The zooming method: a recursive approach to time-space efficient string-matching *

Leszek Gąsieniec Wojciech Plandowski Wojciech Rytter

Instytut Informatyki UW, Banacha 2, 02-097 Warszawa, Poland

Abstract

A new approach to time-space efficient string-matching is presented. The method is flexible, its implementation depends whether or not the alphabet is linearly ordered. The only known linear-time constant-space algorithm for string-matching over nonordered alphabets is the Galil-Seiferas algorithm, see [8, 6] which is rather complicated. The zooming method gives probably the simplest string-matching algorithm working in constant space and linear time for nonordered alphabets. The novel feature of our algorithm is the application of the searching phase (which is usually simpler than preprocessing) in the preprocessing phase. The preprocessing has a recursive structure similar to selection in linear time, see [1]. For ordered alphabets the preprocessing part is much simpler, its basic component is a simple and well-known algorithm for finding the maximal suffix, see [7]. Hence we demonstrate a new application of this algorithm, see also [5]. The idea of the zooming method was applied in [4] to two dimensional patterns.

1 Introduction

The pattern P of length m and the text T of length n are given as *read-only* tables of symbols. The string-matching problem consists in finding all occurrences of P in T. By the space complexity we mean additional memory (we do not count the space occupied by P and T). Constant space means a constant number of small (with logarithmic number of bits) integers. The sequential string-matching algorithm is *time-space optimal* if it works simultaneously in linear time and constant space. Presently, there are known three different time-space optimal string-matching algorithms, see [8, 5, 3]. The first one works in the "classical" model where the only information about strings is by checking equality of symbols. The alphabet is a set without any additional structure (e.g. linear order). The other two algorithms use comparisons of the symbols with

*Supported by the grant KBN 2-1190-91-01

respect to some linear order, thus they do not work in the classical model. In this paper we produce the fourth algorithm, which can be implemented in the classical model. However if the alphabet is ordered then our algorithm can be simplified.

Our strategy is to consider a sequence of nonperiodic subpatterns whose lengths form a decreasing geometric sequence of integers (modulo floors). We check their occurrences naively starting from the smallest one. Searching for a nonperiodic pattern P is followed by a match of its nonperiodic subpattern P'. If a mismatch occurs, the subpattern P' is sufficiently large to guarantee a long shift which amortizes the work done in symbol comparisons.

Denote by $|w|$ the length of the word w. The number p is *a period* of the word w if $w[i+p] = w[i]$ for $1 \leq i \leq |w| - p$. Denote by $per(P)$ the shortest period of P. We use a weaker version of the so called *periodicity lemma* which is the main tool in many advanced string-matching algorithms.

Lemma 1 *If u and w are periods of a word x and $|u|+|w|-1 \leq |x|$ then x has a period of size $gcd(|u|,|w|)$), where gcd stands for the greatest common divisor.*

We say that the pattern is *periodic* iff $per(P) \leq \frac{1}{6}|P|$. To simplify the notation we write cn instead of $\lfloor cn \rfloor$. The constant $\frac{1}{6}$ is important in the preprocessing phase for nonordered alphabets to simplify the procedure $Next$.

For a nonperiodic pattern P the sequence of subpatterns $ZoomSeq(P)$ is defined by
$ZoomSeq(P) = (P_1, P_2, \ldots, P_k)$, where $P_1 = P$, $|P_k| = 1$ and for $1 \leq j < k$ P_{j+1} is a nonperiodic prefix or suffix of P_j of length $\frac{3}{4}|P_j|$ (if both the suffix and the prefix are nonperiodic we take the prefix). The series $ZoomSeq(P)$ is called the *zooming sequence* of P. Its compressed representation is a sequence of $k-1$ bits. The j-th bit is 0 iff P_{j+1} is the prefix of P_j and the j-th bit is 1 iff P_{j+1} is the suffix of P_j. In this way $ZoomSeq(P)$ is stored as one integer with logarithmic number of bits.

Example 2 The zooming sequence for $P = a^{12}b^5$ looks as follows

$$ZoomSeq(P) \quad = \quad (\ P,\ a^7b^5,\ a^7b^2,\ a^4b^2,\ a^4,\ a^3,\ a^2,\ a)$$

and the compressed representation for it is 1011111.

