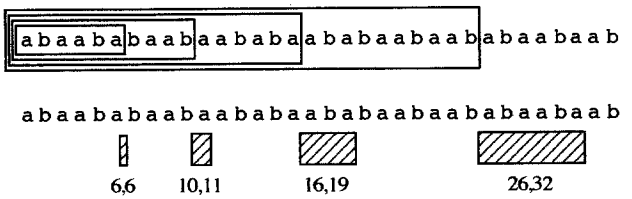

Fig. 3. The ninth Fibonacci word: its repeating prefixes and associated scopes.

LEMMA 1. *Let* $[L_1, R_1]$ *and* $[L_2, R_2]$ *be the respective scopes of two different HRPs* v_1 *and* v_2 *of* x. *Assume that* $|v_1| < |v_2|$. *Then* $R_1 < L_2$.

PROOF. Let z be the prefix of length R_1 of x. If $L_2 \leq R_1$, then the square v_2^2 is a prefix of z. Therefore, the periodicity lemma applies to periods $|v_1|$ and $|v_2|$ of the prefix v_2^2. It implies that $\gcd(|v_1|, |v_2|)$ is a period of this prefix. However, since $|v_1| < |v_2|$, $\gcd(|v_1|, |v_2|) < |v_2|$ and we get a contradiction with the primitivity of v_2. □

LEMMA 2. *Let* $(v_i/i = 1, \ldots, r)$ *be the sequence of k-HRPs of* x. *Let* $([L_i, R_i]/i = 1, \ldots, r)$ *be the corresponding sequence of scopes. Then any nonempty prefix* u of x *satisfies*

$$\text{per}(u) = L_i/2, \quad \text{if } |u| \text{ is in } [L_i, R_i] \text{ for some } i,$$
$$\text{per}(u) > |u|/k, \quad \text{if not.}$$

PROOF. Let u be a nonempty prefix of x. If $|u|$ does not belong to any scope of k-HRP, any prefix period of u has length greater than $|u|/k$. Thus $\text{per}(u) > |u|/k$. Assume that $|u|$ belongs to some $[L_i, R_i]$. It is then a prefix of some power v_i^e $(e \geq 2 > 1)$ of the ith k-HRP v_i of x. Word u has period $|v_i|$ which is equal to

```
Algorithm SIMPLE-TEXT-SEARCH
/* Searches text t for pattern p. */
/* An O(r)-space version of the KMP algorithm, where r is the number of k-HRPs
   of p. ([L_i, R_i]/i = 1, ..., r) is their sequence of scopes. */
pos := 0; j := 0;
while pos ≤ n − m do
  { while j < m and p[j + 1] = t[pos + j + 1] do j := j + 1;
    if j = m then return match at position pos;
    if j belongs to some [L_i, R_i] then  { pos := pos + L_i/2; j := j − L_i/2; }
    else  {pos := pos + ⌊j/k⌋ + 1; j := 0; }
     }
   return false;
end.
```

Fig. 4. The searching-phase algorithm.

$L_i/2$ by the definition of scopes. Applying the preceding lemma, it can also be deduced that u has no shorter period, which means $\text{per}(u) = L_i/2$. □

Algorithm SIMPLE-TEXT-SEARCH is a version of the KMP algorithm. It is an adaptation of the algorithms of [GS1] and [GS2]. During a run of the algorithm, lengths of shifts are computed according to the property stated in Lemma 2 on the list of scopes of k-HRPs. We assume that these intervals have been computed previously.

Inside Algorithm SIMPLE-TEXT-SEARCH, the test "j belongs to some $[L_i, R_i]$" can be implemented in a straightforward way. It needs $O(1)$ space in addition to the list of scopes, and does not affect the asymptotic time complexity of the algorithm. The clue is to have a pointer to the current scope, whose value is recomputed each time variable j is modified.

THEOREM 3. *Assume that the pattern p has r k-HRPs and that their list of scopes and their list of lengths are already computed. Then the Algorithm SIMPLE-TEXT-SEARCH solves the string-searching problem on p and t using $O(r)$ space and performing at most $k \cdot |t|$ symbol comparisons. The total complexity of the algorithm is also linear in $|t|$.*

PROOF. see [GS2]. To evaluate the time performance of the algorithm it can be shown that the value of expression $k \cdot \text{pos} + j$ is strictly increased by each symbol comparison. □

We now come to the preprocessing phase. It is presented in Figure 5 as Algorithm PREPROCESS. We leave the proof of the time linearity of Algorithm PREPROCESS to the reader. It is worth noting that the algorithm works in the same way as Algorithm SIMPLE-TEXT-SEARCH, i.e., as if the pattern is being sought inside itself, which explains why it is linear.

Algorithms SIMPLE-TEXT-SEARCH and PREPROCESS require $O(r)$ extra space to work. This space is used to store the scopes of the k-HRPs of the pattern

```
Algorithm PREPROCESS
/* Computes the k-highly repeating prefixes of pattern p of length m. */
/* Returns their list of scopes (in increasing order) ([L_i, R_i]/i = 1, ..., r) */
    let SCOPE be an empty list;
    pos := 1; j := 0;
    while pos + j < m do
    { while pos + j < m and p[j + 1] = p[pos + j + 1] do j := j + 1;
      if pos ≤ (pos + j)/k then add [2*pos, pos + j] to the end of SCOPE;
      if j belongs to some [L_i, R_i] in SCOPE then { pos := pos + L_i/2; j := j - L_i/2; }
      else { pos := pos + ⌊j/k⌋ + 1; j := 0; }
    }
    return SCOPE;
end.
```

Fig. 5. The preprocessing-phase algorithm.

p. A simple application of the periodicity lemma shows that, for $k \geq 3$, a nonempty word x has no more than $\log_{k-1}|x|$ k-HRPs. This relies on the following fact: if u and v are two HRPs of x and $|u| < |v|$, then the stronger inequality holds, $(k-1)\cdot|u| < |v|$. However, the case $k = 2$ is more difficult to handle, and one of our results is the logarithmic bound for the case $k = 2$ (see Section 5).

