JOURNAL OF ALGORITHMS 16, 163–189 (1994)

Fastest Pattern Matching in Strings

LIVIO COLUSSI*

Dipartimento di Matematica Pura ed Applicata, University of Padova, Via Belzoni 7, 35131 Padua, Italy

Received February 1991; accepted May 1993

An algorithm is presented that substantially improves the algorithm of Boyer and Moore for pattern matching in strings, both in the worst case and in the average. Both the Boyer and Moore algorithm and the new algorithm assume that the characters in the pattern and in the text are taken from a given alphabet Σ of finite size. The new algorithm performs $2n$ character comparisons in the worst case while the Boyer and Moore algorithm requires $3n$ comparisons [4]; the new algorithm requires fewer comparisons than Boyer and Moore on the average (but for the case of a binary alphabet, where the two algorithms perform roughly the same). For large patterns the ratio between the average number of comparisons for Boyer and Moore algorithm and the average number of comparisons for the new algorithm is close to the size $|\Sigma|$ of the alphabet. As a shortcoming of the new algorithm, the preprocessing of the pattern requires $O(m)$ time on the average but $O(m^2)$ in the worst case. A mixed strategy between the two algorithms is suggested in order to make the preprocessing linear, at the expense of a slightly less efficient performance of the algorithm. The new algorithm has been obtained from the Boyer and Moore algorithm using a correctness proof (in the style of Hoare's axiomatic semantics) as a tool to improve algorithms.

1. INTRODUCTION

Pattern matching is the problem of finding all occurrences of a word $w = w_0 \dots w_{m-1}$ of length m in a text $t = t_0 \dots t_{n-1}$ of length n. The best known algorithms for this problem are due to Knuth, Morris, and Pratt [12] and to Boyer and Moore [3]. Both the algorithms are in linear time. The worst-case bound of the number of characters inspected for the algorithm of Knuth, Morris, and Pratt (KMP for short) is $2n - m$ and this is a tight bound for $m \geq 2$. A first bound of $7n$ for the Boyer and Moore algorithm (BM for short) was given by Knuth in [12], a bound of $4n$ was

*email:colussi@pdmat1.unipd.it

0196-6774/94 $6.00

given by Guibas and Odlyzko [10], and last, a bound of $3n$ is given by Cole in [4], where it is also proved that this bound is tight. A modified version of BM that performs $2n$ character comparisons in the worst case is proposed by Apostolico and Giancarlo [2]. The algorithm proposed in [2] is not practical due to the complicated data structures involved. Thus KMP is definitely superior to BM with respect to the worst-case behaviour. However, BM is sublinear on the average and the number of inspected characters decreases as a function of the length of the pattern. This makes BM superior to KMP in practical applications.

Both the KMP and the BM algorithms begin by computing a shift function to be used in case of a mismatch is detected between a character of the pattern and a character of the text. On the other end KMP compares the characters of the pattern with the corresponding characters of the text from left to right, while BM proceeds in the reverse order. Moreover, BM makes use of another precomputed shift function (occurrence function) that depends on the character inspected in the text. Thus BM needs to know the alphabet while KMP does not.

Recently, in [5], the author proposed a new pattern matching algorithm (C for short) and proved that it performs $1.5n$ character comparisons in the worst case. Galil and Giancarlo in [8] used a slightly modified version of the algorithm C to obtain an upper bound of $\frac{4}{3}n$ and in [9] they proved that this bound is tight for on line algorithms. All those results are jointly presented in [6].

The algorithm C was obtained by successive transformations starting from a naive algorithm that compares from left to right the characters of the pattern with the characters of the text. In all the transformation steps we used the same technique:

- make a formal correctness proof of the algorithm (in the style of Hoare's axiomatics);
- look for points in the proof where we used the null statement rule of correctness

$$\frac{P \Rightarrow Q}{\{P\}\{Q\}}$$

and in case that the converse $Q \Rightarrow P$ does not hold look for some information in the precondition P that does not appear in Q;
- modify the algorithm either to avoid the computation of such an information or to exploit the information itself to speed up the subsequent computations.

Two steps are needed to obtain the KMP algorithm from the naive one.

More important, using this technique we are able to make two more steps, thus obtaining the algorithm C.

Such a technique can be applied to obtain BM algorithm starting from a naive algorithm that compares from right to left the characters of the pattern with the characters of the text. More importantly, we can obtain an algorithm RC (for reverse C) that substantially improves BM. Indeed, RC performs $2n$ comparisons in the worst case and improves the asymptotic (with respect to the pattern length m) average behaviour of BM for a factor of order $|\Sigma|$, where $|\Sigma|$ is the size of the alphabet. Recall that BM assumes that the characters in the pattern and in the text are taken from a given alphabet Σ of finite size. This holds for RC algorithm too.

A shortcoming of RC is that the preprocessing of the pattern, as proposed in Section 5, requires $O(m)$ time on the average but is $O(m^2)$ in the worst case while the preprocessing for BM is $O(m)$ both in the average and in the worst case. The $O(m^2)$ bound is due to the computation of a table Δ_1 of size $|\Sigma| \times m$. However, in Section 5, a mixed strategy is proposed that uses only a portion of size $|\Sigma|^3$ of the table Δ_1 and uses the table δ_1 of BM in place of the unused portion of Δ_1. This makes $O(m)$ the preprocessing in the worst case (at the expense of a slight loss of efficiency of the pattern matching algorithm).

For the sake of brevity we start the derivation from BM and we omit the derivation of BM from the naive algorithm. The paper is organized as follows. In Section 2 the steps needed to obtain RC from BM are reported. In Section 3 the results of some test are reported and implementation hints are given. Section 4 contains the complexity analysis of RC. In Section 5, the preprocessing of the pattern is discussed. Last, in Section 6 some notes about the method we used to improve algorithms are given.