Assume, now that P may be periodic. Denote by $sub(P)$ the set consisting of the prefix and the suffix of P of length $\frac{3}{4}|P|$. Let $(f_1, f_2, \ldots)$ be the sequence of integers satisfying $f_1 = |P|$ and $f_t = \frac{3}{4}f_{t-1}$ for $t > 1$. Denote by $head(P)$ the longest nonperiodic prefix of P whose length is in the sequence. Clearly, $head(P) = P$ for nonperiodic P.

Lemma 3 (key lemma)
a) If P is nonperiodic then there is a nonperiodic subword in $sub(P)$.
b) If P is periodic then $per(head(P)) = per(P)$.

Proof:

a) Let $n = |P|$. Since words shorter than 6 are nonperiodic we may assume $n \geq 6$. Suppose, that both subwords in $sub(P)$ are periodic. They have a large overlap of size $2\lfloor\frac{3}{4}n\rfloor - n \geq 2(\frac{3}{4}n - 1) - n = \frac{1}{2}n - 2$. On the other hand their periods are not longer than $\lfloor\frac{1}{6}\lfloor\frac{3}{4}n\rfloor\rfloor \leq \frac{1}{6}\frac{3}{4}n = \frac{1}{8}n$. Since periods of words in $sub(P)$ are periods of the overlap and $\frac{1}{8}n + \frac{1}{8}n - 1 \leq \frac{1}{2}n - 2$ for $n \geq 4$ Lemma 1 guarantees that the shortest periods of the words in $sub(P)$ are the same. This contradicts the nonperiodicity of P.

b) It is enough to prove that if P is periodic then $per(P') = per(P)$ where P' is the prefix of P of length $\frac{3}{4}|P|$. If $per(P') \neq per(P)$ then $per(P') < per(P)$ and P' has two different periods $per(P')$, $per(P)$. Since

$$per(P') + per(P) - 1 \leq \lfloor\frac{1}{6}|P|\rfloor + \lfloor\frac{1}{6}|P|\rfloor - 1 \leq \frac{2}{3}|P| - 1 \leq |P'|$$

Lemma 1 becomes applicable and $per(P')$ is a period of P. A contradiction. □

For a nonperiodic pattern P the zooming sequence $ZoomSeq(P) = (P_1, \ldots, P_k)$ is constructed as follows:
$P_1 = P$ and for each $1 \leq l < k$ the word P_{l+1} is a nonperiodic element of $sub(P_l)$ (if both are nonperiodic take the prefix).

Similar to other pattern-matching algorithms, the preprocessing phase is more involved than the searching one. The preprocessing consists of two parts:

1. check if P is periodic and if it is compute $per(P)$,
2. find $ZoomSeq(head(P))$.

Thus the goal of the preprocessing phase is to compute the pair $preprocess(P) = (quasiper(P),\ ZoomSeq(head(P)))$ where $quasiper(P) = per(P)$ if P is periodic and $quasiper(P) = |P|$ otherwise.

The preprocessing algorithm for nonordered alphabets is simple due to two features of our preprocessing:

- it has a recursive structure,
- searching phase is a basic component in the preprocessing.

2 Searching phase

2.1 Searching phase for nonperiodic patterns

We deal first with nonperiodic patterns. Denote by $Partial\_Match(i, P_l)$ the function which in a given text T for $l > 0$ checks if the pattern P placed at a (starting position) i in T agrees with T on the subpattern P_l. The function works in a naive way.

Observation 4 *If $Partial\_Match(i, P_l) = true$ then there is no occurrence of the pattern strictly between positions i and $i + \frac{1}{6}|P_l|$.*

The observation follows from the nonperiodicity of subpatterns in the zooming sequence. Assume, for technical reasons, that $|P_{k+1}| = 0$. In the algorithm below m is the length of the pattern P, n is the length of the text; the pattern is nonperiodic and the sequence $ZoomSeq(P) = (P_1, \ldots P_k)$ is precomputed.

```
ALGORITHM  Text_Searching_By_Zooming;
begin
 i:= 1;
 while i ≤ n − m do
 begin
    l:=k + 1;
    while l > 1 and Partial_Match(i, P_{l−1}) do l:= l − 1;
    if l = 1 then  {P_l = P} report full match at i;
    Shift:= max(1, (1/6)|P_l|);
    i:= i + Shift;
 end;
end.
```