Denote by HRP1(x) and HRP2(x) respectively, when defined, the shortest and the second shortest k-HRPs of a given word x. The basic tools are now properties and interplay between HRP1s and HRP2s starting at some specific positions inside the pattern. We can use Algorithm PREPROCESS to compute HRP1s and HRP2s. The trick is simply to stop the execution of the algorithm the first time it discovers the second HRP. This proves the following.

LEMMA 4. *Assume that pattern p has at least two k-HRPs. The shortest and the second shortest k-HRPs of p,* HRP1(x) *and* HRP2(x) *respectively, can be computed in $O(1)$ space and time proportional to the length of* HRP2(x).

3. Cube Prefixes. What happens if the pattern has at most one k-HRP? We say that such a pattern is *k-simple*. In this case, obviously, we can make string searching in linear time using only $O(1)$ memory space by applying Algorithms PREPROCESS and SIMPLE-TEXT-SEARCH of the previous section. The most memory consuming is the list of scopes, but now this list is reduced to only one pair of integers specifying only one scope (or even no pair if the pattern has no HRP at all). Unfortunately not all patterns are simple. For instance, if we take $k = 2$, then the Fibonacci words are not simple. However, these words become simple with $k = 3$. On the contrary, the word $((a^e b)^e c)^e$ is not simple for any $k \leq e$.

When $k \leq 3$, words satisfy a remarkable combinatorial property (Theorem 5 below), originally discovered by Galil and Seiferas for $k \geq 4$ (see [GS2]):

each pattern p can be decomposed into uv
where u is "short" and v is a k-simple word.

With such decomposition of the pattern p the searching algorithm can be realized conceptually in two phases as follows. First, find all occurrences of v, which is efficient due to the simplicity of v. The algorithm SIMPLE-TEXT-SEARCH can be used for this purpose. Next, check whether the occurrences of v are preceded by u, which is fast due to the "shortness" of u. We define the shortness of u in relation to the period of v. More precisely, we say that the decomposition $p = uv$ is *k-perfect* iff v is k-simple and $|u| \leq 2\cdot \text{per}(v)$. The time–space optimal string-searching algorithm is based on the next theorem. The proof is reported after the algorithm.

THEOREM 5 (Decomposition Theorem). *For $k \geq 3$, each nonempty pattern p has a* k-perfect decomposition.

```
Algorithm TEXT-SEARCH
/* Searches text t for pattern p. */
/* Let k ≥ 3 and let uv be a k-perfect decomposition of pattern p */
    find all occurrences of v in t with Algorithm SIMPLE-TEXT-SEARCH;
        for each position i of an occurrence of v in t do
          { check by a naive algorithm if u ends at i;
            if "yes" then report the match at position i-|u|;
    }
    end.
```

Fig. 6. The searching phase with k-perfect decomposition.

Once we have a perfect decomposition of the pattern p, as explained above, we can search for it with Algorithm TEXT-SEARCH (Figure 6).

The following theorem, proved in [GS2], shows that Algorithm TEXT-SEARCH is time–space optimal, and that its time complexity depends monotonically on the parameter k.

THEOREM 6 [GS2]. *Algorithm TEXT-SEARCH computes the positions of occurrences of pattern p inside text t. It uses constant extra memory space. It is linear in time and makes at most* $(k+2)\cdot|t|$ *symbol comparisons.*

The rest of the section is devoted to the proof of Theorem 5. The proof is constructive and eventually shows how to compute a perfect decomposition. We first prove constructively that such a decomposition exists, and then analyze the complexity of the construction. The proof essentially relies on two lemmas. In the rest of the section, unless otherwise stated, HRP means 3-HRP, and analogously for HRP1 and HRP2.

LEMMA 7. *Assume that* $z = \mathrm{HRP1}(x)$ *is defined and let* $x = zx'$.

(a) *If* $\mathrm{HRP2}(x)$ *is defined,* $|\mathrm{HRP2}(x)| > 2\cdot|\mathrm{HRP1}(x)|$.
(b) *If* $\mathrm{HRP1}(x')$ *is defined,* $\mathrm{HRP1}(x)$ *is a prefix of* $\mathrm{HRP1}(x')$.

PROOF. The statements are illustrated in Figures 7 and 8. Use the periodicity lemma and the fact that k-HRPs are primitive words. □

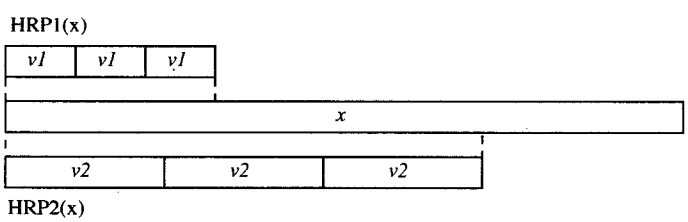

Fig. 7. (Lemma 7(a).) $\mathrm{HRP2}(x)$ is at least twice as long as $\mathrm{HRP1}(x)$.

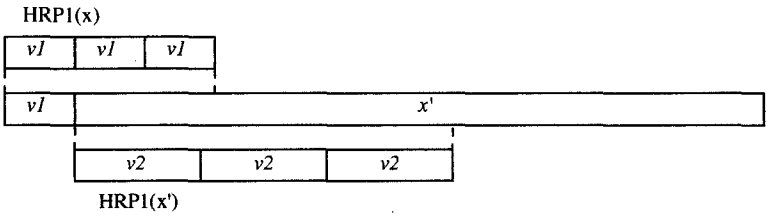

Fig. 8. (Lemma 7(b).) Sizes of HRP1 are nondecreasing.

The structure of the decomposition algorithm is based on a sequence $V(x)$ of HRP1s. The elements of the sequence $V(x) = (v_1, v_2, \ldots)$ are called *working factors*, and are defined as follows. The first element v_1 is HRP1(x). Let $x = v_1x'$, then v_2 is HRP1(x'), and so on until there is no HRP1. In particular, the sequence is empty if x does not start with any kth power. Lemma 7 shows that the sizes of the v_i's are in nondecreasing order. The next lemma completes properties of their sizes: some v_i eventually reach the length of HRP2(x), which is at least twice the length of v_1 by Lemma 7 again (see Figure 9).

