INFORMATION AND COMPUTATION 95, 225–251 (1991)

Correctness and Efficiency of Pattern Matching Algorithms

LIVIO COLUSSI

Dipartimento di Matematica Pura ed Applicata, Università di Padova, Italy

A few lines pattern matching algorithm is obtained by using the correctness proof of programs as a tool to the design of efficient algorithms. The new algorithm is obtained from a brute force algorithm by three refinement steps. The first step leads to the algorithm of Knuth, Morris, and Pratt that performs $2n$ character comparisons in the worst case and $(1+\alpha)n$ comparisons in the average case $(0<\alpha\leqslant 0.5)$. Two more steps give a faster algorithm that performs $1.5n$ character comparisons in the worst case and is sublinear on a random text for all patterns. Moreover, those bounds are less than the corresponding bounds of the Boyer and Moore algorithm because the Boyer and Moore algorithm performs more than $2n$ character comparisons in the worst case and because there exist some patterns that require more than n character comparisons on a random text. However, if we consider the average on all the patterns of a given length, then on a random text the Boyer and Moore algorithm is sublinear too, with better performance the longer the pattern gets.

INTRODUCTION

It is a widely accepted opinion that program correctness and program efficiency should be considered as two unrelated aspects of the program development process (Alagić and Arbib, 1978; Dijkstra, 1976). However, in our experience on formal verification of programs using the Hoare axiomatic semantics (Alagić and Arbib, 1978; DeBakker, 1980; Hoare and Wirth, 1973), we noted many times that a careful analysis of program correctness proofs can indeed lead to gains in efficiency and we hope that this paper gives some evidence of this. Indeed, we use the correctness proof of programs as a tool to stepwise refine a brute force algorithm for the pattern matching problem

> "Given a pattern $w=w_0w_1\cdots w_{m-1}$ and a text $f=f_0f_1\cdots f_{n-1}$, test if there is a match of the pattern into the text and, if such a match exists, give the first position in f where the match occurs"

until we obtain a very efficient algorithm that performs $1.5n$ character comparisons in the worst case and is sublinear on a random text for all pattern.

0890-5401/91 $3.00

Moreover, a slightly modified version of the algorithm that solves the problem of finding all occurences of the pattern in the text is given.

We obtain such an algorithm starting from a simple but inefficient naive algorithm and then repeatedly improving it by applying the following strategy:

— write down a formal correctness proof of the algorithm (also in cases where the correctness is straightforward);

— look for the presence in the algorithm of some "true" null statement (where "true" means that, in its correctness proof

$$\frac{P \supset Q}{\{P\}\{Q\}},$$

$P \equiv Q$ does not hold);

— whenever any such statement is found, perform an analysis of the information this statement throws away;

— use the results of this analysis either to avoid the computational effort required to synthesize the information not used or to exploit that information to lower the subsequent computational effort needed to attain the final result.

To compare the new algothm with the classical algorithms of Knuth, Morris, and Pratt (KMP) (Knuth, Morris, and Pratt, 1977) and of Boyer and Moore (BM) (Boyer and Moore, 1977) we recall that, in worst case, KMP requires $2n$ character comparisons while in Guibas and Odlyzko (Guibas and Odlyzko, 1980) it is proved that the worst case bound for BM lies between $2n$ and $4n$ and it is conjectured that the true bound is $2n$. We also recall that, on the average, KMP requires $(1+\alpha)n$ character comparisons on a random text, where the coefficient $0 < \alpha \leqslant 0.5$ depends on the cardinality of the alphabet and on the relative frequency of the characters, and that BM is sublinear on the average on a random text with better performance the longer the pattern gets. However, BM is superlinear on a random text for some pattern (see note *d* about the results of the test cases reported in Table I of Section 3 below).

The paper is organized as follows: In Section 1 we describe the first refinement step that leads to Algorithm 2, a slightly improved version of the KMP algorithm. Section 2 contains two more refinement steps that lead to the new algorithm (Algorithm 4). Section 2 also contains the proofs of two theorems and of a lemma that are needed to prove that Algorithm 4 and its previous version, Algorithm 3, are indeed correct. Section 3 contains the result of some comparative tests while in Section 4 the preprocessing algorithms are reported. In Section 5 we prove that the worst case time bound of Algorithm 4 is $1.5n + 0.5(m-1)$.

1. Deriving KMP via the Correctness Proof of a Brute Force Algorithm

The following algorithm is a straightforward solution of the pattern matching problem stated in the introduction:

begin
$b := 0$;
repeat
$i := 0$;
while $w_i = f_{b+i}$ **and** $i < m$ **do** $i := i + 1$;
match $:= i = m$;
if not match **then** $b := b + 1$
until match **or** $b > n - m$
end

If the text does not contain a match of the pattern then the algorithm terminates with false as the value of the boolean variable match, otherwise it terminates with match = true and in such a case the integer variable b holds the first position where the pattern matches the text. This algorithm is so simple that a formal verification of its correctness is hardly required. However, we develop such a verification anyway.

To improve the readability of the correctness proof we use Mch(b) as an abbreviation for the assertion "the pattern matches the text in position b," NMchU(b) as an abbreviation for the assertion "the pattern does not match the text up to position b," and Eq(b, i) as an abbreviation for the assertion "the first i characters of the pattern match the text starting at position b." Formally,

$$\text{Mch}(b) \underset{\text{def}}{\equiv} f_b \cdots f_{b+m-1} = w_0 \cdots w_{m-1},$$
$$\text{NMchU}(b) \underset{\text{def}}{\equiv} \forall q((1 \leqslant q \leqslant b) \supset \textbf{not}\ \text{Mch}(q)),$$
$$\text{Eq}(b, i) \underset{\text{def}}{\equiv} f_b \cdots f_{b+i-1} = w_0 \cdots w_{i-1}.$$

The pattern matching problem can be formally specified by the precondition

$$P \underset{\text{def}}{\equiv} (f = f_0 f_1 \cdots f_{n-1}\ \textbf{and}\ w = w_0 w_1 \cdots w_{m-1}\ \textbf{and}\ n \geqslant m)$$

and the postcondition

$$Q \underset{\text{def}}{\equiv} (\text{match}\ \textbf{and}\ \text{Mch}(b)\ \textbf{and}\ \text{NMchU}(b-1))\ \textbf{or}$$
$$(\textbf{not}\ \text{match}\ \textbf{and}\ \text{NMchU}(n-m)).$$

The algorithm with the assertions needed for a formal verification of its correctness is

ALGORITHM 1.

begin $\{f = f_0 f_1 \cdots f_{n-1}$ **and** $w = w_0 w_1 \cdots w_{m-1}$ **and** $n \geqslant m\}$
$b := 0$;
repeat $\{\mathrm{NMchU}(b-1)\}$
$i := 0$;
while $w_i = f_{b+i}$ **and** $i < m$ **do** $i := i+1$;
$\{(\mathrm{NMchU}(b-1)$ **and** $i = m$ **and** $\mathrm{Eq}(b, i))$ **or**
$(\mathrm{NMchU}(b-1)$ **and** $i < m$ **and** $\mathrm{Eq}(b, i)$ **and** $f_{b+i} \neq w_i)\}$
match $:= i = m$;
$\{($match **and** $\mathrm{NMchU}(b-1)$ **and** $\mathrm{Mch}(b))$ **or**
(**not** match **and** $\mathrm{NMchU}(b-1)$ **and** $\mathrm{Eq}(b, i)$ **and** $f_{b+i} \neq w_i)\}$ (a)
$\{($match **and** $\mathrm{NMchU}(b-1)$ **and** $\mathrm{Mch}(b))$ **or**
(**not** match **and** $\mathrm{NMchU}(b))\}$ (b)
if not match **then** $b := b+1$
$\{($match **and** $\mathrm{NMchU}(b-1)$ **and** $\mathrm{Mch}(b))$ **or**
(**not** match **and** $\mathrm{NMchU}(b-1))\}$
until match **or** $b > n-m$
end $\{($match **and** $\mathrm{NMchU}(b-1)$ **and** $\mathrm{Mch}(b))$ **or**
(**not** match **and** $\mathrm{NMchU}(n-m))\}$

Observe that, in Algorithm 1, the assertion (a) implies the assertion (b) because from $\mathrm{NMchU}(b-1)$ **and** $\mathrm{Eq}(b, i)$ **and** $f_{b+i} \neq w_i$ we can deduce $\mathrm{NMchU}(b)$, but the converse does not hold. So, if we recall that, in Hoare axiomatic semantics, the proof rule of the null statement is

$$\frac{P \supset Q}{\{P\}\{Q\}},$$

we can say that between (a) and (b) there is a "true" null statement (where true means that $P \equiv Q$ does not hold). The net effect of a true null statement is some loss of information. So, in order to improve the poor efficiency of Algorithm 1 (worst case $m(n-m)$ character comparisons, see Fig. 1), it seems natural that we should try either to avoid the elaboration

```
step     f:  aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
1        w:  aaab
2             aaab
...            ......................................
n-m-1                                                  aaab
n-m                                                     aaab
```

FIG. 1. Example of the worst case for Algorithm 1. Character comparisons: $m(n-m)$.

of the unused information or to use it to avoid some subsequent elaboration.

Since it seems to be impacticable to avoid the elaboration of the unused information, we try to use it. To do so we observe that from (a) we know that $f_{b+i} \neq w_i$ and that $f_{b+k} \cdots f_{b+i-1} = w_k \cdots w_{i-1}$, for any $k, 0 \leqslant k \leqslant i$. Thus, in order for the pattern to match the text in position $b+k$, it is necessary that $w_k \cdots w_{i-1} = w_0 \cdots w_{i-k-1}$ and that $w_i \neq w_{i-k}$.