Our algorithm checks if there is an occurrence of the pattern at position i in the text by checking occurrences of words from the zooming sequence. Since it analyzes the zooming sequence from the shortest words to the longest ones, we need a method to find a subword P_l on the basis of P_{l+1} in constant space and time. We store the word P_l as the pair: the starting position of P_l in the pattern and the length of P_l. The compressed representation of $ZoomSeq(P)$ allows to find out if P_{l+1} is the prefix or the suffix of P_l. It remains to find the length of P_l. Since $|P_{l+1}| = \lfloor \frac{3}{4}|P_l| \rfloor$ we know that $|P_l| = \lceil \frac{4}{3}|P_{l+1}| \rceil + b_l$ where $b_l = 0$ or $b_l = 1$. To find appriopriate length we store an additional $k-1$-length bit sequence whose l-th element equals b_l. This sequence can be easily computed in logarithmic time in the preprocessing phase on the basis of $|P|$ and the compressed representation of $ZoomSeq(P)$.

Lemma 5 *The algorithm* Text_Searching_By_Zooming *is time-space optimal if $preprocess(P)$ has been already computed.*

Proof: The linearity of the algorithm is clear since the number of comparisons done during every execution of the main iteration is proportional to the shift done at the end of the iteration. □

2.2 Searching phase for periodic patterns

Assume, the pattern P is periodic. Then $quasiper(P) = per(P)$ and the preprocessing phase for P computes the zooming sequence for $head(P)$ and the length of the shortest period of P . The algorithm for periodic patterns searches for $head(P)$ using the algorithm *Text_Searching_By_Zooming*. As it finds $head(P)$ at position i in the text, starting from the position $i + head(P)$, it searches for the continuation of the period $per(P)$ from $head(P)$ in the text. Additionally, it reports occurrences of the pattern when necessary. In case the period is broken at position $i + t$, the algorithm does the shift equal to $t/6$. We can do such a shift since the word which starts at position i and ends at position $i + t$ in the text is nonperiodic.

Lemma 6 *The algorithm for periodic patterns finds all occurrences of the pattern in linear time and constant space.*

Proof: In the algorithm *Text_Searching_By_Zooming* the shift is proportional to the work done just before. As we find $head(P)$ the total work done during finding $head(P)$ and searching for the continuation of the period is also proportional to the shift we do next. □

3 Preprocessing phase for ordered alphabets

The Crochemore-Perrin version of Duval's algorithm, see [7], is ideally suited to the preprocessing in the zooming method. The Duval's algorithm was originally related to some algebraic properties of words, see [10]. Then it was simplified, see page 668 in [5], where it is presented as a nonrecursive function *Maximal_Suffix*. Denote by $max(P)$ lexicographically maximal suffix of P. The algorithm *Maximal_Suffix* computes $max(P)$ and, as a side effect, the smallest period of $max(P)$.

If the pattern P is periodic then denote by $period(P)$ the prefix u of P of size $per(P)$. Then P is of the form $u^r v$ for some v, possibly empty word, which is a prefix of u. Denote such v by $tail(P)$.

The algorithm *Maximal_Suffix* is based on the following three observations proved in [5]. The only nontrivial point is the point (a).

Observation 7 *Denote* $j = |tail(x)|$. *Let* $<$ *means here linear order in the alphabet. Assume* $max(x) = x$. *Then there are three cases depending on how the next symbol affects continuation of periodicity of* x*:*

(a) *if* $a < x[j+1]$ *then* $per(xa) = |xa|$, $max(xa) = xa$ *and* $tail(xa) = \epsilon$*;*
(b) *if* $a = x[j+1]$ *then* $per(xa) = per(x)$, $max(xa) = xa$ *and* $tail(xa) = tail(x)a$*;*
(c) *if* $a > x[j+1]$ *then* $per(xa) = per(tail(x)a)$, $max(xa) = max(tail(x)a)$ *and* $tail(xa) = tail(tail(x)a)$.

```
function  Maximal_Suffix(P);  {|P| = m}
begin
ms := 1; j := 0; p := 1; i := 2;
while i ≤ m do
  {j = |tail(P[ms..i − 1])|, p = |per(P[ms..i − 1])|,
  P[ms..i − 1] = max(P[1..i − 1])}
    case
    P[i] < P[ms + j]: {follow Observation 7 (a)}
                     j := 0; p := i − ms; i := i + 1
    P[i] = P[ms + j]: {follow Observation 7 (b)}
                     j := j + 1; i := i + 1
    else: {P[i] > P[ms + j], follow Observation 7 (c)}
                      ms := i − j; j := 0; p := 1; i := ms + 1
    endcase
return ms, p
```