LEMMA 8 (Key Lemma). *Let $V(x) = (v_1, v_2, \ldots)$ be the sequence of working factors of x, and assume that* HRP2(x) *exists. Let i be the greatest integer such that $|v_1 \cdots v_i| < |\text{HRP2}(x)|$. Then, if v_{i+1} exists, $|v_{i+1}| \geq |\text{HRP2}(x)|$.*

PROOF. Let w be HRP2(x) and assume that v_{i+1} exists. The assumptions imply that v_{i+1} overlaps the boundary between the two first occurrences of w as shown in Figures 11 and 12. We show that v_{i+1} cannot be shorter than w. Assume that the contrary holds. Let y and $y' \neq \varepsilon$ be defined by the equalities $w = v_1v_2 \cdots v_iy'$ and $v_{i+1} = y'y$. Because we assume v_{i+1} is shorter than w, y is a prefix of w, so that we can consider the word z defined by $w = yz$. We now focus our attention on the word $w' = zy$, a rotation of w.

Case 1. We first consider the situation when the beginning of w' (inside the first occurrence of w) falls properly inside some v_j (see Figure 11). Integer j is less than $i + 1$ because $|v_{i+1}| < |w'|$. Since $w = \text{HRP2}(x)$, w^3 is a prefix of x, and then another occurrence of w' immediately follows the occurrence we consider at position $|y|$. Because v_{i+1} is in HRP1, two other occurrences of it appear after the one at position $|v_1 \cdots v_i|$; v_{i+1} is thus a prefix of w'. By Lemma 7(b), we know that v_j is a prefix of v_{i+1}. Both arguments imply that v_j is a prefix of w'. This eventually

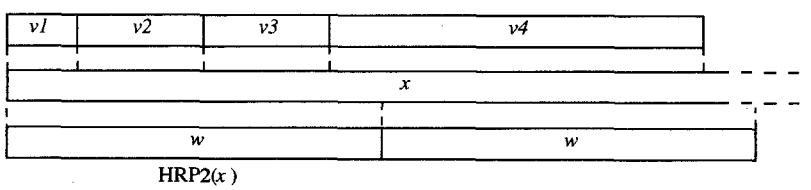

Fig. 9. The sequence of HRP1s ($|v4| = |w|$).

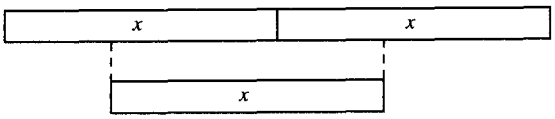

Fig. 10. An impossible situation when x is primitive.

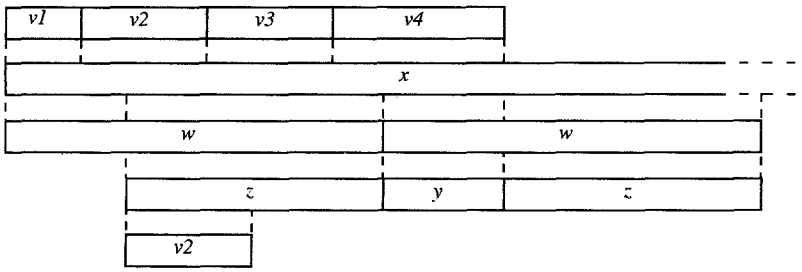

Fig. 11. Case 1: impossible because $v2$ is primitive.

implies that v_j is an internal factor of v_jv_j, a contradiction with the primitivity of v_j because, for a primitive word x, the situation presented in the Figure 10 is impossible.

Case 2. The second situation is when $w' = v_j \cdots v_iv_{i+1}$ (see Figure 12). Again hypothesis $|v_{i+1}| < |w|$ leads to $j < i + 1$. This implies $2|v_j| < |w|$ by a simple application of the periodicity lemma. Therefore, the word v_j is also a 3-HRP1 of zyz, because the latter is longer than $3|v_j|$. Thus, just after the occurrence of w' which is considered, v_j is still an HRP1. This proves, through Lemma 7(b), that $v_{i+1} = v_j$. However, then w' is a nontrivial power of v_j, and its rotation $w =$ HRP2(x) is not primitive, a contradiction. □

Lemma 8 yields the notion of *special positions* in x. The first special position of x is defined as the length of the word $v_1 \cdots v_i$ which arises from Lemma 8. If $x = v_1 \cdots v_ix'$, then the second special position is defined on x' in the same manner, and so on until no application of Lemma 8 is possible (see Figure 13). If x has no HRP2, or even no HRP1 at all, the first position 0 is the only special position (corresponding to the decomposition (ε, x) of x).

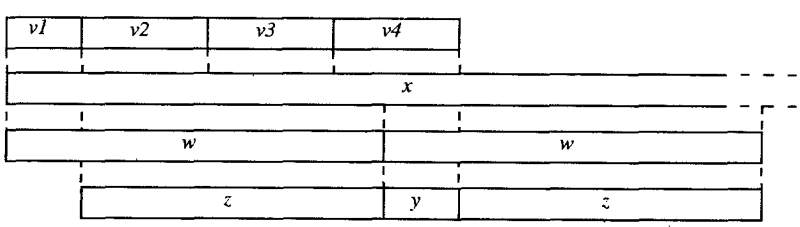

Fig. 12. Case 2: impossible because w = HRP2(x) is primitive.

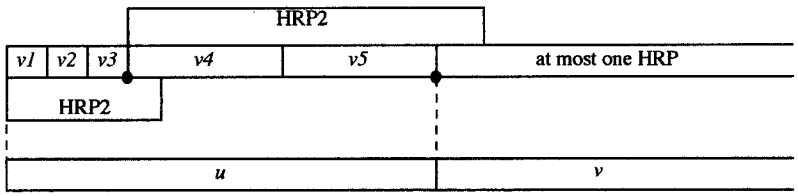

Fig. 13. The sequence of working factors $v1, v2, \ldots$ leads to special position •, and to the 3-perfect decomposition uv of x.