2. The Derivation of RC from BM

Let $w = w_0 \dots w_{m-1}$ be the pattern and let $t = t_0 \dots t_{n-1}$ be the text. We use the notation $w_{[i:j]}$ for the segment of w between the character w_i and the character w_j. The same notation is used for segments of t. Moreover, we say that the pattern w is k-periodic whenever $w_i = w_{i-k}$ for all i such that $k \le i \le m-1$ or, equivalently, if $w_{[k:m-1]} = w_{[0:m-k-1]}$. Note that w is trivially zero-periodic and m-periodic. The (principal) period π_w of the pattern is the minimum positive integer such that w is π_w-periodic.

The algorithm BM assumes that the alphabet Σ is known and makes use of two tables δ_1 and δ_2 to shift the pattern over the text. The table δ_1 has an entry for each character in Σ and the table δ_2 has an entry for each

integer from -1 to $m-1$. A simple way to define δ_1 and δ_2 and to show its use in BM is as follows.

For each integer i such that $-1 \le i \le m-1$ and for each character c in Σ let the set of integers from zero to i be split into the two sets,

$$\begin{aligned} \mathrm{Pos}(c,i) &= \{k \mid 0 \le k \le i \text{ and } c = w_{i-k}\} \\ \mathrm{Neg}(c,i) &= \{k \mid 0 \le k \le i \text{ and } c \ne w_{i-k}\}. \end{aligned} \tag{1}$$

Note that for all i such that $-1 \le i < m - \pi_w$ and for all $c \in \Sigma$ we have that $\mathrm{Pos}(c,i) \subseteq \mathrm{Pos}(c, i+\pi_w)$ and $\mathrm{Neg}(c,i) \subseteq \mathrm{Neg}(c, i+\pi_w)$. The names *Pos* and *Neg* are due to the following fact. Let the pattern be aligned with the text in position b (i.e., w is over the segment $t_{[b:b+m-1]}$ of the text). Then the following properties hold:

P1. if $w_i \ne t_{b+i}$ for some i such that $0 \le i \le m-1$ then the pattern cannot match the text in all the positions $b+k$ such that $k \in \mathrm{Pos}(w_i, i)$;

P2. if $w_i = t_{b+i}$ for some i such that $0 \le i \le m-1$ then the pattern cannot match the text in all the positions $b+k$ such that $k \in \mathrm{Neg}(w_i, i)$.

Then $\delta_2(i)$ is defined as the minimum positive integer k that does not belong either to the set $\mathrm{Pos}(w_i, i)$ or to any set $\mathrm{Neg}(w_j, j)$ such that $i+1 \le j \le m-1$. That is,

$$\delta_2(i) = \min\left\{k \mid k \notin \mathrm{Pos}(w_i, i) \cup \bigcup_{j=i+1}^{m-1} \mathrm{Neg}(w_j, j)\right\}. \tag{2}$$

Note that

- m does not belong to any set $\mathrm{Pos}(w_i, i)$ or $\mathrm{Neg}(w_i, i)$
- an integer k such that $1 \le k \le m-1$ does not belong to any set $\mathrm{Neg}(w_i, i)$ if and only if w is k-periodic and
- $\mathrm{Pos}(w_i, i)$ contains all integers k such that $0 \le k \le i$ and w is k-periodic.

It follows that $\delta_2(i) \le m$ and, since $\mathrm{Pos}(w_{-1}, -1) = \varnothing$ and $\mathrm{Pos}(w_0, 0) = \{0\}$, then $\delta_2(-1) = \delta_2(0) = \pi_w$.

BM uses the following property of δ_2:

P3. if we know that $w_{[i+1:m-1]} = t_{[b+i+1:b+m-1]}$ and that $w_i \ne t_{b+i}$ then w cannot match t when it is aligned with $t_{[b+k:b+k+m-1]}$ in all the positions $b+k$ such that $1 \le k < \delta_2(i)$.

The proof of property P3 uses properties P1 and P2. Let k be such that $1 \le k < \delta_2(i)$, then either $k \in \mathrm{Pos}(w_i, i)$ and the thesis follows from

property P1 or $k \in \text{Neg}(w_j, j)$ for a j such that $i + 1 \leq j \leq m - 1$ and the thesis follows from property P2.

Table δ_1 has an entry for each character in the alphabet and is defined on the basis of the following property:

P4. if we know that $t_{b+i} = c$ then w cannot match t when it is aligned with $t_{[b+k:b+k+m-1]}$ for all k such that $k \in \text{Neg}(c, i)$.

BM uses property P4 as follows. Assume that $w_{[i+1:m-1]} = t_{[b+i+1:b+m-1]}$ and $w_i \neq t_{b+i}$ and let $\delta(t_{b+i}, i)$ be the minimum positive integer that does not belong to the set $\text{Neg}(t_{b+i}, i)$. Then the pattern can be shifted safely of $\max(\delta_2(i), \delta(t_{b+i}, i))$ positions.

If the character t_{b+i} does not occur in $w_{[i+1:m-1]}$, then $\delta(t_{b+i}, i) = \delta(t_{b+i}, m - 1) + i - m + 1$. On the other hand, if t_{b+i} occurs in $w_{[i+1:m-1]}$, assume that l is the minimum, such that $i + 1 \leq l \leq m - 1$ and $t_{b+i} = w_l$. Then $\delta(t_{b+i}, i) = \delta(w_l, l) + i - l$. However, since the set $\text{Neg}(w_l, l)$ belongs to the set $\text{Pos}(w_i, i) \cup \bigcup_{j=i+1}^{m-1} \text{Neg}(w_j, j)$, then $\delta(w_l, l) \leq \delta_2(i)$, and a fortiori $\delta(t_{b+i}, i) \leq \delta_2(i)$. Thus $\max(\delta_2(i), \delta(t_{b+i}, i))$ is always equal to $\max(\delta_2(i), \delta(t_{b+i}, m - 1) + i - m + 1)$ and the shift can be computed by using only the two tables $\delta_2(i)$ and $\delta_1(c) = \delta(c, m - 1)$.