Perhaps, the best way to understand such a necessary condition is in terms of periodicities. We say that a string is k periodic when any two characters in the string that are k places away are equal. Observe that such a definition can apply also when k is 0 and when k is equal to the length of the string. Since two characters of a string that are 0 places away are obviously the same character, every string is 0 periodic and since in a string of length m there do not exist any two characters that are m places away, every string of length m is m periodic. Now, $w_k \cdots w_{i-1} = w_0 \cdots w_{i-k-1}$ means that the string $w_0 \cdots w_{i-1}$ is k periodic while $w_i \neq w_{i-k}$ means that $w_0 \cdots w_i$ does not. Thus, the condition can be expressed as "$w_0 \cdots w_{i-1}$ is the maximum k periodic prefix of the pattern" and, for all $0 \leqslant k \leqslant i < m$, we define MaxPP$(k, i)$ as an abbreviation for such an assertion. Formally,

$$\text{MaxPP}(k, i) \underset{\text{def}}{\equiv} w_k \cdots w_{i-1} = w_0 \cdots w_{i-k-1} \textbf{ and } w_i \neq w_{i-k}.$$

Then, the necessary condition takes the following form:

LEMMA 1. *Assume that* Eq(b, i) *and* $f_{b+i} \neq w_i$ *for some* i, $0 \leqslant i < m$. *Then, in order for the pattern to match the text in position* $b+k$ *for any* k, $0 \leqslant k \leqslant i$, *it is necessary that* MaxPP(k, i) *hold.* ∎

Such a lemma allows us to update more conveniently the position b of the pattern on the text when a mismatch occurs. Indeed, if MaxPP(k, i) does not hold for all k, $0 \leqslant k \leqslant i$, then we can move the pattern from position b to position $b+i+1$. Otherwise, let kmin(i) be the minimal k such that MaxPP(k, i) holds. Then we can move the pattern from position b to position $b+\text{kmin}(i)$ (observe that, since MaxPP$(0, i)$ is always false, kmin(i) is always greater than 0). Moreover, since $w_0 \cdots w_{i-1}$ matches the text in position b and $w_0 \cdots w_{i-1}$ is k periodic then $w_0 \cdots w_{i-k-1}$ matches the text in position $b+\text{kmin}(i)$ and so, in the new position, we can start with the comparison between the character w_{i-k} of the pattern and the character f_{b+i} of the text.

The new version of the algorithm is

ALGORITHM 2.

begin $\{f = f_0 f_1 \cdots f_{n-1}$ **and** $w = w_0 w_1 \cdots w_{m-1}$ **and** $n \geqslant m\}$
$b := 0$; $i := 0$;

repeat {NMchU($b-1$) **and** Eq(b, i)}
while $w_i = f_{b+i}$ **and** $i < m$ **do** $i := i+1$;
{(NMchU($b-1$) **and** $i = m$ **and** Eq(b, i)) **or**
(NMchU($b-1$) **and** $i < m$ **and** Eq(b, i) **and** $f_{b+i} \neq w_i$)}
match := $i = m$;
{(match **and** NMchU($b-1$) **and** Mch(b)) **or**
(**not** match **and** NMchU($b-1$) **and** Eq(b, i) **and** $f_{b+i} \neq w_i$)} (a)
if not match **then if** "kmin(i) is defined"
then begin
{**not** match **and** NMchU(b + kmin(i) − 1) **and**
Eq(b + kmin(i), i − kmin(i))} (b)
$b := b$ + kmin(i); $i := i$ − kmin(i)
end
else begin
{**not** match **and** NMchU($b+i$)} (c)
$b := b+i+1$; $i := 0$
end
{(match **and** NMchU($b-1$) **and** Mch(b)) **or**
(**not** match **and** NMchU($b-1$) **and** Eq(b, i))}
until match **or** $b > n-m$
end {(match **and** NMchU($b-1$) **and** Mch(b)) **or**
(**not** match **and** NMchU($n-m$))}

The essential difference between Algorithm 2 and the KMP algorithm is that in KMP, to restart the main loop when a failure has been detected, only the assertion Eq(b, i) is considered while the assertion $f_{b+i} \neq w_i$ is ignored. Taking account of the assertion $f_{b+i} \neq w_i$ allows us to avoid some subsequent useless comparisons that are done in KMP. However, in the worst case, both Algorithm 2 and the KMP algorithm require $2(n-m)$ character comparisons (see Fig. 2).

Such an improvement of the KMP algorithm is not new. Indeed, it was already known and suggested by Knuth, Morris, and Pratt in their original paper.

However, it is interesting to note the strategy we used to obtain such an

```
step     f:  aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
1        w:  abbb
2             abbb
...           ..........................................................
n-m-1                                                        abbb
n-m                                                           abbb
```

FIG. 2. Example of the worst case for Algorithm 2 and for the KMP Algorithm. Character comparisons: $2(n-m)$.

improved version of the algorithm since, in the next section, we will repeatedly use the same strategy to obtain a string matching algorithm that will be proved to behave better than KMP.

2. Further Improvements of the Pattern Matching Algorithm

This section contains two more refinement steps of the algorithm. In the first step we try to avoid the computation of the information that is lost between points (a) and (b) of Algorithm 2. Unfortunately, this cannot be done without the loss of some useful information too. The Algorithm 3 we obtain is faster than Algorithm 2 on the average (due to the saving in the computation of the useless information) but it shows a worst case bound of mn comparisons (due to the loss of useful information).

From the correctness proof of Algorithm 2 we can realize that we do not need all the information given by the assertion $\mathrm{Eq}(b, i)$ in (a) to assert that $\mathrm{NMchU}(b + \mathrm{kmin}(i) - 1)$ holds in (b) and that $\mathrm{NMchU}(b + i)$ holds in (c). Indeed, as a corollary of the following theorem, it suffices to know that $f_{b+h} = w_h$ holds for those positions h such that $\mathrm{kmin}(h)$ is defined.

THEOREM 1. *Let the pattern be aligned with the text starting at position b of the text. Assume that for a given i such that $0 \leqslant i < m$, we have that $f_{b+i} \neq w_i$ and that $f_{b+h} = w_h$ for all h between 0 and $i-1$ such that* $\mathrm{kmin}(h)$ *is defined; then* $\mathrm{Mch}(b+k)$ *does not hold for all k between 0 and i such that* $\mathrm{MaxPP}(k, i)$ *does not hold.*

Proof. Let k be any integer between 0 and i. Then one of the following three cases should occur:

(a) $\mathrm{MaxPP}(k, i)$ holds.

(b) There exists $h < i$ such that $\mathrm{MaxPP}(k, h)$ holds. In such a case $\mathrm{kmin}(h)$ is defined and so $f_{b+h} = w_h$ by the hypothesis. Since $\mathrm{MaxPP}(k, h)$ holds we have $w_h \neq w_{h-k}$. Then $p = h - k$ is an integer less than m such that $f_{b+k+p} = f_{b+h} = w_h$ is different from $w_p = w_{h-k}$ and so $\mathrm{Mch}(b+k)$ does not hold.

(c) $\mathrm{MaxPP}(k, h)$ does not hold for all h, $0 \leqslant h \leqslant i$. In such a case the string $w_0 \cdots w_i$ is k periodic and, because of by hypothesis $f_{b+i} \neq w_i$, $p = i - k$ is an integer less than m such that $f_{b+k+p} = f_{b+i}$ is different from $w_p = w_{i-k} = w_i$. Thus $\mathrm{Mch}(b+k)$ does not hold. ∎

COROLLARY. *Under the hypothesis of Theorem* 1, *if* $\mathrm{kmin}(i)$ *is defined then* $\mathrm{Mch}(b+k)$ *does not hold for all k between* 0 *and* $\mathrm{kmin}(i) - 1$, *while if* $\mathrm{kmin}(i)$ *is undefined then* $\mathrm{Mch}(b+k)$ *does not hold for all k between* 0 *and i.*

Proof. If kmin(i) is defined then it is the first k such that MaxPP(k, i) holds, while if kmin(i) is undefined then MaxPP(k, i) does not hold for all k, $0 \leqslant k \leqslant i$. ∎

Let $h_1, ..., h_{nd}$ be all the integers between 0 and $m-1$ such that kmin(h_i) is defined and let $h_{nd+1}, ..., h_m$ be the others. Suppose that $h_1, ..., h_m$ are ordered in such a way that $h_1 < \cdots < h_{nd}$ and $h_{nd+1} > \cdots > h_m$.

Moreover let Eqh(b, i) be the assertion "the equality $w_h = f_{b+h}$ holds for the first i elements in the sequence $h_1, ..., h_{nd}$," formally

$$\text{Eqh}(b, i) \underset{\text{def}}{\equiv} \forall p((1 \leqslant p \leqslant i) \supset (w_{h_p} = f_{b+h_p})).$$

Then the following algorithm takes advantage of the corollary of Theorem 1 by testing for the equality $f_{b+h_i} = w_{h_i}$ in the order $h_1, ..., h_m$.