THEOREM 5 (Decomposition Theorem). *Let j be the last special positon of x. Let $u = x[1..j]$ and $v = x[j+1..n]$. Then the decompositon uv of x is a k-perfect decomposition for $k \geq 3$.*

PROOF. By definition of the sequence of special positions of x, v is certainly 3-simple because it has at most one HRP. Indeed, word v is k-simple for any $k \geq 3$. It remains to prove that $|u| < 2 \cdot \text{per}(v)$. The key point is that each next HRP2 is at least twice as large as the preceding one. This follows from Lemmas 7 and 8. □

The algorithm given by Galil and Seiferas [GS2] to compute perfect decompositions can be described informally as follows:

— Compute the sequence of special positions from left to right (see Figure 13).
— Compute the sequence of HRP1s to get the special position within the current HRP2 (see Figure 9).

A more formal description of the algorithm is given in Figure 14.

The correctness of Algorithhm DECOMPOSE immediately follows from the decomposition theorem (through Lemmas 7 and 8). The time complexity of the algorithm may be roughly analyzed as follows: from Lemma 4, the cost of the computation of the working factors is proportional to the total length of consecutive HRP1s, which is linear; the cost of the computation of consecutive HRP2s is proportional to their total length, which is also linear. Hence the algorithm takes linear time and constant extra space.

```
Algorithm DECOMPOSE(x)
/* Computes a k-perfect decomposition of pattern x (k ≥ 3), HRP is related to k. */
    j := 0; hrp1 := |HRP1(x)|; hrp2 := |HRP2(x)|;
    while hrp1 and hrp2 exist do
    { j := j + hrp1; hrp1 := |HRP1(x_{j+1} ··· x_n)|;
      if hrp1 ≥ hrp2 then hrp2 := |HRP2(x_{j+1} ··· x_n)|;
    }
    return the decomposition (x_1 ··· x_j, x_{j+1} ··· x_n) of x;
end.
```

Fig. 14. Computing perfect decompositions.

ε, aabaababaabaababaabaabab	*aabaa, babaabaababaabaabab*
aa	*baba*
a, abaababaabaababaabaabab	*aabaab, abaabaababaabaabab*
abaaba	*abaaba*
aa, baababaabaababaabaabab	*aabaaba, baabaababaabaabab*
baababaaba	*baabaa*
aab, aababaabaababaabaabab	*aabaabab, aabaababaabaabab*
aa	*aa*
aaba, ababaabaababaabaabab	
abab	

Fig. 15. Shortest squares, prefixes of the right parts of the factorizations.

4. The Perfect Decomposition Theorem Does Not Hold for $k = 2$. In the previous section the k-perfect decomposition theorem has been proved for $k > 2$. We show here that this is the smallest bound. This means that the GS algorithm does not work in constant space for $k = 2$. It works, however, in logarithmic space, as we will see in the next section.

The consequence of the following proposition is that if uv is a factorization of x with v having at most one 2-HRP, then u cannot always be kept small. The proof is based on a rather short example shown in Figure 15.

PROPOSITION 9. *The decomposition theorem does not hold for $k = 2$. More precisely, for any constant $c > 0$ infinitely many words x with the following property exist: if uv is a factorization of x such that v has at most one* 2-HRP, *then* $|u|/\mathrm{per}(v) > c$ (*indeed, an even stronger inequality holds*: $|u|/|v| > c$).

PROOF. *Let $w = aabaabab$.* Note that the smallest period of this word and thus of w^n ($n > 0$) is its length $|w| = 8$. Consider a factorization of the word w^3 of the form $uu'w^2$ (with $u, u' \in A^*$ and $w = uu'$). The word $v = u'w^2$ obviously starts with the square $(u'u)^2$ but, as shown in Figure 15, it also starts with a strictly shorter square. For instance, if $u = aa$ and $u' = baabab$, v starts with the square $(baaba)^2$.

The factorization (*aabaababaa, baababaabaabab*) *of* w^3 is such that the right part has at most one 2-HRP. The word $v = baababaabaabab$ is indeed the longest suffix of w^3 with this property. Thus, to meet the result of the proposition, one simply has to choose $x = w^n$ for a large enough integer n. □

5. Square Prefixes: $k = 2$ Works Well With Logarithmic Space. In this section we consider again 2-HRPs of patterns. It remains to consider prefixes of the form u^2 of a word x, with u a primitive word. We call them *square prefixes.*

The number of square prefixes of a word can be logarithmic as happens for Fibonacci words (see Figure 3). This follows easily from the recurrences defining Fibonacci words and from the fact that two consecutive Fibonacci words "almost" commute (up to the exchange of the last two symbols). We show that the maximal

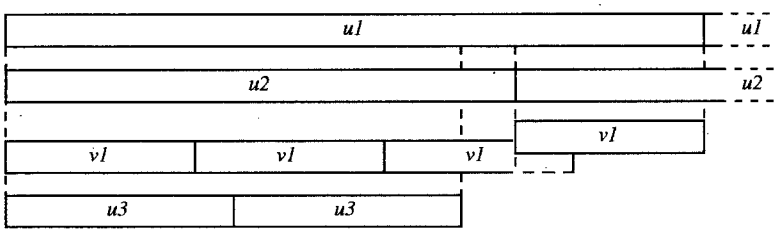

Fig. 16. Case 1 in the proof of Lemma 10.

number of square prefixes of a word is also logarithmic and is precisely less than $\log_\Phi |x|$, where Φ is the golden ratio. Moreover, we characterize the words that reach the bound. As a consequence this shows that Algorithms SIMPLE-TEXT-SEARCH and PREPROCESS of Section 2 work in logarithmic space when applied with $k = 2$.

Theorem 11 completes the note of Section 2 on the logarithmic number of k-HRPs for $k \geq 3$. Its proof is based on the next lemma, that gives a fundamental property on the lengths of square prefixes. The relation between three periods of a square prefix parallels the recurrence relation defining Fibonacci numbers.

LEMMA 10. *Let $u1^2, u2^2, u3^2$ be three prefixes of x such that $u1, u2, u3$ are primitive and $|u3| < |u2| < |u1|$. Then $|u2| + |u3| < |u1|$.*

PROOF. Let $p1$, $p2$, $p3$ be the respective lengths of the three words $u1$, $u2$, $u3$. When $p2 \leq p1/2$, since $p3 < p2$, we obviously get the conclusion $p2 + p3 \leq p1$. We then assume $p2 > p1/2$ which implies that $u1$ is a prefix of $u2^2$. We consider the word of length $p1 - p2$, $v1 = u2^{-1}u1$, that is both a prefix of $u2$ and a suffix of $u1$. The integer $|v1|$, less than $p2$, is a period of $u2$ ($u2$ occurs at position $p2$ in x, and also at position $p1$ in x because $u2$ is a prefix of $u1$).