The algorithm *Maximal_Suffix* is on-line, it scans the pattern from left to right and keeps the position ms of the maximal suffix of the current prefix of P and the smallest period p of this suffix. We refer the reader to [5], page 668, for the detailed description of this algorithm and the (implicit) proof of point (a) of the lemma below.

Lemma 8 *Let* $m = |P|$.
(a) *The algorithm Maximal_Suffix computes* $max(P)$ *and* $per(max(P))$ *using at most* $2m$ *comparisons.*
(b) $quasiper(P)$ *and* $head(P)$ *can be computed in linear time and constant space using at most* $2\frac{1}{6}m$ *comparisons.*

Proof: The following observation helps us in proving part b).

Observation 9 *Let* ms *be the starting position of* $max(P)$ *in* P*. If* P *is periodic then* $per(P) = per(max(P))$ *and* $ms < |per(P)|$.

First, we use the algorithm Maximal_Suffix to compute $max(P)$ and the starting position ms of $max(P)$ in P. By the observation above, if $ms \geq \frac{1}{6}m$ then the pattern is nonperiodic. Otherwise, we check naively whether or not the $ms - 1$ length prefix of P breaks periodicity of $max(P)$. This allows to compute $quasiper(P)$ in the claimed number of comparisons. By Lemma 3, if P is periodic then $per(head(P)) = quasiper(P)$. To find $head(P)$ we consider consecutive prefixes of P of lengths from the sequence f_s and find the first one which is nonperiodic. It does not require additional comparisons. $\square$

It remains to show how to compute $ZoomSeq(P)$ for nonperiodic patterns. Let $next(P)$ be a nonperiodic element of $sub(P)$, if P is nonperiodic, and the prefix of P of size $\frac{3}{4}|P|$, otherwise. The zooming sequence can be constructed in an iterative way due to the following observation.

Observation 10 *If P is nonperiodic then*

$$ZoomSeq(P) \quad = \quad P \diamond ZoomSeq(next(P)) \tag{1}$$

where $\diamond$ denotes a concatenation of one element and a sequence of elements.

The function below computes $ZoomSeq(P)$ for a nonperiodic pattern P.

function $Zooming\_Sequence\_1(P)$; $\{P$ is nonperiodic $\}$
begin
if $|P| = 1$ **then return** P
else begin
    $P_0 :=$ prefix of P of size $\frac{3}{4}|P|$;
    compute $quasiper(P_0)$ by the algorithm *Maximal_Suffix*;
    if $quasiper(P_0) = |P_0|$ **then** $next := P_0$
    else $next :=$ suffix of P of size $\frac{3}{4}|P|$;
    return $P \diamond Zooming\_Sequence\_1(next)$,
    end
end

Theorem 11 *The preprocessing phase for ordered alphabets can be done in linear time and constant space with at most $8\frac{2}{3}n$ comparisons.*

Proof: We use $Maximal\_Suffix$ for the whole pattern P to compute $head(P)$ and $quasiper(P)$. It costs at most $2\frac{1}{6}n$ comparisons. Then we use the function $Zooming\_Sequence\_1$ to compute $ZoomSeq(head(P))$. In the worst case $head(P) = P$ and during the computation of the zooming sequence for P the procedure $Maximal\_Suffix$ is applied to all elements in $ZoomSeq(P)$ except P. Each element of the sequence is $\frac{3}{4}$ times smaller than the preceding one. Processing one element of length l by *Maximal_Suffix* requires at most $2\frac{1}{6}l$ comparisons. We have a power series which is bounded by $2\frac{1}{6} \cdot \frac{3}{4}n \cdot \frac{1}{1-3/4}$. Summing with the cost of computing $head(P)$ and $quasiper(P)$ we obtain the claimed estimation. $\square$

4 Preprocessing phase for nonordered alphabets

Let P be the pattern to preprocess. The preprocessing part computes the pair $(quasiper(P), ZoomSeq(head(P))$. Having $ZoomSeq(head(P))$ we can easily find $quasiper(P)$ by searching the second from the left occurrence of $head(P)$ in P. Since $head(P)$ is nonperiodic this can be done using the algorithm $Text\_Searching\_By\_Zooming$. The sequence $ZoomSeq(head(P))$ is computed by the procedure $Zooming\_Sequence\_2$. It has a recursive structure which is similar to the structure of the computation of the median.