ALGORITHM 3.

begin $\{f = f_0 f_1 \cdots f_{n-1}$ **and** $w = w_0 w_1 \cdots w_{m-1}$ **and** $n \geqslant m\}$
$b := 0$; $i := 1$;
repeat $\{$NMchU($b-1$) **and** Eqh(b, $i-1$)$\}$
while $w_{h_i} = f_{b+h_i}$ **and** $i \leqslant m$ **do** $i := i+1$;
$\{$(NMchU($b-1$) **and** $i = m+1$ **and** Eqh(b, $i-1$)) **or**
(NMchU($b-1$) **and** $i \leqslant m$ **and** Eqh(b, $i-1$) **and** $f_{b+h_i} \neq w_{h_i}$)$\}$
match := $i = m+1$;
$\{$(match **and** NMchU($b-1$) **and** Mch(b)) **or**
(**not** match **and** NMchU($b-1$) **and** Eqh(b, $i-1$) **and** $f_{b+h_i} \neq w_{h_i}$)$\}$
if not match **then** if $i \leqslant nd$
then begin $\{$kmin(h_i) is defined$\}$
$\{$**not** match **and** NMchU(b + kmin(h_i) -1)$\}$
$b := b +$ kmin(h_i); $i := 1$
end
else begin (kmin(h_i) is undefined$\}$
$\{$**not** match **and** NMchU($b + h_i$)$\}$
$b := b + h_i + 1$; $i := 1$
end
$\{$(match **and** NMchU($b-1$) **and** Mch(b)) **or**
(**not** match **and** NMchU($b-1$) **and** Eqh(b, $i-1$))$\}$
until match **or** $b > n - m$
end $\{$(match **and** NMchU($b-1$) **and** Mch(b)) **or**
(**not** match **and** NMchU($n-m$))$\}$

As we can see in Table 1, where the results of some comparative tests are reported, Algorithm 3 is more effective than Algorithm 2 on the average. In the worst case, instead, Algorithm 3 requires $m(n-m)$ comparisons (e.g.,

```
step     f:   aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
1        w:   baaa
2              baaa
...             ...........................................
n-m-1                                                  baaa
n-m                                                     baaa
```

FIG. 3. Example of the worst case for Algorithm 3. Character comparisons: $m(n-m)$.

$w = ba^{m-1}$ and $f = a^n$, see Fig. 3) while Algorithm 2 requires only $2(n-m)$ comparisons (e.g. $w = ab^{m-1}$ and $f = a^n$, see Fig. 2).

To attain a better performance in the worst case too, we observe, once more, that in Algorithm 3 some information given by the assertion Eqh$(b, i-1)$ is lost (as in Algorithm 1 with the assertion Eq(b, i)). So we use that information to avoid some computation in the next iteration. This is done in two ways. First, since in Algorithm 3 we first test all the h_i such that kmin(h_i) is defined, we are able to prove (Theorem 2) that we can update b more conveniently in case of a mismatch on an h_i such that kmin(h_i) is undefined. Second, we prove that, both in the case kmin(h_i) undefined (Lemma 2) and in the case kmin(h_i) defined (Lemma 3), the index i can be updated more conveniently.

To state Theorem 2 we need a function rmin(i) that, like the function kmin(i), is defined in terms of the periodicities of the pattern. The function rmin(i) is defined as the first positive integer r greater than i such that the pattern is r periodic. Observe that, since the pattern $w_0 \cdots w_{m-1}$ is m periodic, rmin(i) is always defined and $i < \text{rmin}(i) \leqslant m$.

THEOREM 2. *Let the pattern be aligned with the text starting at position b of the text. Assume that $f_{b+h} = w_h$ for all h, $0 \leqslant h < m$, such that* kmin(h) *is defined and that for a given i, $0 \leqslant i < m$, such that* kmin(i) *is undefined we have that $f_{b+i} \neq w_i$, then* Mch$(b+k)$ *does not hold for all k between* 0 *and* rmin$(i)-1$.

Proof. Let k be any integer between 0 and rmin$(i)-1$. Then one of the following two cases should occur:

(a) there exists $h < m$, such that MaxPP(k, h) holds. In such a case kmin(h) is defined and so $f_{b+h} = w_h$. Then $p = h-k$ is a non-negative integer less than m such that $f_{b+k+p} = f_{b+h} = w_h$ is different from $w_p = w_{h-k}$ and so Mch$(b+k)$ does not hold;

(b) MaxPP(k, h) does not hold for all h, $0 \leqslant h < m$. In such a case the pattern is k periodic and the minimality of rmin(i) implies that $k \leqslant i$. Then $p = i-k$ is a non-negative integer less than m such that $f_{b+k+p} = f_{b+i}$ is different from $w_p = w_{i-k} = w_i$ and so Mch$(b+k)$ does not hold. ∎

In order to update more conveniently the index i in the case $\mathrm{kmin}(h_i)$ undefined, we note that, since $h_1 < \cdots < h_{nd}$, $h_{nd+1} > \cdots > h_i > \cdots > h_m$, and $\mathrm{rmin}(h_i) > h_i$, the assertion $\mathrm{Eqh}(b, i-1)$ implies that $f_{b+\mathrm{rmin}(h_i)} \cdots f_{b+m-1}$ is equal to $w_{\mathrm{rmin}(h_i)} \cdots w_{m-1}$ and therefore it is equal to $w_0 \cdots w_{m-\mathrm{rmin}(h_i)-1}$ because of the periodicity of the pattern. Let $\mathrm{nhd}(i, j)$, for $0 \leqslant i \leqslant j \leqslant m$, be defined as the number of h such that $i \leqslant h < j$ and $\mathrm{kmin}(h)$ is defined. Then, for all indices such that $1 \leqslant p \leqslant \mathrm{nhd}(0, m-\mathrm{rmin}(h_i))$, we have that $w_{h_p} = f_{b+\mathrm{rmin}(h_i)+h_p}$. This proves the following:

LEMMA 2. *If the assertion* $\mathrm{Eqh}(b, i-1)$ *holds for any* $i > nd$ *when the pattern is aligned with the text in position* b *then the assertion* $\mathrm{Eqh}(b+\mathrm{rmin}(h_i), \mathrm{ndh}(0, m-\mathrm{rmin}(h_i)))$ *holds when the pattern is shifted in position* $b+\mathrm{rmin}(h_i)$. ∎

To take advantage of the assertion $\mathrm{Eqh}(b, i-1)$ when $i \leqslant nd$ we observe that, if we shift the pattern $\mathrm{kmin}(h_i)$ places over itself, then any character w_p such that $\mathrm{kmin}(p)$ is defined and $0 \leqslant p < h_i - \mathrm{kmin}(h_i)$ will be moved over a character $w_{p+\mathrm{kmin}(h_i)}$ equal to w_p and such that $\mathrm{kmin}(p+\mathrm{kmin}(h_i))$ is defined too.

Perhaps the best way to prove such a property is to prove it as a Corollary of the following Lemma that shows how a periodicity in the pattern reflects into a periodicity of the relation $\mathrm{MaxPP}(k, i)$.

LEMMA 3. *Let* $w_0 \cdots w_q$ *be* r *periodic and let* k *and* h *be such that* $1 \leqslant k \leqslant h \leqslant q-r$. *Then* $\mathrm{MaxPP}(k, h)$ *holds if and only if* $\mathrm{MaxPP}(k+r, h+r)$ *holds.*

Proof. Let j be any integer such that $k \leqslant j \leqslant h$. Then $w_j = w_{j+r}$ and $w_{j+r-(k+r)} = w_{j-k}$ and so $w_j = w_{j-k}$ if and only if $w_{j+r} = w_{j+r-(k+r)}$. So $w_k \cdots w_{h-1} = w_0 \cdots w_{h-1-k}$ if and only if $w_{k+r} \cdots w_{h+r-1} = w_0 \cdots w_{h+r-1-(k+r)}$ and $w_h \neq w_{h-k}$ if and only if $w_{h+r} \neq w_{h+r-(k+r)}$. ∎

COROLLARY. *Let* i *be such that* $i \leqslant nd$ *and* $\mathrm{Eqh}(b, i-1)$ *holds when the pattern is aligned to the text in position* b *of the text. Then* $\mathrm{Eqh}(b+\mathrm{kmin}(h_i), \mathrm{nhd}(0, h_i-\mathrm{kmin}(h_i)))$ *holds when the pattern is shifted in position* $b+\mathrm{kmin}(h_i)$.

Proof. Since $i \leqslant nd$, $\mathrm{kmin}(h_i)$ is defined and the string $w_0 \cdots w_{h_i-1}$ is $\mathrm{kmin}(h_i)$ periodic. Let h_j be such that $j \leqslant \mathrm{nhd}(0, h_i-\mathrm{kmin}(h_i))$. Since $\mathrm{MaxPP}(\mathrm{kmin}(h_j), h_j)$ holds and $1 \leqslant \mathrm{kmin}(h_j) \leqslant h_j < h_i - \mathrm{kmin}(h_i)$ then $\mathrm{MaxPP}(\mathrm{kmin}(h_j)+\mathrm{kmin}(h_i), h_j+\mathrm{kmin}(h_i))$ holds by Lemma 3, and so $\mathrm{kmin}(h_j+\mathrm{kmin}(h_i))$ is defined. Since $h_j+\mathrm{kmin}(h_i)$ is less than h_i and

Eqh$(b, i-1)$ holds, the character $f_{b+\text{kmin}(h_i)+h_j}$ of the text is equal to the character $w_{\text{kmin}(h_i)+h_j}$ of the pattern and, since $w_0 \cdots w_{h_i-1}$ is kmin(h_i) periodic, $w_{\text{kmin}(h_i)+h_j}$ is equal to w_{h_j}. Then $f_{b+\text{kmin}(h_i)+h_j} = w_{h_j}$ for all h_j such that $j \leqslant \text{nhd}(0, h_i - \text{kmin}(h_i))$ and so Eqh$(b+\text{kmin}(h_i), \text{nhd}(0, h_i - \text{kmin}(h_i)))$ holds. ▮

Let us define the two functions shift(i) and next(i) as follows:

(i) shift$(i) = \text{kmin}(h_i)$ and next$(i) = \text{nhd}(0, h_i - \text{kmin}(h_i)) + 1$ for $i \leqslant nd$,

(ii) shift$(i) = \text{rmin}(h_i)$ and next$(i) = \text{nhd}(0, m - \text{rmin}(h_i)) + 1$ for $nd < i \leqslant m$, and

(iii) shift$(m+1) = 0$ and next$(m+1) = 0$.