It remains to prove that $p3 \leq |v1|$. We assume *ab absurdo* that $|v1| < p3$. We consider two cases according to the length of $u3$. They are illustrated in Figures 16 and 17.

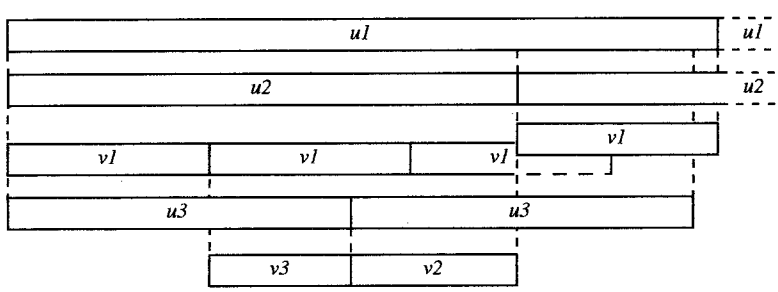

Fig. 17. Case 2 in the proof of Lemma 10.

Case 1 (see Figure 16). We first rule out the situation when $u3^2$ is a prefix of $u2$. It happens then that $u3^2$ has two periods $p3$ and $|v1|$. Moreover $p3 + |v1| < 2 \cdot p3 = |u3^2|$. The periodicity lemma applies and proves that $\gcd(p3, |v1|)$ is also a period of $u3^2$. Since this period, less than $p3$, divides $p3 = |u3|$, it follows that $u3$ is not primitive, a contradiction.

Case 2 (see Figure 17). We consider now the opposite situation when $u2$ is a prefix of $u3^2$. The word $u2$ has periods $p3$ and $|v1|$. As in Case 1, if the periodicity lemma applies to the word $u2$, we get a contradiction with the primitivity of $u3$. Thus, the lemma does not apply, which means that $p3 + |v1| > p2$. Let $v2$ be the word $u3^{-1}\,u2$. It is both a prefix of $u3$ and a suffix of $u2$. Since it has length $p2 - p3$, it is shorter than $v1$: $|v2| < |v1|$. Note that $|v2|$ is a period of $u3$ ($u3$ occurs at position $p3$, and at position $p2$ as a prefix of $u2$). Let $v3$ be such that $u3 = v1v3$ ($v1$ is a prefix of $u3$ because we assume *ab absurdo* that $|v1| < p3$). We have $u2 = vlv3v2$ (see Figure 17).

Consider as a first step the word $v3v2$. Since $|v1|$ is a period of $u2$, $v3v2$ is a prefix of $u2$. Moreover, it is a proper prefix of $u3$ because its length $|v3| + |v2|$ is smaller than the length of $u3$, $|v3| + |v1|$. Therefore, the word $v3v2$ occurs at positions $|v1|$ and $|v1v3|$ in x, which proves that it has a period $|v3|$. It also has a period $|v2|$, like $u3$. The periodicity lemma applies: $r = \gcd(|v3|, |v2|)$ is a period of $v3v2$. The conclusion of this first step is that r is a period of $u3$ itself because r divides the period $|v2|$ of $u3$.

In a second step we consider the word $u3$. It has periods r and $|v1|$. We have $r + |v1| \leq |v3| + |v1| = |u3|$. By the periodicity lemma once more, $r' = \gcd(r, |v1|)$ is a period of $u3$. However, r' that divides both $|v1|$ and $|v3|$ also divides their sum $|v1| + |v3| = |u3|$. This is impossible because $u3$ is primitive. This ends Case 2.

Since both cases are impossible, $p3 \leq |v1|$, and the conclusion follows: $p2 + p3 \leq p1$. □

The inequality conclusion of Lemma 10 cannot be improved. For instance, the Fibonacci word of Figure 3 has four square prefixes of respective lengths 6, 10, 16, and 26. Furthermore, it is well known that f_i^2 is a prefix of the Fibonacci word f_j, for all i in the interval $(4, \ldots, j-2)$.

EXAMPLE. The word *ababaababababaab* begins with squares of lengths 4, 10, and 14 corresponding to the 2-HRPs *ab*, *ababa*, and *ababaab*. Computation proves that no shorter word can have three 2-HRPs like it.

FACT. The word *aabaabaaabaabaabaaab* begins with squares of lengths 2, 6, 14, and 20 corresponding to the 2-HRPs *a*, *aab*, *aabaaba*, and *aabaabaaab*. Computation proves that no shorter word can have four 2-HRPs like it. Moreover, it is the unique word of length 20 on the alphabet $\{a, b\}$ (up to the exchange of letters a and b) having this property.

THEOREM 11. *The number of square prefixes of a nonempty word x is less than* $\log_\Phi |x|$ (*where Φ stands for the golden ratio*; $\Phi = (1 + \sqrt{5})/2 = 1.618\ldots$).

PROOF. We first prove that, for $i \geq 1$, if x has i 2-HRPs, then $|x| > 2 \cdot F_{i+1}$ (where F_i is the ith Fibonacci number, $F_i = |f_i|$). For $i = 1$, $|x| \geq 2 = 2 \cdot F_2$. For $i = 2$, $|x| \geq 6 > 2 \cdot F_3 = 4$. By the fact stated in the example above, for $i = 3$, $|x| \geq 14 > 2 \cdot F_4$. The rest of the proof is by induction on i. Let $i \geq 4$. Let $u1$, $u2$, and $u3$ be the three longest 2-HRPs of x (in order of decreasing lengths). The word $u3^2$ has exactly i-2 2-HRPs. By induction its length is then greater than $2 \cdot F_{i-1}$. For the same reason $|u2^2| > 2 \cdot F_i$. By Lemma 10, we get $|u1| \geq |u3| + |u2|$ which implies $2 \cdot |u1| > 2 \cdot F_{i-1} + 2 \cdot F_i = 2 \cdot F_{i+1}$. Thus $|x| \geq 2 \cdot F_{i+1}$, as expected. The Fibonacci number F_{i+1} is greater than (or equal to) Φ^{i-1}. Thus $|x| > 2 \cdot \Phi^{i-1}$ which yields $i < \log_\Phi |x|$, an inequality that also holds for $i = 1$. □

It is known from [KMP] that the number of certain periods of a word is bounded by $\log_\Phi |x|$. These periods of x are all greater than (or equal to) per(x), the smallest one. From this point of view, the result of Theorem 11 is a dual result that gives the same bound on the number of small starting periods of x. They are all not greater than per(x). The word shown in the above fact gives an example of a word which reaches the maximum number of square prefixes. The next proposition is a continuation of this example.