Denote by $Next(P, P_0)$ the function computing $quasiper(P)$ and $next(P)$ assuming $preprocess(P_0)$ has been already computed.

Lemma 12 *Assume P_0 is the prefix of P of length $|P_0| = \frac{1}{5}|P|$. Then we can compute $Next(P, P_0)$ in linear time and constant space.*

Proof: We consider two cases.
Case 1: P_0 is nonperiodic.
We find all occurrences of P_0 in P by the algorithm $Text\_Searching\_By\_Zooming$. There is a constant number of positions which start an occurrence of P_0 in P. Each of them corresponds to a potential small period of P. The potential periods are checked (if they are real periods) naively, each one in linear time and constant space. Afterwards we know whether the whole pattern and its prefix of size $\frac{3}{4}|P|$ are periodic. This determines $quasiper(P)$ and $next(P)$.
Case 2: P_0 is periodic.
We check the continuation of the period $per(P_0)$ in the whole pattern. If it continues till the end then $quasiper(P) = quasiper(P_0)$. Otherwise, it can be easily proved (in the proof the constant $\frac{1}{6}$ from the definition of the periodic words is important) that the pattern is nonperiodic and $next(P)$ is the prefix in $sub(P)$ iff the period of P_0 breaks inside this prefix. □

Due to equation (1) we can recursively preprocess the pattern using the function $Next(P, P_0)$. The algorithm has the structure quite similar to the algorithm for selection given in [1].

```
function Zooming_Sequence_2(P); {|P| = n}
begin
if n = 1 then return P
else begin
    P_0 := prefix of P of size (1/5)n;
    ZoomSeq(P_0) := Zooming_Sequence(P_0); {step 1}
    compute quasiper(P_0) using ZoomSeq(P_0)
    next(P):= Next(P, P_0)
    if P is nonperiodic then
        return P ◇ Zooming_Sequence_2(next(P)); {step 2}
    else
        return Zooming_Sequence_2(next(P)) {step 3}
    end
end
```

Theorem 13 *The preprocessing phase can be done in linear time and constant space.*

Proof: Let $n = |P|$. Observe, that $|P_0| = \frac{1}{5}n$ and $|next(P)| = \frac{3}{4}n$. Denote by $T(n)$ be the time complexity of $Zooming\_Sequence\_2(P)$. Due to Lemma 12 and equation (1), $T(n)$ satisfies the following recurrence:

$$T(n) \leq T(\frac{3}{4}n) + T(\frac{1}{5}n) + O(n).$$

It is the same recurrence equation as the one related to the complexity of selection and presented in [1]. The solution to the recurrence is $T(n) = O(n)$.

The recurrence stack from the algorithm can be encoded by a logarithm length sequence of numbers {1,2,2',3}. A number from the top of the stack means how the current procedure was called. 1 means that it was called in step 1 of the algorithm, 2 means that it was called in step 2 with a prefix as a parameter, 2' means that it was called in step 2 with a suffix as a parameter and 3 means that it was called in step 3. Additionally, we store a stack which keeps for every level of a recurrence a few bits determining how to retrieve the length of a parameter of the procedure which called the current procedure on the basis of the length of the current parameter and the top of the first stack.

This completes the proof. □

5 Improving the worst case performance of the searching phase to $(2 + \varepsilon)n$

In this section we present an improved implementation of the searching phase. We give a family of algorithms $\{A_s\}_{1<s<2}$ such that the worst case performance of the algorithm A_s is $(2 + \varepsilon(s))n$ where n is the length of the text and $\lim_{s\to 1}\varepsilon(s) = 0$. The improvement however require slight changes in our previous definitions.