The BM algorithm, as proposed in [3], contains two loops, a "fast" one inside a "slow" one. The fast loop compares the last character w_{m-1} of the pattern with the corresponding character t_{b+m-1} of the text and makes use of table δ_1 alone to shift the pattern (because $\delta_1(t_{b+m-1}) \geq \delta_2(m - 1)$ always holds).

The fast loop is repeated until either the text is completely scanned or a position b is found such that $w_{m-1} = t_{b+m-1}$. The slow loop uses either of δ_1 or δ_2 testing for $\max(\delta_2(i), \delta_1(t_{b+i}) + i - m + 1)$.

In Appendix A we show that, under the assumption that the alphabet Σ contains at least two symbols and that the pattern is chosen at random in Σ^m, then the average $\tilde{N}(i)$ of $\min\{k | k \notin \text{Neg}(c, i)\}$ is

$$\tilde{N}(i) = \sum_{j=0}^{i} \left(\frac{|\Sigma| - 1}{|\Sigma|} \right)^j = |\Sigma| \left[1 - \left(\frac{|\Sigma| - 1}{|\Sigma|} \right)^{i+1} \right] \tag{3}$$

while the average $\tilde{P}(i)$ of $\min\{k | k \notin \text{Pos}(c, i)\}$ is

$$\tilde{P}(i) = \sum_{j=0}^{i} \frac{1}{|\Sigma|^j} = \frac{|\Sigma|}{|\Sigma| - 1} \left(1 - \frac{1}{|\Sigma|^{i+1}} \right). \tag{4}$$

Thus,

$$\lim_{i \to \infty} \tilde{N}(i) = |\Sigma| \qquad \text{and} \qquad \lim_{i \to \infty} \tilde{P}(i) = \frac{|\Sigma|}{|\Sigma| - 1}$$

and so, when i becomes large $\tilde{N}(i)$ becomes close to $|\Sigma|$ and $\tilde{P}(i)$ becomes close to $|\Sigma|/|\Sigma| - 1$. In Appendix A it is also shown that the inequalities

$$\left(1 - \frac{1}{e}\right)\min(i + 1, |\Sigma|) \leq \tilde{N}(i) \leq \min(i + 1, |\Sigma|), \qquad 1 \leq \tilde{P}(i) \leq 2 \tag{5}$$

hold for all i.

When $i < m - 1$, the union $\mathrm{Pos}(w_i, i) \cup \bigcup_{j=i+1}^{m-1} \mathrm{Neg}(w_j, j)$ contains the set $\mathrm{Neg}(w_{m-1}, m - 1)$. Thus the average of $\delta_2(i)$ is greater than $\tilde{N}(m - 1)$. On the other hand, the average of $\delta_1(t_{b+i}) + i - m + 1$ is equal to $\tilde{N}(m - 1) + i - m + 1$, and so it is less than the average of $\delta_2(i)$. Thus, the gain in efficiency due to the use of δ_1 in the slow loop is small and does not compensate the loss due to the computation of the expression $\max(\delta_2(i), \delta_1(t_{b+i}) + i - m + 1)$.

Thus we start with the following simplified version of BM algorithm that uses δ_1 alone in the fast loop and δ_2 alone in the slow loop.

```
b := 0;
repeat
  while w_{m-1} ≠ t_{b+m-1} do
    begin
      s := δ1(t_{b+m-1});
      b := b + s;
      if b > n - m then exit
    end;
  i := m - 2;
  while i ≥ 0 and w_i = t_{b+i} do i := i + 1;
  if i = -1 then match found at b;
  s := δ2(i);
  b := b + s
until b > n - m
```

To obtain a correctness proof of the BM algorithm, we use the assertion "all matches of w in $t_{[0:b+m-1]}$ have been found," AMF(b) for short, as an invariant for both the repeat statement (the slow loop) and the first while statement (the fast loop). The invariant for the second while statement is "AMF(b) **and** $w_{[i+1:m-1]} = t_{[b+i+1:b+m-1]}$." Moreover, $\delta_1(c)$ may be characterized as the (only) k that satisfies the assertion "the first occurrence of c from the end of the pattern is k characters before the

end." Formally,

$$\text{focc}(c, k) \equiv_{\text{def}} (c \notin w_{[m-k:m-1]}) \textbf{ and } (k = m \textbf{ or } w_{m-k-1} = c),$$

where the disjunct $k = m$ is needed to hold the case where c does not occur in w.

The algorithm BM decorated with the invariants needed for the correctness proof is as follows.