Then the following algorithm correctly solves the pattern matching problem.

ALGORITHM 4.

begin $\{f = f_0 f_1 \cdots f_{n-1}$ **and** $w = w_0 w_1 \cdots w_{m-1}$ **and** $n \geqslant m\}$
$b := 0$; $i := 1$;
repeat $\{$NMchU$(b-1)$ **and** Eqh$(b, i-1)\}$
while $w_{h_i} = f_{b+h_i}$ **and** $i \leqslant m$ **do** $i := i+1$;
$\{$(NMchU$(b-1)$ **and** $i = m+1$ **and** Eqh$(b, i-1)$) **or**
(NMchU$(b-1)$ **and** $i \leqslant m$ **and** Eqh$(b, i-1)$ **and** $f_{b+h_i} \neq w_{h_i}$)$\}$
match $:= i = m+1$;
$\{$(match **and** NMchU$(b+\text{shift}(i)-1)$ **and** Mch$(b+\text{shift}(i))$) **or**
(**not** match **and** NMchU$(b+\text{shift}(i)-1)$ **and**
Eqh$(b+\text{shift}(i), \text{next}(i)-1)$)$\}$
$b := b+\text{shift}(i)$; $i := \text{next}(i)$;
$\{$(match **and** NMchU$(b-1)$ **and** Mch(b)) **or**
(**not** match **and** NMchU$(b-1)$ **and** Eqh$(b, i-1)$)$\}$
until match **or** $b > n-m$
end$\{$(match **and** NMchU$(b-1)$ **and** Mch(b)) **or**
(**not** match **and** NMchU$(n-m)$)$\}$.

Observe that Algorithm 4 can be easily modified to find all the occurrences of the pattern in the text. To do so we replace the statement "match $:= i = m+1$" by the output statement "if $i = m+1$ then write(b)". We take out the condition "match" from the exit condition of the repeat statement and we define shift$(m+1) = \text{shift}(m)$ and next$(m+1) = \text{next}(m)$ instead of shift$(m+1) = 0$ and next$(m+1) = 0$. The modified algorithm is

ALGORITHM 5.

begin
$b := 0$; $i := 1$;
repeat
while $w_{h_i} = f_{b+h_i}$ **and** $i < m$ **do** $i := i + 1$;
if $i = m + 1$ **then** write(b);
$b := b + \text{shift}(i)$; $i := \text{next}(i)$
until $b > n - m$
end.

3. THE RESULT OF SOME TESTS

Algorithms 2, 3, and 4, the basic algorithm KMP, and the algorithm BM have been run with some test cases and the ratios c/n between the number c of character comparisons performed and the length n of the text have been reported in Table 1. In the first 18 test cases, 100 random generated patterns of length m have been tested for a match into a random sequence f of 10,000 characters on a given alphabet of T symbols using all the five algorithms. In the following 12 test cases, the first 10,000 characters of two text files ($tx1$, which is the text of this paper, and $tx2$, which is the text of a large Pascal program) have been used as f and the 100 patterns of length m have been extracted from the same text file, starting from the 10,001th character. In the last 2 test cases, the first 10,000 characters of $tx1$ and $tx2$ have been used as f and the first 100 whole words (including an initial and a final character) that follow the 10,000th character have been used as patterns.

To make comparable the results relative to the cases with a low value of m (where the probability of a successful termination is very high) and to the cases with an high value of m (where successful termination is very rare) we chose the sequence f at random between the sequences that do not contain a match of the pattern in the first 18 test cases, and, in the other test cases we forced a mismatch with the last element of the pattern anywhere a match was found. So the results reported in Table 1 are all relative to unsuccessful executions of the algorithms.

In all the cases the average and the maximum value of the ratio c/n relative to the 100 patterns had been reported in Table 1.

About the results reported in Table 1, it should be noted that:

(a) In all the test cases Algorithm 2 works better than KMP and Algorithm 4 works better than KMP, Algorithm 2, and Algorithm 3.

(b) In general Algorithm 3 works better than both KMP and Algorithm 2 with a performance that is very close to that of Algorithm 4. There

TABLE 1

The Ratio c/n for Some Test Cases

		Alg. KMP		Alg. 2		Alg. 3		Alg. BM		Alg. 4	
T	m	ave.	max.	ave.	max.	ave.	max.	ave.	max.	ave.	max
2	2	1.66	2.00	1.49	2.00	0.83	1.00	0.83	1.00	0.83	1.00
2	3	1.69	2.00	1.59	2.00	0.89	1.00	0.86	1.20	0.89	1.00
2	5	1.48	2.00	1.31	2.00	0.87	1.09	0.80	1.07	0.81	1.00
2	10	1.42	1.50	1.25	1.49	0.82	1.01	0.61	0.92	0.78	1.00
2	20	1.40	1.50	1.19	1.50	0.72	1.00	0.45	0.66	0.70	1.00
2	30	1.42	1.50	1.24	1.50	0.78	1.00	0.38	0.66	0.76	1.00
5	2	1.21	1.21	1.19	1.21	1.10	1.16	0.96	1.00	0.96	1.00
5	3	1.20	1.21	1.15	1.21	1.01	1.07	0.88	1.03	0.94	1.00
5	5	1.20	1.20	1.15	1.20	0.96	1.01	0.74	0.97	0.95	1.00
5	10	1.20	1.20	1.16	1.20	0.96	1.00	0.59	0.94	0.96	1.00
5	20	1.20	1.20	1.16	1.20	0.96	1.00	0.51	0.89	0.96	1.00
5	30	1.20	1.20	1.16	1.20	0.96	1.00	0.47	0.92	0.96	1.00
20	2	1.05	1.05	1.05	1.05	1.02	1.05	0.98	1.00	0.98	1.00
20	3	1.05	1.05	1.05	1.05	1.00	1.01	0.95	1.00	1.00	1.00
20	5	1.05	1.05	1.05	1.05	1.00	1.00	0.86	1.00	1.00	1.00
20	10	1.05	1.05	1.05	1.05	1.00	1.00	0.76	0.99	1.00	1.00
20	20	1.05	1.05	1.05	1.05	1.00	1.00	0.68	0.99	1.00	1.00
20	30	1.05	1.05	1.04	1.05	0.99	1.00	0.62	0.95	0.99	1.00
*tx*1	2	1.05	1.17	1.04	1.17	1.04	1.17	0.98	1.00	0.98	1.00
*tx*1	3	1.06	1.17	1.05	1.17	1.01	1.05	0.96	1.00	1.00	1.00
*tx*1	5	1.06	1.17	1.06	1.17	1.00	1.04	0.91	1.00	1.00	1.00
*tx*1	10	1.06	1.17	1.05	1.17	1.00	1.01	0.81	1.00	1.00	1.00
*tx*1	20	1.06	1.17	1.06	1.17	0.99	1.00	0.70	0.99	0.99	1.00
*tx*1	30	1.07	1.17	1.07	1.17	1.00	1.00	0.69	1.00	1.00	1.00
*tx*2	2	1.07	1.24	1.06	1.24	1.05	1.24	0.98	1.00	0.98	1.00
*tx*2	3	1.08	1.24	1.07	1.24	1.00	1.22	0.92	1.01	0.98	1.00
*tx*2	5	1.10	1.24	1.07	1.24	0.95	1.17	0.82	1.00	0.94	1.00
*tx*2	10	1.08	1.24	1.05	1.24	0.98	1.00	0.78	0.99	0.98	1.00
*tx*2	20	1.09	1.23	1.05	1.23	0.98	1.00	0.68	0.99	0.98	1.00
*tx*2	30	1.11	1.23	1.06	1.23	0.98	1.00	0.59	0.99	0.98	1.00
*tx*1	*wrd*	1.17	1.17	1.17	1.17	1.00	1.00	0.66	1.00	1.00	1.00
*tx*2	*wrd*	1.23	1.24	1.23	1.24	1.00	1.01	0.69	1.00	1.00	1.00

are, however, some cases where the very poor performance of Algorithm 3 in the worst case ($c/n = m$) makes it work very badly, even worse than KMP.

(c) The maximum ratio c/n for Algorithm 4 is always less than (and very close to) 1. Observe that in the first 18 test cases, since the text is random and very large, the ratio c/n relative to a single pattern will be a good

estimate of the average value of c/n relative to the given pattern. So a maximum ratio less than 1 can suggest that the average value of c/n is less than 1 for all patterns. We conjecture that 1 is a worst case bound of c/n on a random text for all patterns and the fact that the ratio c/n happens to be lower than 1 in all the test cases we have reported in Table 1 (and in all the other cases we have tested) supports this conjecture to some extent.

(d) The ratios c/n for Algorithm 3 and for BM are not always less than 1. So there is some pattern such that the number of comparisons is greater than n on a random text of length n.

(e) The comparison between BM ad Algorithm 4 is very involved. On one side BM works better than Algorithm 4 in the average in all the test cases with a performance which is close to that of Algorithm 4 for short patterns and much better for large patterns. On the other side, the theoretical bound we will prove in Section 5 says (and the experimental data confirm) that the performance of Algorithm 4 is very stable while the lack of goods upper bounds makes BM very unsafe (more unsafe than KMP and Algorithm 2). So Algorithm 4 is superior when the worst case behaviour is of primary importance (real time applications) while BM should be preferred when the stress is on the average behaviour (batch applications).

(f) With respect to the algorithm KMP, Algorithm 4 saves about 45% of the computer time when $T=2$, about 20% when $T=5$, and about 5% when $T=20$. Moreover, the saving is about 6% on a normal text, which grows to 15% when we are looking for a whole word and the saving is about 10% on a program text which grows to 20% when we are looking for a whole word.