For each $1 < s < 2$ define the sequence of natural numbers $\{f_t^s\}$ satisfying the recurrence formulae f_0^s is the minimal natural number x such that $\lfloor sx \rfloor > x$, (one can calculate that $f_0^s = \lceil \frac{1}{s-1} \rceil$) and $f_{t+1}^s = \lfloor s f_t^s \rfloor$. The sequences have a nice feature that on the basis of each element of the sequence it is easy to compute the previous one and $f_t^s = \lceil \frac{f_{t+1}^s}{s} \rceil$. In further considerations we omit the index s in f_t^s assuming that s is fixed.

The sequences are used to modify the definition of the zooming sequence as follows: Let k be a natural number such that $f_{k-2} < |P| \leq f_{k-1}$. For any nonperiodic pattern P, $ZoomSeq(P) = (P_1, \ldots, P_k)$ where $P_1 = P$ and $|P_i| = f_{k-i}^s$ for $i > 1$ and as previously P_{i+1} is the nonperiodic suffix or prefix of P_i (if both are nonperiodic we take the prefix). Note, that (with this definition of the zooming sequence) there are no problems to calculate $|P_i|$ on the basis of $|P_{i+1}|$.

Change other definitions as follows. The pattern P is periodic if $per(P) \leq \lceil p|P| \rceil$ where $p = 1 - \frac{s}{2}$. The set $sub(P)$ consists of the prefix and the suffix of sizes f_k where $f_k < |P| \leq f_{k+1}$. We define $head(P)$ to be the longest nonperiodic prefix of P whose length is in the sequence f_n.

Example 14 Let $s = 4/3$ and $P = a^{12}b^5$. Then $p = 1/3$, first elements of the sequence f_t are 3, 4, 6, 8, 11, 15, 20 ... and the zooming sequence for P looks as follows.

$$ZoomSeq(P) \quad = \quad (\,P,\ a^{12}b^3,\ a^8b^3,\ a^5b^3,\ a^5b,\ a^3b,\ a^2b)$$

Under the modified definitions the key lemma is easily restated as follows. Its proof is similar to the proof of the key lemma.

Lemma 15 (modified key lemma)
a) If P is nonperiodic then one of the words in $sub(P)$ is nonperiodic.
b) If P is periodic then $per(P) = per(head(P))$.

Denote by $KMP(i, P')$ the function which starting from i in the text returns the first to the right occurrence of the pattern P'. Additionally, we assume that KMP uses any algorithm which scans the text from left to right and such that finding the first occurrence of the pattern costs at most $2(i-1) + m'$ symbol comparisons where i is the starting position of the pattern in the text and m' is the length of the pattern. Moreover, the algorithm has to use constant size additional memory for constant size patterns. One of the algorithms with these properties is the well-known Knuth-Morris-Pratt algorithm, see [9].

The improved algorithm (presented below) for nonperiodic patterns is a slight modification of the algorithm $Text\_Searching\_By\_Zooming$. Assume, that $ZoomSeq(P) = (P_1, \ldots P_k)$ is precomputed.

```
ALGORITHM Improved_Text_Searching_By_Zooming;
{ r - the starting position of P_k in P }
begin
 i:= 1;
 while i ≤ n − m do
 begin
    i:=KMP(i + r − 1, P_k) − (r − 1); l:=k − 1;
    while l ≥ 1 and Partial_Match(i, P_l) do l:= l − 1;
    if l = 0 then {P_{l+1} = P } report full match at i;
    i:= i + ⌈p|P_{l+1}|⌉;
 end;
end.
```

In the algorithm we may assume that the procedure $Partial\_Match(i, P_l)$ do not compare the symbols from P_{l+1} because the previous calls of $Partial\_Match$ have already done it.

The algorithm for periodic patterns is almost the same as the one from Section 2.2. The only difference is that, now since the definition of periodic words is changed, the shift is changed from $t/6$ to $\lceil p \cdot t \rceil$.

Theorem 16
a) The worst case performance of the modified algorithm for nonperiodic patterns is $(2 + \varepsilon(s))n$ *symbol comparisons where* $\lim_{s\to 1} \varepsilon(s) = 0$.
b) The algorithm for periodic patterns makes at most $(2 + \varepsilon(s))n$ *symbol comparisons where* $\lim_{s\to 1} \varepsilon(s) = 0$.