$b := 0;$
repeat
  {AMF($b - 1$)}
  while $w_{m-1} \neq t_{b+m-1}$ **do**
    begin
      $s := \delta_1(t_{b+m-1})$;
A {AMF($b - 1$) **and** focc(t_{b+m-1}, s)}
B {AMF($b + s - 1$)}
      $b := b + s$; {AMF($b - 1$)}
      if $b > n - m$ **then** {AMF($n - m$)} exit
    end;
  $i := m - 2$;
    {AMF($b - 1$) **and** $w_{[i+1:m-1]} = t_{[b+i+1:b+m-1]}$ **and** $i \geq -1$}
  while $i \geq 0$ **and** $w_i = t_{b+i}$ **do** $i := i - 1$;
    {AMF($b - 1$) **and** $w_{[i+1:m-1]} = t_{[b+i+1:b+m-1]}$
    and ($i = -1$ **or** $w_i \neq t_{b-i}$)}
  if $i = -1$ **then** {AMF($b - 1$) **and** $w_{[0:m-1]} = t_{[b:b+m-1]}$} match found at b;
C {AMF(b) **and** $w_{[i+1:m-1]} = t_{[b+i+1:b+m-1]}$
    and ($i = -1$ **or** $w_i \neq t_{b+i}$)}
D {AMF($b + \delta_2(i) - 1$)}
  $s := \delta_2(i)$;
    {AMF($b + s - 1$)}
  $b := b + s$
    {AMF($b - 1$)}
until $b > n - m$
  {AMF($n - m$)}

Note that there are two points where the proof rule of a null statement is used. The first point is in the fast loop between assertions A and B and the second point is in the slow loop between assertions C and D. Let us consider first the null statement in the slow loop. To show that it is correct, i.e., that C $\Rightarrow$ D, we used property P3. If we look carefully to the proof of property P3 we can observe that the thesis follows because for all k such that $1 \leq k < \delta_2(i)$, either $k \in \text{Pos}(w_i, i)$ or there exists a j between

$i + 1$ and $m - 1$ such that $k \in \text{Neg}(w_j, j)$; i.e.,

$$k \in \text{Pos}(w_i, i) \cup \bigcup_{j=i+1}^{m-1} \text{Neg}(w_j, j).$$

Thus, we do not need to know that $w_h = t_{b+h}$ for all the indexes h from $i + 1$ to $m - 1$. It suffices that this equality holds for a subset $h_1, \ldots, h_d$ such that $k \in \text{Pos}(w_i, i) \cup \bigcup_{j=1}^{d} \text{Neg}(w_{h_j}, h_j)$. It follows that there is a loss of information between C and D in the slow loop. Thus, we may try to avoid the computation of such information by choosing a suitable order $h_1, \ldots, h_m$ to compare the characters of the pattern with the characters of the text. Obviously we need a different shift function $\Delta_2(i)$ to be used when a mismatch is found in comparing the character w_{h_i} of the pattern with the corresponding character of the text.

A good strategy to chose the order $h_1, \ldots, h_m$ and to compute the shift function $\Delta_2(i)$ is as follows. Let $h_{\min}(k)$ be defined, for all k such that $1 \le k \le m$, as the minimum integer h such that $h \ge k - 1$ and $k \notin \text{Neg}(w_j, j)$ for all j such that $h < j \le m - 1$. Since $h = m - 1$ obviously satisfies the condition for all k, then $h_{\min}(k)$ is always defined and $k - 1 \le h_{\min}(k) \le m - 1$. Note that

- $w_{[h_{\min}(k)-k+1:m-1]}$ is the maximum k-periodic postfix of the pattern w,
- $h_{\min}(k) = k - 1$ if and only if the whole pattern w is k-periodic, and
- $h_{\min}(k) \ge m - \pi_w$ whenever $h_{\min}(k) \ge k$, since $k \in \text{Neg}(w_{j+\pi_w}, j + \pi_w)$ whenever $j < m - \pi_w$ and $k \in \text{Neg}(w_j, j)$.

Moreover, for all h such that $0 \le h \le m - 1$, let $k_{\min}(h)$ be defined as the minimum integer k such that $h_{\min}(k) = h \ge k$ if any such k exists, $k_{\min}(h) = 0$ otherwise. Note that $k_{\min}(h)$ satisfies the following properties:

- $1 \le k_{\min}(h_{\min}(k)) \le k$ whenever $h_{\min}(k) \ge k$;
- $h_{\min}(k_{\min}(h)) = h$, $k_{\min}(h) \notin \text{Pos}(w_h, h)$ and $h \ge m - \pi_w$ whenever $k_{\min}(h) \ne 0$;
- $k_{\min}(h) \ne k_{\min}(j)$ whenever $k_{\min}(h) \ne 0$ and $h \ne j$.

Last, for all h such that $0 \le h \le m - 1$, let $r_{\min}(h)$ be defined as the minimum integer r such that $r > h$ and $h_{\min}(r) = r - 1$ (i.e., w is r-periodic). For $h_{\min}(m) = m - 1$, $r_{\min}(h)$ is always defined and $h < r_{\min}(h) \le m$.

Then, using the functions $k_{\min}(h)$ and $r_{\min}(h)$, we can choose $h_1, \ldots, h_m$ and compute $\Delta_2(i)$ as follows:

i. When the fast loop terminates we already know that $w_{m-1} = f_{b+m-1}$. So we choose $h_1 = m - 1$. We do not define $\Delta_2(1)$ since only Δ_1 is used in the fast loop.

ii. Then we choose, in increasing order of $k_{\min}(h)$, all the indexes $h_2, \ldots, h_d$, other than h_1 and such that $k_{\min}(h_i) \neq 0$. Then, we take $\Delta_2(i) = k_{\min}(h_i)$ (since for all k such that $1 \leq k < k_{\min}(h_i)$ either $k \in \mathrm{Pos}(w_{h_i}, h_i)$ or $k = k_{\min}(h_j)$ for a j less than i and so $k \in \mathrm{Neg}(w_{h_j}, h_j)$). This ensures that $\Delta_2(i) \geq i - 1$ and that $d \leq \pi_w$ (since $h \geq m - \pi_w$ whenever $k_{\min}(h) \neq 0$).