4. Computation of kmin(h), rmin(h), $h(i)$, shift(i), and next(i).

The functions kmin(h), rmin(h), $h(i)$, shift(i), and next(i) do not depend on f and so we can compute and store them before the start of the main loop.

The function kmin(h) is defined in terms of the relation MaxPP(k, h). So we compute such a relation first. To do so we observe that MaxPP(0, h) is always false while, for all k such that $1 \leqslant k \leqslant m$, either there exists one and only one value hmax(k) $< m$ such that MaxPP(k, hmax(k)) holds or, otherwise, the pattern is k periodic.

To compute hmax(k) for all $k = 1, ..., m$, we start with the following very simple Algorithm 6, where the only trick is the use of a sentinel to avoid the test for the end of the pattern. Observe that in such a way what we really compute is the function hmax(k) for the extended pattern. However, the extension of the pattern does not alter the value that the function

hmax(k) takes where it is defined. The extension only makes the function defined with a value of m whenever it was undefined. So hmax(k) = m in the computed function really means that hmax(k) is undefined.

ALGORITH 6.

begin
$w_m := \#$; {set the sentinel}
for $k := 1$ **to** m **do**
  begin $\{\forall p((1 \leqslant p < k) \supset \text{MaxPP}(p, \text{hmax}[p]))\}$
  $i := k$;
  while $w_i = w_{i-k}$ **do** $i := i + 1$;
  $\{\forall p((1 \leqslant p < k) \supset \text{MaxPP}(p, \text{hmax}[p]))$ **and**
   $w_1 \cdots w_{i-1}$ is k periodic **and** $w_i \neq w_{i-k}\}$ (a)
  $\{\forall p((1 \leqslant p < k) \supset \text{MaxPP}(p, \text{hmax}[p]))$ **and** $\text{MaxPP}(k, i)\}$ (b)
  $\text{hmax}[k] := i$
  end
end $\{\forall p((1 \leqslant p \leqslant m) \supset \text{MaxPP}(p, \text{hmax}[p]))\}$

Algorithm 6 is not very efficient since it shows a time complexity of $O(m^2)$. To improve it we observe, once again, that some information is lost between (a) and (b) when we deduce that MaxPP(k, i) holds. Indeed, since $w_0 \cdots w_{i-1}$ is k periodic, we can exploit the induced periodicity of the relation MaxPP to deduce, for all q such that $k + 1 \leqslant q \leqslant i$, that if $\text{hmax}(q-k) + k < i$ then $\text{MaxPP}(q, \text{hmax}(q-k)+k)$ holds, otherwise hmax(q) should be greater or equal to i and so the string $w_0 \cdots w_{i-1}$ is q periodic.

So, in addition to setting $\text{hmax}[k] := i$, we can also set $\text{hmax}[q] := \text{hmax}[q-k] + k$ for all q greater than k until we reach the first q such that $\text{hmax}[q-k] + k \geqslant i$. Observe that, since $\text{hmax}(i+1-k) + k \geqslant i + 1 - k + k = i + 1$, there should be a first $q \leqslant i + 1$ such that $\text{hmax}(q-k) + k \geqslant i$. Then, to continue the main loop, we reset k to such a q and, since for $q \leqslant i$ we already know that $w_0 \cdots w_{i-1}$ is q periodic, we need to reset i to the new value of k only if $k = i + 1$.

This leads to the following:

ALGORITH 7.

begin
$w_m := \#$, {set the sentinel}
$k := 1$; $i := k$;
repeat
$\{\forall p((1 \leqslant p < k) \supset \text{MaxPP}(p, \text{hmax}[p]))$ **and** $w_0 \cdots w_{i-1}$ is k periodic$\}$
while $w_i = w_{i-k}$ **do** $i := i + 1$;
$\{\forall p((1 \leqslant p < k) \supset \text{MaxPP}(p, \text{hmax}[p]))$ **and** $w_0 \cdots w_{i-1}$ is k periodic **and**
 $w_i \neq w_{i-k}\}$

```
hmax[k] := i;
{∀p((1 ⩽ p ⩽ k) ⊃ MaxPP(p, hmax[p])) and w_0 ··· w_{i-1} is k periodic}
q := k + 1;
while hmax[q − k] + k < i do
begin hmax[q] := hmax[q − k] + k; q := q + 1 end;
{∀p((1 ⩽ p < q) ⊃ MaxPP(p, hmax[p])) and w_0 ··· w_{i-1} is q periodic}
k := q;
if k = i + 1 then i := k
until k > m
end {∀p((1 ⩽ p ⩽ m) ⊃ MaxPP(p, hmax[p]))}
```

Every iteration of all the loops of this algorithm increases either i or k (or the auxiliary variable q which is eventually used to update k). Since $i = 1$ and $k = 1$ at the start and $i \leqslant k = m + 1$ at the end of the algorithm, a statement of the algorithm cannot be executed more than $2m$ times. So the time complexity of this algorithm is $O(m)$.

The computation of kmin(h), rmin(h), $h(i)$, shift(i), and next(i) on the basis of hmax(k) is simple and can be done in time $O(m)$ as follows:

```
for i := 0 to m − 1 do kmin[i] := 0;
for k := m downto 1 do if hmax[k] < m then kmin[hmax[k]] := k;

for i := m − 1 downto 0 do
begin
  if hmax [i + 1] = m then r := i + 1;
  {r is the first integer greater than i such that the pattern is r periodic}
  if kmin[i] = 0 then rmin[i] := r else rmin[i] := 0
end;

s := 0; t := m + 1;
for i := 0 to m − 1 do if kmin[i] = 0
then begin t := t − 1, h[t] := i end
else begin s := s + 1; h[s] := i end;

for i := 1 to m do shift[i] := kmin[h[i]] + rmin[h[i]];
shift[m + 1] := 0;

s := 0;
for i := 0 to m − 1 do begin nhd0[i] := s; if kmin[i] > 0 then s := s + 1 end;
{∀i((0 ⩽ i < m) ⊃ (nhd0[i] = nhd(0, i)))}

for i := 1 to m do if kmin[h[i]] = 0
  then next[i] := nhd0[m − rmin[h[i]]] + 1
  else next[i] := nhd0[h[i] − kmin[h[i]]] + 1.
```

5. The Worst Case Bound of Algorithm 4

In this section we show that the number of character comparisons Algorithm 4 requires is bounded by $1.5n + 0.5(m-1)$ in the worst case. To do so we need a better insight into the structure of the relation MaxPP(k, i) and of the related function kmin(i).

We already know (Lemma 3) that if the pattern is p periodic for some p then for all k and h such that $1 \leqslant k \leqslant h < m-p$ the relation MaxPP(k, h) holds if and only if MaxPP$(k+p, h+p)$ holds. So in Fig. 4, where the structure of MaxPP(k, h) is sketched, the two triangles indicated by A look the same.

We will prove that MaxPP(k, h) cannot hold if $k < h-p+2$ and so all the graph of MaxPP(k, h) lies in the strip bounded by the straight lines $k = h-p+2$ and $k = h$. To do so we need a result about the periodicities of a string that is due to Knuth, Morris, and Pratt and reported as Lemma 1 in Knuth, Morris, and Pratt (1977). We report such a result in Lemma 4 and, since our definition of periodicity of a string is slightly more general than that of Knuth, Morris, and Pratt (and since the proof in Knuth, Morris, and Pratt (1977) seems to contain a bug), we also give the proof.

LEMMA 4. *Let the string* $w_0 \cdots w_{m-1}$ *be both* p *periodic and* q *periodic with* $0 \leqslant p \leqslant q \leqslant m$. *If* $p+q \leqslant m + \gcd(p, q)$ *then the string is* $\gcd(p, q)$ *periodic* (*where we assume* $\gcd(0, 0) = 0$).

Proof. Note that in our definition the string $w_0 \cdots w_{m-1}$ is said to be p periodic when $w_0 \cdots w_{m-p-1} = w_p \cdots w_{m-1}$ and this makes sense for $p = 0$

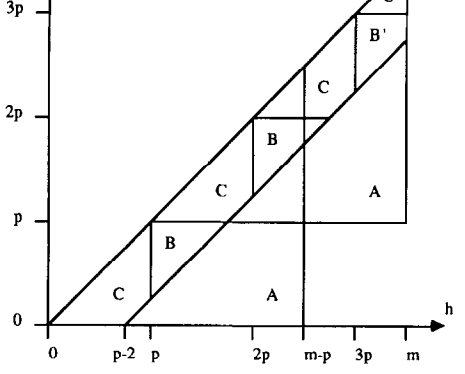

FIG. 4. The structure of the graph of the relation MaxPP(k, h) relative to a p periodic pattern.

and for $p=m$ too. Moreover, since under the hypotheses of the lemma both the equality $w_0 \cdots w_{m-q-1} = w_q \cdots w_{m-1}$ and $w_0 \cdots w_{m-p-1} = w_p \cdots w_{m-1}$ hold then $w_0 \cdots w_{m-q-1} = w_{q-p} \cdots w_{m-p-1}$ and so the string $w_0 \cdots w_{m-p-1}$ is $q-p$ periodic.

We prove the Lemma by induction on $p+q$. Indeed, if $p+q=0$ then $p=0$, $q=0$ and $\gcd(p, q)=0$ and so $w_0 \cdots w_{m-1}$ is $\gcd(p, q)$ periodic.