PROPOSITION 12. *Consider* $(u_i / i \geq 0)$ *the sequence of words on the alphabet* $\{a, b\}$ *defined by*

$$u_0 = a, \quad u_1 = aab, \quad u_2 = aabaaba, \quad \text{and} \quad u_i = u_{i-1}u_{i-2} \quad \text{for} \quad i \geq 3.$$

Then, for each i, u_i^2 *has the maximum number of square prefixes. Moreover, for* $i \geq 3$, u_i^2 *is the unique word on the alphabet* $\{a, b\}$ *having this property (up to a permutation of letters a and b).*

PROOF. A consequence of the above fact and Lemma 10. □

6. Time–Space Optimal Computation of Periods ($k = 7$). The evaluation of the periods of a word is strongly related to the computation of all overhanging occurrences of a pattern in a text. Indeed, periods of a word correspond to overhanging occurrences of the word over itself (see Figure 18). An overhanging occurrence of the pattern p in the text t is specified by a suffix z of t that is also a proper prefix of p. If t is written yz, the position of the overhanging occurrence of p is $|y|$, and if $t = p$, the position $|y|$ is also a period of p, as already noted in Section 2.

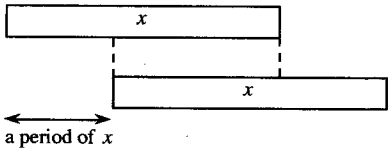

Fig. 18. Period and overhanging occurrence.

String-searching algorithms which use left-to-right scanning of the pattern against the text can be naturally extended to compute overhanging occurrences of the pattern. Thus, they can be used to compute all the periods of a word. It is the case for the algorithm of Knuth, Morris, and Pratt [KMP] and its variations, and also for the time–space optimal algorithm of Crochemore [C]. We show that the algorithm of Galil and Seiferas is also adapted for this purpose, provided the parameter k is large enough ($k \geq 7$). This then leads to a new time–space optimal algorithm for computing periods in words.

The computation of periods through the GS algorithm given in [GS2] was based on the following assertion: "per(x) is the position of the second occurrence of x inside xx." Unfortunately, this assertion does not always hold. For instance, $x = ababa$ has period 2 and the second occurrence of x inside xx is at position 5. Indeed, the assertion holds when x is a power of its shortest prefix period.

Let uv be a k-perfect decomposition of the pattern p. If u is empty, the algorithm of Section 2 naturally extends to the computation of overhanging occurrences. It works in linear time and constant space. The situation is different when u is nonempty because v can have many overhanging occurrences in t and thus u could be scanned too many times by the algorithm. To avoid this situation we impose a condition to the decomposition of the pattern and this yields the choice $k = 7$ for the parameter.

Assume u is not empty and $k \geq 3$. The pattern p can then be written $p = uv = u'w^{k'}v'$ with u' a proper prefix of u, $|u| < |w| \cdot (k-1)/(k-2)$, and $k \leq k'$. The word w is the last k-HRP2 computed by Algorithm DECOMPOSE of Figure 14.

We define the following condition:

$$(*) \qquad |u| + 2 \cdot |w| \leq |p|/2.$$

LEMMA 13. *Let uv be the k-perfect decomposition of p computed by Algorithm DECOMPOSE. If u is nonempty and $k \geq 7$, then condition $(*)$ is satisfied.*

PROOF. If u is nonempty we know that p contains k-HRPs. Let p $u'w^{k'}v'$ where w is the last k-HRP2 computed by Algorithm DECOMPOSE. Then $|u| < |w| \cdot (k-1)/(k-2)$ and $|v| > |w| \cdot (k-1)$. Condition $(*)$ is also $|v|/2 - |u|/2 - 2 \cdot |w| \geq 0$ or $|v| - |u| - 4 \cdot |w| \geq 0$. We have $|v| - |u| - 4 \cdot |w| \geq |w| \cdot (k-1) - |w| \cdot (k-1)/(k-2) - 4 \cdot |w|$. The inequality $k - 5 - (k-1)/(k-2) \geq 0$ is satisfied for $k \geq 7$. □

To compute overhanging occurrences of p in t we may assume without loss of generality that $|t| = |p|$. The core of the algorithm is a procedure that computes the overhanging occurrences of p which starts in the left half of t (see Figure 19).

LEMMA 14. *On entry p and t, Algorithm* Left_Occ *works in constant space and time $O(n)$ where $n = |t| = |p|$.*

PROOF. The result is obvious if u is empty. If not, since condition $(*)$ is satisfied, the number of overhanging occurrences of v starting in the left half of t is less than $n/|w|$, where w is the last k-HRP2 computed by Algorithm DECOMPOSE.

```
Function Left_Occ (p, t)
/* It is assumed that |t| = |p| */
/* Computes the set {|y|/t = yz, z prefix of p, and |y| ≤ |t|/2} of positions of p in
   the left half of t */
    P := ∅;
    compute 7-perfect decomposition (u, v) of p;
    find all overhanging occurrences of v starting in the left half of t with
      Algorithm SIMPLE-TEXT-SEARCH;
    for each position i of an occurrence of v in t do
      if u is a suffix of t[1] ... t[i] then add i to P;
    return P;
end.
```

Fig. 19. Computing left overhanging occurrences.

Thus, testing for occurrences of u in t takes less than $(n/|w|) \cdot 6/5 \cdot |w|$ comparisons because $|u| < 6/5 \cdot |w|$. The time is then $O(n)$. □

To compute all overhanging occurrences of p in t, we first apply the algorithm Left_Occ and then repeat the same process with both the right half of t and the left half of p (Figure 20).