iii. Suppose we have chosen $h_1, \ldots, h_d$ as in points i and ii. Then, for all k such that $1 \leq k \leq m$, either $h_{\min}(k) = k - 1$ (and so w is k-periodic) or $k_{\min}(h_{\min}(k)) \geq 1$ (and so $h_{\min}(k) = h_j$ for some j such that $1 \leq j \leq d$). Thus, $\bigcup_{j=1}^{d} \mathrm{Neg}(w_{h_j}, h_j) = \bigcup_{j=0}^{m-1} \mathrm{Neg}(w_j, j)$. Then the set $\bigcup_{j=1}^{d} \mathrm{Neg}(w_{h_j}, h_j)$ contains all integers k such that $1 \leq k \leq m$ and w is not k-periodic. Since $\mathrm{Pos}(w_h, h)$ contains all integers k such that $0 \leq k \leq h$ and w is k-periodic, then for all h_i other than $h_1, \ldots, h_d$, we can take $\Delta_2(i) = r_{\min}(h_i)$, irrespective of the order in which they are considered. We choose $h_{d+1}, \ldots, h_m$ in increasing order, since this ensures a worst-case bound of $2n$ character comparisons, as we will show later.

iv. To handle uniformly the case where a match is found and $i = m + 1$ we put $\Delta_2(m + 1) = \pi_w$ (equivalently: $h_{m+1} = -1$).

The algorithm becomes:

```
   b := 0;
   repeat
     {AMF(b − 1)}
     while w_{m−1} ≠ t_{b+m−1} do
       begin
         s := δ1(t_{b+m−1});
A          {AMF(b − 1) and focc(t_{b+m−1}, s)}
B          {AMF(b + s − 1)}
         b := b + s;
           {AMF(b − 1)}
         if b > n − m then {AMF(n − m)} exit
       end;
     i := 2;
       {AMF(b − 1) and ∀q ∈ [1 : i − 1](w_{h_q} = t_{b+h_q}) and i ≤ m + 1}
     while i ≤ m and w_{h_i} = t_{b+h_i} do i := i + 1;
       {AMF(b − 1) and ∀q ∈ [1 : i − 1](w_{h_q} = t_{b+h_q})
       and (i = m + 1 or w_{h_i} ≠ t_{b+h_i})}
```

if $i = m + 1$ **then** {AMF($b - 1$) **and** $w_{[0:m-1]} = t_{[b:b+m-1]}$} match found at b;
{AMF(b) **and** $\forall q \in [1 : i - 1](w_{h_q} = t_{b+h_q})$
and ($i = m + 1$ **or** $w_{h_i} \neq t_{b+h_i}$)}
{AMF($b + \Delta_2(i) - 1$)}
$s := \Delta_2(i)$;
{AMF($b + s - 1$)}
$b := b + s$
{AMF($b - 1$)}
until $b > n - m$
{AMF($n - m$)}

To speed up the fast loop, we observe that the part

$$s = m \textbf{ or } w_{m-s-1} = t_{b+m-1}$$

of the assertion $\text{focc}(t_{b+m-1}, s)$ is not used to infer B from A. In order to use such information to speed up the subsequent computations, we add it to B and we add the assertion

$$s = m \textbf{ or } w_{m-s-1} = t_{b+m-s-1}$$

to the loop invariant. Now the assertion

$$\text{AMF}(b - 1) \textbf{ and } (s = m \textbf{ or } w_{m-s-1} = t_{b+m-s-1}) \textbf{ and } w_{m-1} \neq t_{b+m-1}$$

holds before to compute a shift in the fast loop. It is easy to see that, in such a situation, the pattern cannot match the text in any position $b + k$ such that either $k \in \text{Neg}(w_{m-s-1}, m - s - 1)$ or $k \in \text{Neg}(t_{b+m-1}, m - 1)$. Thus, we can define a new shift function

$$\Delta_1(c, s) = \min\{k \mid k \notin \text{Neg}(w_{m-s-1}, m - s - 1) \cup \text{Neg}(c, m - 1)\}.$$

Note that an integer k does not belong to the set $\text{Neg}(w_{m-s-1}, m - s - 1) \cup \text{Neg}(c, m - 1)$ if and only if it satisfies the assertion

$$R(k) \equiv_{\text{def}} (k \geq m \textbf{ or } w_{m-k-1} = c) \textbf{ and } (k \geq m - s \textbf{ or } w_{m-k-s-1} = w_{m-s-1}).$$

Then $\Delta_1(c, s)$ may be characterized as the minimum k such that $R(k)$ holds.

The assertion

$$s = m \textbf{ or } w_{m-s-1} = t_{b+m-s-1}$$

holds at the end of the slow loop too. This ensures that the invariant of

the fast loop holds when the fast loop is entered after the execution of a slow loop. We need only to initialize the variable s to m to make the invariant hold the first time that the fast loop is executed. The final version of the algorithm is:

```
b := 0;
s := m;
repeat
  {AMF(b − 1) and (s = m or w_{m−s−1} = t_{b+m−s−1})}
  while w_{m−1} ≠ t_{b+m−1} do
    begin
      s := Δ1(t_{b+m−1}, s);
        {AMF(b − 1) and occ(t_{b+m−1}, s)}
        {AMF(b + s − 1) and (s = m or w_{m−s−1} = t_{b+m−1})}
      b := b + s;
        {AMF(b − 1) and (s = m or w_{m−s−1} = t_{b+m−s−1})}
      if b > n − m then {AMF(n − m)} exit
    end;
  i := 2;
    {AMF(b − 1) and ∀q ∈ [1 : i − 1](w_{h_q} = t_{b+h_q}) and i ≤ m + 1}
  while i ≤ m and w_{h_i} = t_{b+h_i} do i := i + 1;
    {AMF(b − 1) and ∀q ∈ [1 : i − 1](w_{h_q} = t_{b+h_q})
    and (i = m + 1 or w_{h_i} ≠ t_{b+h_i})}
  if i = m + 1 then {AMF(b − 1) and w_{[0:m−1]} = t_{[b:b+m−1]}} match
  found at b;
    {AMF(b) and ∀q ∈ [1 : i − 1](w_{h_q} = t_{b+h_q})
    and (i = m + 1 or w_{h_i} ≠ t_{b+h_i})}
  s := Δ2(i);
    {AMF(b + Δ2(i) − 1) and (s = m or w_{m−s−1} = t_{b+m−1})}
  b := b + s
    {AMF(b − 1) and (s = m or w_{m−s−1} = t_{b+m−s−1})}
until b > n − m
  {AMF(n − m)}
```