Let $p+q>0$. If $p=0$ or $p=q$ then $\gcd(p, q)=q$ and so $w_0 \cdots w_{m-1}$ is $\gcd(p, q)$ periodic. So let us suppose $1 \leqslant p < q$. In such a case $p+(q-p)=q$ is strictly less than $p+q$ and so we can apply the induction hypothesis to the string $w_0 \cdots w_{m-p-1} = w_p \cdots w_{m-1}$ to show that, since $\gcd(p, q-p) = \gcd(p, q)$ and $p+(q-p) \leqslant m-p+\gcd(p, q)$, it is $\gcd(p, q)$ periodic. Moreover, since $\gcd(p, q) = \gcd(p, q-p) \leqslant q-p$ and $p+q \leqslant m+\gcd(p, q)$ by hypothesis, then $p \leqslant m-p$ and the two strings $w_p \cdots w_{m-1}$ and $w_0 \cdots w_{m-p-1}$ cover the whole string $w_0 \cdots w_{m-1}$.

Let now i and j be two indexes such that $j=i+\gcd(p, q)$ and $0 \leqslant i \leqslant j < m$. If both w_i and w_j belong to the same string $w_p \cdots w_{m-1}$ or $w_0 \cdots w_{m-p-1}$ then $w_i = w_j$. Otherwise w_i belongs to $w_0 \cdots w_{m-p-1}$ and w_j belongs to $w_p \cdots w_{m-1}$. Then $i \leqslant m-p-1$ and $j \geqslant p$. In such a case $w_j = w_{j-p}$ because of the p periodicity of $w_0 \cdots w_{m-1}$ and $w_{j-p} = w_{i+\gcd(p,q)-p} = w_i$ because of the $\gcd(p, q)$ periodicity of $w_0 \cdots w_{m-p-1}$. So the whole string $w_0 \cdots w_{m-1}$ is $\gcd(p, q)$ periodic. ∎

LEMMA 5. *Let the pattern $w_0 \cdots w_{m-1}$ be p periodic and let k and h be such that $h<m$ and* MaxPP(k, h) *holds. Then $k \geqslant h-p+2$.*

Proof. Since MaxPP(k, h) holds then $q=\text{kmin}(h)$ is defined and $k \geqslant q$. We prove that $q \geqslant h-p+2$ by contradiction. Indeed, let us suppose that $q < h-p+2$. Then $p+q \leqslant h+1 \leqslant h+\gcd(p, q)$. Since $h<m$ and $q=\text{kmin}(h)$, the string $w_0 \cdots w_{h-1}$ is both p and q periodic and so it is also $\gcd(p, q)$ periodic. Moreover, $w_h \neq w_{h-q}$ and $\gcd(p, q)$ divides q and so the string $w_0 \cdots w_h$ is not $\gcd(p, q)$ periodic and then MaxPP$(\gcd(p, q), h)$ holds. Since neither p divides q nor q divides p (otherwise $w_0 \cdots w_h$ would be q periodic) $\gcd(p, q)$ is strictly less than q and this contradict with the minimality of q. ∎

So, in Fig. 4, all the graph of MaxPP(k, h) lies in the strip bounded by the straight lines $k=h-p+2$ and $k=h$. Moreover, by Lemma 3, all the small triangles indicated by B and all the regions indicated by C are equal and the partial regions B′ and C′ are equal to the corresponding parts of the regions B and C.

The structure of the function kmin(h) is similar to the structure of the relation MaxPP(k, h). Indeed, let i and j be such that $j=i+p$ and $0 \leqslant i \leqslant j < m$. If $i \geqslant p-2$ then kmin(i) is defined if and only if kmin(j) is, and in such a case $\text{kmin}(j) = \text{kmin}(i)+p$. Otherwise, due to the lack of the

region B in the graph of the relation MaxPP(k, h) for $0 \leqslant h < p-2$, it can happen that kmin(j) is defined but kmin(i) is not and, even if kmin(i) defined still implies kmin(j) defined, the value of kmin(j) can be strictly less than kmin(i) + p. But how many kmin(i) are undefined? The question is not academic since such a difference represents a loss of information when, in Algorithm 4, we deduce Eqh(b + shift(i), next(i) − 1)) from Eqh(b, $i-1$) and $f_{b+h_i} \neq w_{h_i}$. The answer is in the following lemma.

LEMMA 6. *Let the string $w_0 \cdots w_{m-1}$ be p periodic and let i and j be such that $p \leqslant i \leqslant j \leqslant m$ and let* nhd(i, j), *for $0 \leqslant i \leqslant j \leqslant m$, be defined as the number of h such that $i \leqslant h < j$ and* kmin(h) *is defined. Then the following inequalities hold*:

(a) nhd($i-p$, i) $\leqslant$ nhd($j+p$, j), *where the equality holds whenever $i \geqslant 2p-2$*;

(b) nhd($j-p$, j) $\leqslant$ nhd(0, p) + $(p-1)/2$.

Proof. Without loss of generality we can suppose $m \geqslant 2p$. Indeed, the value of the relation MaxPP(k, h) depends only on the prefix $w_0 \cdots w_h$ of the string and so if $m < 2p$ then the value of nhd($j-p$, j) does not change if we compute it on the string $(w_0 \cdots w_{p-1})^2$ instead of on the original string $w_0 \cdots w_{m-1}$.

(a) Since nhd($i-p$, i) = nhd($i-p$, j) − nhd(i, j) and nhd($j-p$, j) = nhd($i-p$, j) − nhd($i-p$, $j-p$) the inequality nhd($i-p$, i) $\leqslant$ nhd($j-p$, j) is equivalent to nhd($i-p$, $j-p$) $\leqslant$ nhd(i, j). Let now h be such that $i \leqslant h < j$. Then, by Lemma 3, if kmin($h-p$) is defined kmin(h) is also defined, and this proves the inequality nhd($i-p$, $j-p$) $\leqslant$ nhd(i, j). Moreover, if kmin(h) is defined and $i \geqslant 2p-2$ then kmin(h) $\geqslant p$ by Lemma 5 and so kmin($h-p$) is defined by Lemma 3 and so nhd($i-p$, $j-p$) = nhd(i, j);

(b) by point a) it is enough to prove that nhd(p, $2p$) $\leqslant$ nhd(0, p) + $(p-1)/2$ and, since kmin($h-p$) is defined whenever kmin(h) is defined and kmin(h) $> p$, then it suffices to prove that there are at most $(p-1)/2$ values of h such that $p \leqslant h < 2p$, kmin(h) is defined and kmin(h) $< p$. Indeed, let k_i = kmin(h_i) and k_j = kmin(h_j) be such that $p \leqslant h_i < h_j < 2p$. Then the inequality $k_i \geqslant h_i - k_j + 2$ holds by Lemma 5. So $k_i + k_j \geqslant h_i + 2 \geqslant p + 2$ and since $k_i \neq k_j$ either $k_i > (p+2)/2$ or $k_j > (p+2)/2$. So there is at most one point h_i such that $k_i \leqslant (p+2)/2$. Since there are no more than $(p-1)/2 - 1$ different integers that are greater than $(p+2)/2$ and less or equal to $p-1$ the inequality follows. ∎

Now we have a sufficiently deep knowledge of the structure of the relation MaxPP(k, i) and of the function kmin(i) and so we come back to the evaluation of the worst case bound for the Algorithm 4.

Let ifirst_j and ilast_j be respectively the first and the last value of i in the jth step of the repeat statement, where $\text{ifirst}_1 = 1$ and $\text{ifirst}_j = \text{next}(\text{ilast}_{j-1})$ for all $j > 1$. Then every step of the repeat statement shifts the pattern of $D_j = \text{shift}(\text{ilast}_j)$ places over the text after having done exactly $\text{ilast}_j - \text{ifirst}_j + 1$ character comparisons.

However when we start a new step, owing to the character comparisons done in the previous steps and the periodicities of the pattern, we already know that some character of the pattern matches the text in the new position.

Let neq_j be the number of characters of the text that, in the jth step, we know to match the pattern due to the character comparisons done in the previous steps. Then, to make comparable the shift of the pattern with the computational effort needed to make it, we charge the jth step with a number of comparisons C_j equal to the number of character comparisons which are done during the step plus neq_j comparisons done in the previous steps and minus neq_{j+1} comparisons which will be charged to the next step. So the total number of comparisons charged to the jth step is

$$C_j = \text{neq}_j + \text{ilast}_j - \text{ifirst}_j + 1 - \text{neq}_{j+1}.$$

However, the last step does not have a next step. So the number of character comparisons charged to the last step is

$$C_{\text{fin}} = \text{neq}_{\text{fin}} + \text{ilast}_{\text{fin}} - \text{ifirst}_{\text{fin}} + 1.$$

In the first step $\text{neq}_1 = 0$ while, to evaluate neq_j for $j > 1$, we should consider separately the two cases where $i \leqslant nd$ and $i > nd$ at the end of the previous step. We say that the jth step is short when $\text{ilast}_j \leqslant nd$ and we indicate it by S_j, otherwise we say that it is long and we indicate it by L_j.

Then, in case the previous step L_{j-1} is long, we know that, when it terminates, the characters $w_{1+h_i} \cdots w_{m-1}$ have been successfully compared with the characters $f_{b+1+h_i} \cdots f_{b+m-1}$, because of $h_1 < \cdots < h_{nd}$ and $h_{nd+1} > \cdots > h_i > \cdots > h_m$. Since $\text{shift}(i) = \text{rmin}(h_i)$ is greater than h_i and the pattern is $\text{rmin}(h_i)$ periodic, the characters $w_0 \cdots w_{m-\text{shift}(i)-1}$ of the pattern should be equal to the characters $f_{b+\text{shift}(i)} \cdots f_{b+m-1}$ of the text. So $\text{neq}_j = m - \text{shift}(\text{ilast}_{j-1})$ when the previous step L_{j-1} is long.