THEOREM 15. *Algorithm* Ov_Occ *finds all overhanging occurrences of p in t in constant space and time* $O(n)$ *where* $n = |p| = |t|$.

PROOF. The correctness is left for the reader. Constant space is enough to memorize left or right parts of input strings. The time complexity is bounded by

$$c \cdot n + c \cdot n/2 + \cdots + c \cdot n/2^i,$$

where c is the constant of proportionality for Algorithm Left_Occ. This quantity is less than $2 \cdot c \cdot n$ which proves the result. □

Corollary 16. *Algorithm* Ov_Occ *finds all periods of a word of length n in constant space and time* $O(n)$.

```
Function Ov_Occ (p, t)
/* Computes all the overhanging occurrences of p in t; we assume |t| = |p| */
    P := Left_Occ(p, t);
    while 7-perfect decomposition of p is different from (ε, v) do
    { n := |t|/2; t := right half of t, t[n + 1 .. |t|]; p := left half of p of length
      |t| − n; add Left_Occ(p, t) + n to P: }
    return( P );
end.
```

Fig. 20. Computing overhanging occurrences.

PROOF. Apply Ov_Occ to the pair (p, p) and note that the position of an overhanging occurrence of p over itself is a period of p (see Figure 18). □

7. Optimal Parallel Computation of the 4-Perfect Decomposition. We assume in this section that $k = 4$. Assume that $v1 = \text{HRP1}(x)$ exists. Consider the maximum integer e such that $v1^e$ is prefix of x ($e \geq 4$ because $k = 4$). Define LongHRP1(x) to be $w = v1^{e-3}$. Let x' be such that $x = wx'$. By definition of $v1$ and e, $v1$ is not the HRP1(x'). Since HRP(x') cannot be shorter than $v1$, if it exists it is longer than $v1$ (see Figure 21).

We define the parallel working sequence of the word x as follows: $P(x)$ is empty if LongHRP1 (x) is undefined; otherwise $x = \text{LongHRP1}(x)x'$; the first element of $P(x)$ is LongHRP1(x) and the rest of the sequence is defined similarly on x'.

We compute the elements of the parallel sequence $P(x)$, or more precisely the sequence of positions of LongHRP1s occurring in the sequence. The crucial point in the parallel algorithm is the following fact due to $k = 4$ (and the periodicity lemma):

FACT Let $P(x) = (w_1, w_2, \ldots)$ be the parallel working sequence of x. For $i > 0$, $2 \cdot |w_i| < |w_{i+1}|$. The length of $P(x)$ is $O(\log|x|)$.

We use the components of the string-searching algorithm of Vishkin [V] to compute the basic functions used in the preprocessing phase of the GS algorithm.

LEMMA 17. *Assume* HRP1(x), HRP2(x), *and* LongHRP1(x) *exist. Let* $n1$, $n2$, *and* $n3$ *be their respective lengths. Let* $n = |x|$. *Then* HRP1(x), HRP2(x), *and* LongHRP1(x) *can be computed by an optimal parallel algorithm in* $O(\log n)$ *time on a CRCW PRAM. They use respectively* $n1/\log(n)$, $n2/\log(n)$, *and* $n3/\log(n)$ *processors. If* HRP1(x) *or* HRP2(x) *do not exist, then the number of processors is* $O(n/\log(n))$.

PROOF. We use the version of Vishkin's algorithm given on pp. 225–230 of [GR] and consider the pattern preprocessing phase. Vishkin's algorithm is technically quite involved. Instead of going into details we use the components of the algorithm. In one stage the procedure MakeSparse is applied to bigger and bigger prefixes of the pattern (the sizes double) until a first periodic prefix

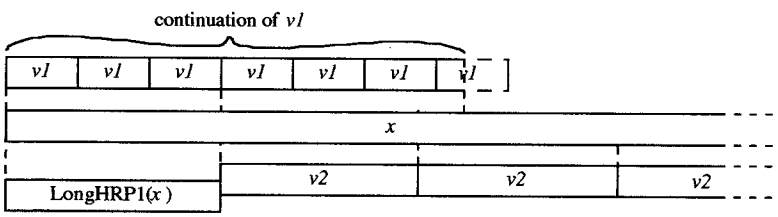

Fig. 21. The next HRP1 is double the length ($|v2| > 2 \cdot |v1|$).

is discovered. The prefix is periodic iff its period does not exceed half of its length.

If we discover the first prefix period $v1$, then we check whether it is an HRP1(x) and compute its continuation C (see Figure 21). This can be done using the procedure PeriodCont on p. 228 of [GR] and a kind of binary search. If we know the continuation C of the prefix period $v1$ and its size, then computing LongHRP1(x) is a simple matter.

The computation of HRP2(x) can be done essentially in the same way, the procedure MakeSparsePeriodic from [GR] can be used. The table of witnesses can be computed for the prefix C and the algorithm looks for the next supposed prefix period. We refer the reader to pp. 225–230 of [GR] for technical details.

The time complexity and the number of processors are of the same order as that of the whole string-searching algorithm of Vishkin, restricted to the part of the pattern which is inspected. Hence the total number of operations is proportional to the size of the computed objects, if they are nonempty. If they are empty, then the number of all operations is merely proportional to the total size n of the input pattern. This completes the proof. □

Using parallel versions of functions HRP1, HRP2, and LongHRP1 the parallel algorithm is a simple version of the sequential pattern preprocessing. Its correctness follows from the correctness of the sequential counterpart.