3. The Result of Some Tests

Both BM and RC have been written in PASCAL and extensively tested for various values of the alphabet size $|\Sigma|$ and of the pattern length m. For each value of $|\Sigma|$ and m, a text of 10,000 random characters has been generated and both BM and RC have been run with 100 random generated patterns of length m. The average number of comparisons, the

TABLE 1
Comparison between BM and RC

		BM					RC				
$\|\Sigma\|$	m	C_{ave}	$\delta 1$	$\delta 2/C$	T	%fast	C_{ave}	$\Delta 1$	$\Delta 2/C$	T	%fast
2	2	9998	1.45	0.78	494	33	9998	1.45	0.78	610	33
2	5	8419	1.94	0.97	330	23	8153	2.21	0.92	396	24
2	10	6143	2.18	1.47	232	22	5728	3.22	1.32	267	23
2	20	4531	1.87	2.30	170	22	4096	3.53	2.12	190	22
2	40	3303	1.88	3.34	123	22	3204	4.05	2.84	148	22
2	80	2656	2.07	4.21	101	22	2652	4.34	3.57	123	22
2	160	2126	1.94	5.39	79	23	2225	4.26	4.49	104	22
2	320	1755	1.94	6.56	66	22	1887	4.49	5.37	87	22
2	640	1461	1.84	7.82	55	22	1564	4.25	6.58	73	22
5	2	6669	1.81	0.88	296	67	6669	1.81	0.88	383	67
5	5	3739	3.38	1.53	156	62	3687	3.44	1.51	200	62
5	10	2877	4.07	2.51	120	62	2460	5.02	2.48	133	62
5	20	2256	4.84	3.77	93	62	1446	8.97	3.41	78	63
5	40	2032	4.61	5.37	84	62	897	14.86	4.67	50	63
5	80	1785	4.89	6.65	74	62	633	21.64	5.46	34	63
5	160	1577	4.76	8.71	65	62	540	24.68	6.75	29	64
5	320	1375	4.89	10.61	57	62	525	24.33	8.54	28	63
5	640	1189	5.07	12.54	49	62	492	24.98	9.04	27	63
26	2	5291	1.96	0.97	235	93	5291	1.96	0.97	312	93
26	5	2248	4.63	2.28	100	92	2247	4.63	2.29	132	92
26	10	1235	8.45	3.90	54	92	1231	8.47	3.90	72	92
26	20	733	14.16	7.03	33	92	715	14.56	6.86	42	92
26	40	503	20.53	11.27	23	93	433	24.00	10.93	26	93
26	80	414	24.91	13.33	18	92	252	41.87	12.64	15	92
26	160	400	25.21	18.27	18	92	134	78.47	16.25	8	92
26	320	386	25.25	23.76	17	92	71	148.29	16.78	4	92
26	640	368	25.59	24.48	16	92	38	284.81	18.43	2	92

average shift in the fast loop, the average shift per character comparison in the slow loop, the average execution time, and the percent of comparisons done in the fast loop has been computed. The results are listed in Table 1.

We outline the following facts about data reported in Table 1:

• The average $\Delta 1$ of the shifts done in the fast loop of RC is always greater than the average $\delta 1$ of the shifts done in the fast loop of BM. For large m, $\delta 1$ gets close to $|\Sigma|$, in accordance with the theoretical value $\delta 1 = \tilde{N}(m-1) = |\Sigma|[1 - ((|\Sigma| - 1)/|\Sigma|)^m]$. For large m, $\Delta 1$ gets close

TABLE 2
Comparison between Theoretical and Experimental Values of $\delta 1$ and $\Delta 1$

	$\lvert\Sigma\rvert = 2$				$\lvert\Sigma\rvert = 5$				$\lvert\Sigma\rvert = 26$			
m	$\delta 1$	$\tilde{\delta} 1$	$\Delta 1$	$\tilde{\Delta} 1$	$\delta 1$	$\tilde{\delta} 1$	$\Delta 1$	$\tilde{\Delta} 1$	$\delta 1$	$\tilde{\delta} 1$	$\Delta 1$	$\tilde{\Delta} 1$
2	1.45	1.50	1.45	1.50	1.81	1.80	1.81	1.80	1.96	1.96	1.96	1.96
5	1.94	1.94	2.21	2.53	3.38	3.36	3.44	3.68	4.63	4.63	4.63	4.65
10	2.18	2.00	3.22	3.45	4.07	4.46	5.02	6.17	8.45	8.43	8.47	8.67
20	1.87	2.00	3.53	3.94	4.84	4.94	8.97	10.2	14.2	14.1	14.6	15.6
40	1.88	2.00	4.05	4.00	4.61	5.00	14.9	16.1	20.5	20.6	24.0	27.6
80	2.07	2.00	4.34	4.00	4.89	5.00	21.6	22.1	24.9	24.9	41.9	48.6
160	1.94	2.00	4.26	4.00	4.76	5.00	24.7	24.8	25.2	25.9	78.5	87.5
320	1.94	2.00	4.49	4.00	4.89	5.00	24.3	25.0	25.3	26.0	148.	158.
640	1.84	2.00	4.25	4.00	5.07	5.00	25.0	25.0	25.6	26.0	285.	279.