When the previous step S_{j-1} is short we know that there are $\text{ifirst}_j - 1$ characters of the pattern that match the text in the new position. However, in step S_{j-1} there can be some initial part of the pattern that we know to match the text due to some previous long step and the shift done in the step S_{j-1} can be shorter than such an initial segment. To deal with such a situation, let us define base_j to be the length of the initial part of the pattern that we know to match the text due to some previous long step.

Then:

(i) $\text{base}_1 = 0$;

(ii) $\text{base}_j = \max(0, \text{base}_{j-1} - \text{shift}(\text{ilast}_{j-1}))$ when $j > 1$ and $\text{ilast}_{j-1} \leqslant nd$;

(iii) $\text{base}_j = m - \text{shift}(\text{ilast}_{j-1})$ when $j > 1$ and $\text{ilast}_{j-1} > nd$.

In the prefix of length base_j of the pattern there are $\text{nhd}(0, \text{base}_j)$ positions h_p such that $1 \leqslant p \leqslant \text{ifirst}_j - 1$ and we should not compute them twice. Then $\text{neq}_j = \text{ifirst}_j + \text{base}_j - \text{nhd}(0, \text{base}_j) - 1$ when the previous step S_{j-1} is short.

Observe that $\text{ifirst}_j = \text{nhd}(0, m - \text{shift}(\text{ilast}_{j-1})) + 1 = \text{nhd}(0, \text{base}_j) + 1$ when L_{j-1} is long and that if $j = 1$ then $\text{ifirst}_1 + \text{base}_1 - \text{nhd}(0, \text{base}_1) - 1 = 1 + 0 - 0 - 1 = 0$. So

$$\text{neq}_j = \text{ifirst}_j + \text{base}_j - \text{nhd}(0, \text{base}_j) - 1$$

holds for all steps.

To simplify the notation let us define $\text{nhu}(i, j) = j - i - \text{nhd}(i, j)$ be the number of h such that $i \leqslant h < j$ and $\text{kmin}(h)$ is undefined. Then $\text{neq}_j = \text{ifirst}_j + \text{nhu}(0, \text{base}_j) - 1$ and

$$C_j = \text{nhu}(0, \text{base}_j) + \text{ilast}_j - \text{ifirst}_{j+1} + 1 - \text{nhu}(0, \text{base}_{j+1})$$

holds for all step except the last where

$$C_{\text{fin}} = \text{nhu}(0, \text{base}_{\text{fin}}) + \text{ilast}_{\text{fin}}.$$

Moreover since for a short step S_j

$$\text{ilast}_j - \text{ifirst}_{j+1} = \text{ilast}_j - \text{next}(\text{ilast}_j) = \text{nhd}(h_{\text{ilast}_j} - \text{kmin}(h_{\text{ilast}_j}), h_{\text{ilast}_j})$$

then

$$C_j = \text{nhu}(0, \text{base}_j) + \text{nhd}(h_{\text{ilast}_j} - \text{kmin}(h_{\text{ilast}_j}), h_{\text{ilast}_j}) + 1 - \text{nhu}(0, \text{base}_{j+1})$$

when the step S_j is short, and since for a long step L_j

$$\text{base}_{j+1} = m - \text{rmin}(h_{\text{ilast}_j}) \text{ and}$$

$$\text{ifirst}_{j+1} = \text{nhd}(0, m - \text{rmin}(h_{\text{ilast}_j})) + 1 = \text{nhd}(0, \text{base}_{j+1}) + 1$$

then

$$C_j = \text{nhu}(0, \text{base}_j) + \text{ilast}_j - m + \text{rmin}(h_{\text{ilast}_j})$$

when the step L_j is long.

Let us say that a short step S_j is lazy when the shift it performs does not skip over at least a character f_{b+h} of the text such that $h \geqslant \text{base}_j$ and kmin(h) is undefined and we say that S_j is active when nhu(base$_j$, shift(ilast$_j$)) > 0. Note that the inequality nhu(0, shift(ilast$_j$)) > 0 always holds, since kmin(0) is always undefined and so base$_j > 0$ whenever S_j is lazy.

To prove that the bound $1.5n + 0.5(m-1)$ holds for the whole algorithm, we group each long step with the lazy short steps that follow it. In such a way the whole sequence of steps will be split into subsequences L_j, $S_{j+1}, \ldots, S_t$ with $S_{j+1}, \ldots, S_t$ lazy short steps and into single active short steps S_j. Moreover, since the number of comparisons C_{fin} charged to the last step is computed in a different way, we deal with such a particular case separately and we do not group such a step even if it is lazy. Then the following lemma holds.

LEMMA 7. *Let Algorithm* 4 *be executed with a pattern of length m. Then*:

(I) *for all active short steps S_j with $j < \text{fin}$ the number C_j of comparisons charged to each such step is always less than or equal to* 1.5 *times the shift D_j that it performs*;

(II) *for all subsequences L_j, $S_{j+1}, \ldots, S_t$ with $t < \text{fin}$ and $S_{j+1}, \ldots, S_t$ lazy short steps, the total number $C_j + C_{j+1} + \cdots + C_t$ of comparisons charged to such steps is always less than or equal to* 1.5 *times the total shift $D_j + D_{j+1} + \cdots + D_t$ that they perform*;

(III) *the number C_{fin} of comparisons charged to the last step is always less than or equal to $2m-1$.*

Proof. (I) S_j active and $j < \text{fin}$ (see Fig. 5). In such a case base$_{j+1} = 0$ and so

$$C_j = \text{nhu}(0, \text{base}_j) + \text{nhd}(h_{\text{ilast}_j} - \text{kmin}(h_{\text{ilast}_j}), h_{\text{ilast}_j}) + 1.$$

Moreover

$$\text{nhd}(h_{\text{ilast}_j} - \text{kmin}(h_{\text{ilast}_j}), h_{\text{ilast}_j}) \leqslant \text{nhd}(0, \text{kmin}(h_{\text{ilast}_j})) + (\text{kmin}(h_{\text{ilast}_j}) - 1)/2 \qquad \text{by Lemma 6(b),}$$

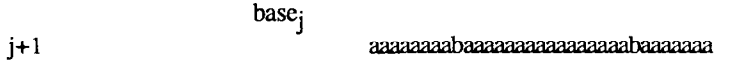

FIG. 5. Example of an active step S_j, $j < \text{fin}$. $C_j = 24$, $D_j = 18$.

$$\text{nhu}(0, \text{base}_j) + \text{nhd}(0, \text{kmin}(h_{\text{ilast}_j}))$$
$$= \text{kmin}(h_{\text{ilast}_j}) - \text{nhu}(\text{base}_j, \text{kmin}(h_{\text{ilast}_j})), \qquad \text{and}$$
$$\text{nhu}(\text{base}_j, \text{kmin}(h_{\text{ilast}_j}))$$
$$\geqslant 1 \qquad \text{because} \quad S_j \text{ is active.}$$

So

$$C_j \leqslant \text{kmin}(h_{\text{ilast}_j}) + (\text{kmin}(h_{\text{ilast}_j}) - 1)/2.$$

Since $D_j = \text{shift}(\text{ilast}_j) = \text{kmin}(h_{\text{ilast}_j})$, we conclude that $C_j \leqslant 1.5 D_j$.

(II) Let now L_j, S_{j+1}, ..., S_t with S_{j+1}, ..., S_t lazy and $t < \text{fin}$ (see Fig. 6). Then

$$\sum_j^t C_i = \text{nhu}(0, \text{base}_j) + \text{ilast}_j - \text{nhd}(0, \text{base}_{j+1}) + \sum_{j+1}^{t} (\text{nhd}(h_{\text{ilast}_i} - \text{kmin}(h_{\text{ilast}_i}), h_{\text{ilast}_i}) + 1) - \text{nhu}(0, \text{base}_{t+1}).$$

Since $t < \text{fin}$, all the steps of the sequence terminate with a mismatch. Then $h_{\text{ilast}_i} \geqslant \text{base}_i$ in all the steps of the sequence. As a consequence h_{ilast_j}, the last index tested in the long step L_j cannot be one of the last $\text{nhu}(0, \text{base}_j)$ elements in the sequence $h_1, ..., h_m$ since all of them are strictly less than base_j. So $\text{ilast}_j \leqslant m - \text{nhu}(0, \text{base}_j)$ and

$$\sum_j^t C_i \leqslant m - \text{base}_{t+1} - \text{nhd}(\text{base}_{t+1}, \text{base}_{j+1}) + \sum_{j+1}^{t} (\text{nhd}(h_{\text{ilast}_i} - \text{kmin}(h_{\text{ilast}_i}), h_{\text{ilast}_i}) + 1).$$

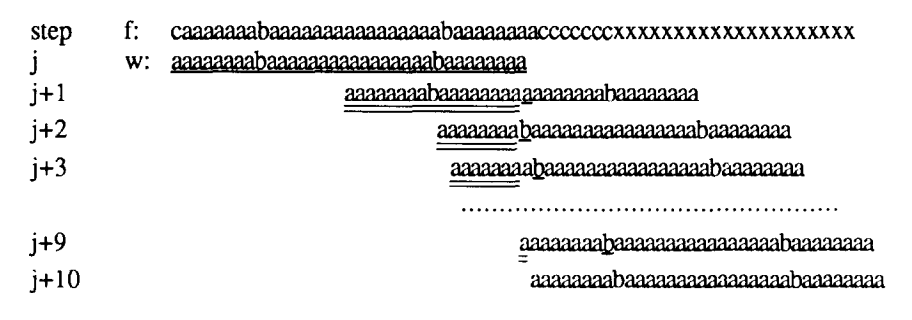

FIG. 6. Example of a sequence L_j, S_{j+1}, ..., S_t with L_j long and S_{j+1}, ..., S_t lazy, $t < \text{fin}$. $C_j = 43$, $D_j = 34$.