Proof:
a) The main informal idea of the proof is that the number of comparisons made after each execution of the main iteration is proportional to the shift made after this execution with the constant which is close to 2 when s is close to 1. This is caused by the fact that for values of s close to 1 a *short* period means a period of a size close to half of the considered subpattern, hence for nonperiodic subpatterns the shift is very close to half of the scanned subpattern. The total work is amortized by twice the total sum of all shifts (which correspond to disjoint subintervals, the sum of these subintervals is at most n).

More precisely, suppose the main iteration is executed exactly t times. Let i_j be the value of the variable i after the j-th execution of the main iteration and i'_j the value of i returned by the function KMP in this execution. Clearly, $i_0 = 1$ and $i_t \leq n$. Let q_j be the total number of symbol comparisons made in the j-th iteration and q'_j be the total number of comparisons done in all operations $Partial\_Match(i, P_l)$ in the j-th iteration.

Assume, that the subword P_{l+1} of length m_j matches the text and P_l does not or the pattern matches the text ($m_j = m$). Then $i_j - i'_j = \lceil pm_j \rceil$ and $q'_j \leq \lfloor sm_j \rfloor - |P_k|$. We have

$$q_j \leq (2(i'_j - i_{j-1}) + |P_k|) + q'_j \leq 2(i'_j - i_{j-1}) + s/p(i_j - i'_j)$$

and, since $s/p = 2s/(2-s) \geq 2$ we obtain

$$q_j \leq 2s/(2-s)(i_j - i_{j-1}) \quad .$$

Summing over all iterations, the total number of comparisons does not exceed $2s/(2-s)(i_t - i_0) \leq 2s/(2-s)n$. Since $\lim_{s\to 1} 2s/(2-s) = 2$ the result follows.

b) The proof is similar as the proof of point a). When $head(P)$ is found we take $m_j = t$ and the rest of the analysis is the same. □

Remark. The preprocessing phase for ordered alphabets is the same as previously. The preprocessing phase for general alphabets should be changed in the following way. Let r be the number such that $f_{t-r} + f_{t-1} < cf_t$ for all $t \geq r$ and where $c < 1$ is a constant. Recall, that k is such that $f_k < |P| \leq f_{k+1}$. Then as P_0 we take the prefix of P of size f_{k+1-r} and the $next(P)$ is the prefix or the suffix of P of length f_k. Now the preprocessing remains linear since the solution of the recurrence

$$T(f_t) = T(f_{t-r}) + T(f_{t-1}) + O(f_t)$$

where $f_{t-r} + f_{t-1} < c \cdot f_t$ for a constant $c < 1$ is $T(f_t) = O(f_t)$.

Remark. The zooming method can be extended to the 2-dimensional pattern matching, however this is much more complicated due to the complicated structure of 2-dimensional periodicities, see [4]

References

[1] J.Aho, J.Hopcroft, J.Ullman, "The design and analysis of computer algorithms", Addison-Wesley, 1974.

[2] D.Breslauer, Saving comparisons in the Crochemore-Perrin string matching algorithm, *in* Proceeding of ESA'93.

[3] M.Crochemore, String matching for ordered alphabets, *TCS* **92** (1992) 225–251.

[4] M.Crochemore, L. Gąsieniec, W.Plandowski, W.Rytter, Time-space efficient searching of 2-dimensional patterns, manuscript

[5] M.Crochemore, D.Perrin, Two-way string matching, *JACM* **38:3** (1991), 651–675.

[6] M.Crochemore, W.Rytter, Periodic prefixes in texts, *in* Sequences II, (ed. R.Capocelli, A.de Santis, U.Vacarro), Springer Verlag, 1993, 153–165.

[7] J.Duval, Factorizing words over an ordered alphabet, *J. Algorithms* **4** (1983) 363–381.

[8] Z.Galil, J.Seiferas, Time-space optimal string matching, *JCSS* **26** (1983) 280–294.

[9] D. Knuth, J. Morris, V. Pratt, Fast pattern matching in strings, *SIAM Journal on Algebraic and Discrete Methods*, **6:2**(1977), 323–350.

[10] M.Lothaire, "Combinatorics on words", Addison-Wesley, Reading, 1983.