to $|\Sigma|^2$. The true theoretical value of $\Delta 1$ is not easy to find since the two sets $\mathrm{Neg}(w_{m-s-1}, m-s-1)$ and $\mathrm{Neg}(c, m-1)$ used in the definition of $\Delta_1(c, s)$ are not independent each other. However, the dependency between $\mathrm{Neg}(w_{m-s-1}, m-s-1)$ and $\mathrm{Neg}(c, m-1)$ becomes smaller as $|\Sigma|$ becomes larger. In Appendix B, a theoretical value $\tilde{\Delta} 1$ is computed under the hypothesis that the two sets are independent, and it is shown that $\lim_{m\to\infty} \tilde{\Delta} 1 = |\Sigma|^2$. In Table 2 theoretical and experimental values of $\delta 1$ and $\Delta 1$ are compared.

Thus, for large m, RC improves the fast loop of BM of a factor of $\Delta 1/\delta 1 = |\Sigma|$. The intuition behind such an improvement is the following. To compute the shift $\delta 1$ we look for the first occurrence of a given character in the pattern (starting form the end of the pattern). If the pattern is sufficiently large this happens, on the average, $|\Sigma|$ positions away from the end of the pattern. To compute the shift $\Delta 1$ we look for the first occurrence in the pattern of two given characters at a given distance each other and, if the pattern is sufficiently large this happens, on the average, $|\Sigma|^2$ positions away from the end of the pattern.

• The average shift per character comparison $\Delta 2/C$ done in the slow loop of RC is slightly lower than the average shift per character comparison $\delta 2/C$ done in the slow loop of BM. This is not a serious drawback, at least when the alphabet is not too small, since the percentage of comparisons done in the fast loops of BM and RC is the same and it is greater than the percentage of comparisons done in the slow loop (but for alphabets of very small size). It is also possible to use a mixed strategy that uses Δ_1 in the fast loop and δ_2 in the slow loop. This can slightly improve the average behaviour but makes $3n$ the worst-case bound.

• The ratio T/C_{ave}, the time spent per character comparison, is always close to $50\mu s$ for BM and to $60\mu s$ for RC, whatever the percentage of comparisons done in the fast loop is.

• The average times τ_1 and τ_2 spent for each comparison in the fast and in the slow loops, respectively, can be estimated as follows. Each row in Table 1 supplies a linear equation

$$\%\text{fast} \times \tau_1 + (1 - \%\text{fast}) \times \tau_2 = T/C_{ave}$$

for BM and a similar one for RC. Solving the two systems of equations (by the min square method, since they are overdetermined), we can estimate that BM takes $46\mu s$ per comparison in the fast loop and $40\mu s$ in the slow loop and that RC takes $60\mu s$ per comparison in the fast loop and $47\mu s$ in the slow loop. Since we used a personal computer with clock frequency of 7.8336 MHz it follows that for each comparison BM needs six clock periods in the fast loop and five in the slow loop and RC needs eight clock periods in the fast loop and six in the slow loop.

4. The Complexity Analysis

Let us start by showing that RC performs $2n$ character comparisons in the worst case when either the pattern does not occur in the text or the period π_w of the pattern is greater or equal to $m/2$ and that the worst case bound is $2n - m + h(m - 2\pi_w)$ if $m > 2\pi_w$ and the pattern occurs h times in the text.

Then we apply our method once more to RC, thus obtaining a new version of RC, where sequences of overlapping occurrences of the pattern in the text are handled separately by means of a new loop that is entered when the first occurrence is found.

Last we show that this makes $2n$ the worst case also for $m > 2\pi_w$. To this aim, we show that the number i of comparisons done by RC with the pattern aligned with the text in a nonmatching position is always less than or equal to twice the translation s of the pattern that is performed. Recall that the pattern is tested against the text in the following order: position $h_1 = m - 1$ first, then the positions $h_2, \ldots, h_d$ chosen using rule ii, and last the positions $h_{d+1}, \ldots, h_m$ chosen using rule iii. The bound is obviously satisfied when the translation of the pattern is performed during the fast loop since $i = 1 \leq s$ in such a case. It is also satisfied when the slow loop terminates on a h_i such that $2 \leq i \leq d$, since we have already observed that $s = \Delta_2(i) \geq i - 1$ in such a case. If the slow loop terminates

on a h_i such that $d + 1 \le i \le m$ then $s = r_{\min}(h_i) > h_i$ by definition and $h_i \ge i - d$ (since $h_{d+1}, \ldots, h_m$ are in increasing order). Thus $i - d < s$. Moreover, $s \ge \pi_w$ (since $s = r_{\min}(h_i)$ is a period of w) and $d \le \pi_w$. Thus $i < 2s$.

When a match is found, the number of character comparisons is m and the shift is π_w. Thus, when a match is found, we have an excess of $m - 2\pi_w$ comparisons only in the case of $m > 2\pi_w$. When $m > 2\pi_w$, the excess of $m - 2\pi_w$ comparisons can disappear if we add to RC a suitable loop to be entered when a match is found. Indeed, when a match is found and the shift of π_w is performed, we know that $w_{[0:m-\pi_w-1]} = t_{[b:b+m-\pi_w-1]}$ and this information is lost in RC.