Now

$$\mathrm{nhd}(\mathrm{base}_{t+1}, \mathrm{base}_{j+1}) = \sum_{j+1}^{t_i} \mathrm{nhd}(\mathrm{base}_{i+1}, \mathrm{base}_i),$$

where $\mathrm{base}_{t+1} = \max(0, \mathrm{base}_t - \mathrm{kmin}(h_{\mathrm{ilast}_t}))$ and $\mathrm{base}_{i+1} = \mathrm{base}_i - \mathrm{kmin}(h_{\mathrm{ilast}_i})$ for all i such that $j+1 \leqslant i < t$. Since $h_{\mathrm{ilast}_i} \geqslant \mathrm{base}_i$ then, by Lemma 6(a), the inequality

$$\mathrm{nhd}(\mathrm{base}_{i+1}, \mathrm{base}_i) \leqslant \mathrm{nhd}(h_{\mathrm{ilast}_i} - \mathrm{kmin}(h_{\mathrm{ilast}_i}), h_{\mathrm{ilast}_i})$$

holds for all i such that $j+1 \leqslant i < t$. If $\mathrm{base}_t \geqslant \mathrm{kmin}(h_{\mathrm{ilast}_t})$ then such inequality holds for $i = t$ too. If $\mathrm{base}_t < \mathrm{kmin}(h_{\mathrm{ilast}_t})$ then $\mathrm{base}_{t+1} = 0$, $\mathrm{nhd}(\mathrm{base}_{t+1}, \mathrm{base}_t) \leqslant \mathrm{nhd}(0, \mathrm{kmin}(h_{\mathrm{ilast}_t}))$ and

$$\mathrm{nhd}(0, \mathrm{kmin}(h_{\mathrm{ilast}_t})) \leqslant \mathrm{nhd}(h_{\mathrm{ilast}_t} - \mathrm{kmin}(h_{\mathrm{ilast}_t}), h_{\mathrm{ilast}_t})$$

by Lemma 6(a). So the inequality

$$\mathrm{nhd}(\mathrm{base}_{t+1}, \mathrm{base}_t) \leqslant \mathrm{nhd}(h_{\mathrm{ilast}_t} - \mathrm{kmin}(h_{\mathrm{ilast}_t}), h_{\mathrm{ilast}_t})$$

always holds.

Moreover, by Lemma 6(b)

$$\mathrm{nhd}(h_{\mathrm{ilast}_i} - \mathrm{kmin}(h_{\mathrm{ilast}_i}), h_{\mathrm{ilast}_i}) \leqslant \mathrm{nhd}(0, \mathrm{kmin}(h_{\mathrm{ilast}_i})) + (\mathrm{kmin}(h_{\mathrm{ilast}_i}) - 1)/2$$

and so

$$\sum_{j}^{t_i} C_i \leqslant m - \mathrm{base}_{t+1} + 0.5\left(t - j + \sum_{j+1}^{t_i} \mathrm{kmin}(h_{\mathrm{ilast}_i})\right).$$

Since $\mathrm{base}_j = m - D_j$, $\mathrm{base}_{i+1} = \mathrm{base}_i - D_i$ for all i such that $j+1 \leqslant i < t$, and $\mathrm{base}_{t+1} = \max(0, \mathrm{base}_t - D_t)$, then

$$m - \mathrm{base}_{t+1} \leqslant \sum_{j}^{t_i} D_i.$$

Then, it remains to prove that $D_j = \mathrm{rmin}(h_{\mathrm{ilast}_j}) \geqslant t - j$. If $\mathrm{rmin}(h_{\mathrm{ilast}_j}) = 1$ then the pattern is p periodic for all p and so $\mathrm{kmin}(h)$ is never defined, and $t - j = 0$. So let $\mathrm{rmin}(h_{\mathrm{ilast}_j}) \geqslant 2$ and $t - j > 2$. In such a case, since $\mathrm{kmin}(h_{\mathrm{ilast}_{j+1}}) \geqslant \mathrm{base}_{j+1} - \mathrm{rmin}(h_{\mathrm{ilast}_j}) + 2$ by Lemma 5, then

$$\mathrm{base}_{j+2} = \mathrm{base}_{j+1} - \mathrm{kmin}(h_{\mathrm{ilast}_{j+1}}) \leqslant \mathrm{rmin}(h_{\mathrm{ilast}_j}) - 2$$

and from $\mathrm{rmin}(h_{\mathrm{ilast}_j}) - 2 \geqslant \mathrm{base}_{j+2} > \mathrm{base}_{j+3} > \cdots > \mathrm{base}_{t-1} > 0$ it follows that $t - j \leqslant \mathrm{rmin}(h_{\mathrm{ilast}_i}) - 1$.

(III) Let now L_{fin} long or S_{fin} short be the last iteration. Then

$$C_{\text{fin}} = \text{nhu}(0, \text{base}_{\text{fin}}) + \text{ilast}_{\text{fin}} \leqslant 2m - 1. \quad \blacksquare$$

Putting together points I, II, and III of Lemma 7 we obtain the worst case bound on the number of character comparisons for the whole execution of Algorithm 4.

THEOREM 3. *The number of comparisons between elements of w and elements of f required by Algorithm* 4 *is always less than or equal to* $1.5n + 0.5(m-1)$. *Therefore Algorithm* 4 *is* $O(n+m)$.

Proof. Summing over all the steps we obtain

$$C_{\text{tot}} = \sum_{1}^{\text{fin}-1}{}_{i} C_i + C_{\text{fin}} \leqslant 1.5 \sum_{1}^{\text{fin}-1}{}_{i} D_i + 2m - 1$$

$$= 1.5 \left(\sum_{1}^{\text{fin}-1}{}_{i} D_i + m \right) + 0.5(m-1)$$

and since $(\sum_{1}^{\text{fin}-1}{}_{i} D_i + m) \leqslant n$ we conclude that $C_{\text{tot}} \leqslant 1.5n + 0.5(m-1)$. $\blacksquare$

6. CONCLUSIONS

We have designed a pattern matching algorithm which always works better than the classical KMP algorithm and, for some problems, is better than the BM algorithm too.

This result has been obtained by some improvement steps starting from a very naive pattern matching algorithm.

The strategy we used in all the steps was always the same. We first wrote down a formal correctness proof of the algorithm (also in cases where the correctness was straightforward). Then, we looked for the presence in the algorithm of some "true" null statement. If any such statement was found then an analysis of the information forgotten by this statement was performed. The result of this analysis was then used either to devise a way to avoid the computational effort required to synthesize the forgotten information or to devise a way to use that information to lower the subsequent computational effort needed to attain the final result.

It was already know to us that the type of program statements we use to solve a programming problem may have a strong impact on the computational complexity of the program we obtain (e.g., recursive statements instead of iterative ones) (Colussi, 1984).

However, we were far from suspecting that, as the examples we had

given suggest, the main source of inefficiency of a program is related to the null statements. This seems to be very strange because the execution of a null statement does not require any computational effort at all.

It seem to us that the bad effect of null statements on program efficiency is a phenomenon very similar to the bad effect of goto statements on program correctness. Indeed, also this latter effect seem to be very strange because the proof of the correctness of a goto statement does not require any deductive effort at all (what is true before the goto is also true after the jump).

We conclude with a list of open questions that will be matter for our future work:

(a) We already noted, in Section 5, that there is a loss of information when we deduce $\mathrm{Eqh}(b+\mathrm{shift}(i), \mathrm{next}(i)-1))$ from $\mathrm{Eqh}(b, i-1)$ and $f_{b+h_i} \neq w_{h_i}$ in the correctness proof of Algorithm 4 and Lemma 6 gives a bound for such a loss of information. Is it possible to further improve Algorithm 4 by exploiting such information?

(b) The worst case bound $1.5n+0.5(m-1)$ we have proved in Section 5 is only an approximation of the true upper bound. Can we determine the true upper bound?

(c) The test cases reported in Table 1 for Algorithm 4 suggest that the upper bound for the average value of c/n for all pattern is 1 (or very close to 1). Can we determine exactly this upper bound?

(d) The analysis we have done in Section 5 to prove the worst case bound $1.5n+0.5(m-1)$ gives some new results about combinatorial properties of strings and, in the attempt to find the upper bound of point (c), we have found some other nice combinatorial properties of strings. Since those results can be valuable independently from the present use it will be useful to organize them in a systematic way.

RECEIVED October 24, 1988; FINAL MANUSCRIPT RECEIVED May 31, 1990

REFERENCES

ALAGIĆ, S., AND ARBIB, M. A. (1978), "The Design of Well-Structured and Correct Programs," Springer-Verlag, New York.

BOYER, R. S., AND MOORE, J. S. (1977), A fast string searching algorithm, *Comm. ACM* **20**, 762.

COLUSSI, L. (1984), Recursion as an effective step in program development, *ACM Trans. Programming Languages Systems* **6**, 55.

DEBAKKER, J. W. (1980), "Mathematical Theory of Program Correctness," Prentice–Hall, Englewood Cliffs, NJ.

DIJKSTRA, E. W. (1976), "A discipline of Programming," Prentice–Hall, Englewood Cliffs, NJ.

GALIL, Z., AND SEIFERAS, J. (1980), Saving space in fast string matching, *SIAM J. Comput.* **9**, 417.

GUIBAS, L. J., AND ODLYZKO, A. M. (1980), A new proof of the linearity of the Boyer–Moore string searching algorithm, *SIAM J. Comput.* **9**, 672.

HOARE, C. A. R., AND WIRTH, N. (1973), An axiomatic definition of the programming language PASCAL, *Acta Inform.* **2**, 335.

KNUTH, D. E., MORRIS, J. H., AND PRATT, V. B. (1977), Fast pattern-matching in strings, *SIAM J. Comput.* **6**, 323.