THEOREM 18. *The 4-perfect decomposition of a word x of length n can be found in $\log^2 n$ time with $n/\log^2 n$ processors of a CRCW PRAM or in $\log^3 n$ time with $n/\log^3 n$ processors of a CREW PRAM.*

PROOF. Let us analyze the Algorithm Parallel 4-perfect decomposition (Figure 22). We have a logarithmic number of positions induced by the parallel working sequence. Hence there are $O(\log n)$ stages in the algorithm. In one stage we compute several objects using a number of operations proportional to the size of computed objects. The computation of HRP1($x_{i+1} \cdots x_n$), LongHRP($x_{i+1} \cdots x_n$), and HRP2($x_{i+1} \cdots x_n$) on a CRCW PRAM takes logarithmic time. Due to Lemma 17 the total number of operations performed in one stage is proportional to the sizes of computed objects. However, the same analysis as that in the sequential case shows that the total size of these objects (HRP1s, LONGHRP1s, and HRP2s) computed in the algorithm is linear. Hence we have a parallel algorithm which works in $\log^2 n$ time and performs in total a linear number of operations. Due to

Algorithm Parallel 4-perfect decomposition
  $i := 0$; $hrp1 := |\text{HRP1}(x)|$; $longhrp1 := |\text{LongHRP1}(x)|$; $hrp2 := |\text{HRP2}(x)|$;
  while $longhrp1$ and $hrp2$ exist **do**
  { $i := i + longhrp1$; $hrp1 := \text{HRP1}(x_{i+1} \cdots x_n)|$; $longhrp1 := |\text{LongHRP1}(x_{i+1} \cdots x_n)|$;
    if $hrp1 \geq hrp2$ **then** $hrp2 := \text{HRP2}(x_{i+1} \cdots x_n)|$ }
  return the decomposition $(x_1 \cdots x_i, x_{i+1} \cdots x_n)$;
end.

Fig. 22. Parallel perfect decomposition.

Brent's theorem (see p. 12 of [GR]) the number of processors can be reduced to $n/\log^2 n$. Brent's theorem does not deal with the allocation of processors. However, it can be easily seen that our algorithm can use the same processor allocation as Vishkin's algorithm for string searching. In fact we use the same components as Vishkin's algorithm and even the structure of the whole algorithm is the same (processing exponentially increasing segments of the pattern).

In the CREW model the situation is analogous. We need another $\log(n)$ factor for the time complexity, similarly as in Vishkin's algorithm. The computation of boolean "or" and "and" now takes logarithmic time instead of constant time like in the CRCW model. Hence the algorithm finds the same 4-perfect decomposition as the sequential one. It does it within the claimed complexity bounds. This completes the proof. □

It would be interesting to know whether it is possible to reduce the time complexity by a logarithmic factor.

8. Final Remarks. Shifts in Algorithm SIMPLE-TEXT-SEARCH can be improved as shown by the following lemma. Let us reconsider first the scope of an HRP. Let v be a k-HRP of x and consider the longest prefix z of x which has prefix period v. We define *the extended scope* of v as the pair $([L, R], a)$ where $[L, R]$ is the scope of v ($L = |v^2|$, $R = |z|$) and a is the (unique) letter such that za has the prefix period v (za is not a prefix of x by the definition of z).

LEMMA 19 (*see Figure* 23). *Let* $(v_i/i = 1, \ldots, r)$ *be the sequence of k-HRPs of x. Also let* $(([L_i, R_i], a_i)/i = 1, \ldots, r)$ *be the corresponding sequence of extended scopes. Let u be a prefix of x, and let a be a letter such that ua is not a prefix of x. Then*

$$\text{per}(ua) = L_i/2 \quad \text{if } |u| \text{ is in } [L_i, R_i] \text{ for some } i \text{ and } a = a_i,$$

$$\text{per}(ua) > |ua| - L_i/2 \quad \text{if } |u| \text{ is in } [L_i, R_i] \text{ for some } i \text{ but } a \neq a_i,$$

$$\text{per}(ua) > |u|/k \quad \text{otherwise.}$$

Extended scopes can also be used to enlarge intervals $[L, R]$: when v is a k-HRP of x, we can define L as the length of the shortest prefix of x having v has the

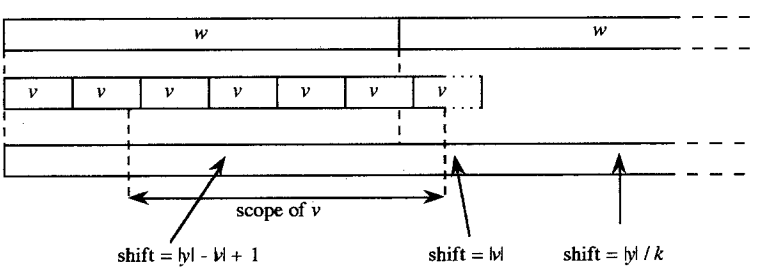

Fig. 23. Rule for shifts in scopes, at the end of a scope and outside a scope.

shortest prefix period. When doing so, the length of this period should be memorized together with the scope, because it is no longer $L/2$ as in previous sections.

Both points mentioned here lead to a more practical implementation of the algorithms of this paper, but have obviously no influence on the asymptotic time complexities.

Acknowledgments. We thank the referees for their interesting questions concerning this work. The discussion on a possible relation between the $\log_\Phi n$ bound for the number of 2-HRPs of a string and the length of its sequence of borders considered in the algorithm of [KMP] has not reached an end. We believe that such a relation possibly exists for words having a large (according to their length) number of 2-HRPs.

References

[A] A. V. Aho, Algorithms for finding patterns in strings, in: J. van Leeuwen, ed. *Handbook of Theoretical Computer Science*, vol. A, Elsevier, Amsterdam, 1990, pp. 255–300.

[C] M. Crochemore, String-matching on ordered alphabets, *Theoret. Comput. Sci.* **92** (1992), 33–47.

[CP] M. Crochemore and D. Perrin, Two-way string-matching, *J. Assoc. Comput. Mach.* **38**(3) (1991), 651–675.

[GS1] Z. Galil and J. Seiferas, Saving space in fast string matching, *SIAM J. Comput.* **9** (1980), 417–438.

[GS2] Z. Galil and J. Seiferas, Time–space optimal string matching, *J. Comput. System Sci.* **26** (1983), 280–294.

[GR] A. Gibbons and W. Rytter, *Efficient Parallel Algorithms*, Cambridge University Press, Cambridge, 1988.

[KMP] D. E. Knuth, J. H. Morris, Jr., and V. R. Pratt, Fast pattern matching in strings, *SIAM J. Comput.* **6** (1977), 323–350.

[L] M. Lothaire, *Combinatorics on Words*, Addison-Wesley, Reading, MA, 1983.

[S] I. Simon, Personal communication, 1989.

[V] U. Vishkin, Optimal parallel pattern matching in strings, *Inform. and Control* **67** (1985), 91–113.