Exploiting such information, we can test only the characters w_i of the pattern such that $m - \pi_w \le i \le m - 1$. Assume the characters in $w_{[m-\pi_w:m-1]}$ are tested from right to left. Then, for all i such that $m - \pi_w \le i \le m - 1$ we can define a new shift function $\Delta_3(i)$ that takes care of $w_{[0:m-\pi_w-1]} = t_{[b:b+m-\pi_w-1]}$ as follows:

$$\Delta_3(i) = \min\left\{k \mid k \notin \mathrm{Pos}(w_i, i) \cup \bigcup_{j=i+1}^{m-1} \mathrm{Neg}(w_j, j) \cup \bigcup_{j=0}^{m-\pi_w-1} \mathrm{Neg}(w_j, j)\right\}.$$

The set $\mathrm{Pos}(w_i, i) \cup \bigcup_{j=0}^{m-\pi_w-1} \mathrm{Neg}(w_j, j)$ contains all integers less than $m - \pi_w$ and so the shift function $\Delta_3(i)$ will be defined as follows. Let $k'_{\min}(h)$ be defined as the minimum k such that $k \ge m - \pi_w$ and $h_{\min}(k) = h \ge k$ if any such k exists, as $k'_{\min}(h) = 0$ otherwise. Then

$$\Delta_3(i) = \begin{cases} k'_{\min}(i) & \text{if } k'_{\min}(i) \ne 0; \\ r_{\min}(i) & \text{if } k'_{\min}(i) = 0. \end{cases}$$

Note that $\Delta_3(i)$ is always greater than $m - \pi_w$ for all i such that $m - \pi_w \le i \le m - 1$.

However, the use of $\Delta_3(i)$ when $i = m - 1$ does not assure that the assertion $s = m$ **or** $w_{m-s-1} = t_{b+m-s-1}$ holds after the shift is performed and this makes the algorithm incorrect.

Thus, when $i = m - 1$ we use the shift $\Delta_1(t_{b+m-1}, s)$ as in the fast loop. Note that, in such a case the previous shift s is equal to π_w. Then $t_{b+m-1} \ne w_{m-s-1} = w_{m-1}$ and so all k, such that $1 \le k \le m - \pi_w$, belongs to at least one of the two sets $\mathrm{Neg}(w_{m-s-1}, m - s - 1)$ and $\mathrm{Neg}(t_{b+m-1}, m - 1)$. Thus $\Delta_1(t_{b+m-1}, s)$ is also greater than $m - \pi_w$.

Last, if an occurrence of the pattern is found (and $i = m - \pi_w - 1$ in such a case) the shift to be performed is π_w.

The modified version of RC is

```
b := 0;
s := m;
repeat
  {AMF(b − 1) and (s = m or w_{m−s−1} = t_{b+m−s−1})}
  while w_{m−1} ≠ t_{b+m−1} do
    begin
      s := Δ1(t_{b+m−1}, s);
        {AMF(b − 1) and focc(t_{b+m−1}, s)}
        {AMF(b + s − 1) and (s = m or w_{m−s−1} = t_{b+m−1})}
      b := b + s;
        {AMF(b − 1) and (s = m or w_{m−s−1} = t_{b+m−s−1})}
      if b > n − m then {AMF(n − m)} exit
    end;
  i := 2;
  {AMF(b − 1) and ∀q ∈ [1 : i − 1](w_{h_q} = t_{b+h_q}) and i ≤ m + 1}
while i ≤ m and w_{h_i} = t_{b+h_i} do := i + 1;
  {AMF(b − 1) and ∀q ∈ [1 : i − 1](w_{h_q} = t_{b+h_q})
  and (i = m + 1 or w_{h_i} ≠ t_{b+h_i})}
if i = m + 1 then
  begin
    repeat
        {AMF(b − 1) and w_{[0:m−1]} = t_{[b:b+m−1]}}
      match found at b;
      s := π_w;
      b := b + s;
        {AMF(b − 1) and w_{[0:m−π_w−1]} = t_{[b:b+m−π_w−1]}}
      if b > n − m then {AMF(n − m)} exit
      i := m − 1;
      while i ≥ m − π_w and w_i = t_{b+i} do i := i − 1;
        {AMF(b − 1) and w_{[0:m−π_w−1]} = t_{[b:b+m−π_w−1]}
        and w_{[i+1:m−1]} = t_{[b+i+1:b+m−1]}
          and (i = m − π_w − 1 or w_i ≠ t_{b+i})}
    until i ≥ m − π_w;
        {AMF(b − 1) and w_{[0:m−π_w−1]} = t_{[b:b+m−π_w−1]}
        and w_{[i+1:m−1]} = t_{[b+i+1:b+m−1]}
      and w_i ≠ t_{b+i}}
    if i = m − 1 then s := Δ1(t_{b+m−1}, s) else s := Δ3(i);
    b := b + s
      {AMF(b − 1) and (s = m or w_{m−s−1} = t_{b+m−s−1})}
  end
else
```

```
    begin
      {AMF(b) and ∀q ∈ [1 : i − 1](w_{h_q} = t_{b+h_q}) and w_{h_i} ≠ t_{b+h_i})}
    s := Δ2(i);
      {AMF(b + Δ2(i) − 1) and (s = m or w_{m−s−1} = t_{b+m−1})}
    b := b + s
      {AMF(b − 1) and (s = m or w_{m−s−1} = t_{b+m−s−1})}
    end
  until b > n − m
    {AMF(n − m)}
```

When the new loop is entered we have an excess of at most $m - 2\pi_w$ comparisons with respect to the total shift performed. In case a mismatch occurs in the new position of the pattern, the number of comparisons being at most π_w and the shift being greater than $m - \pi_w$, we have a gain of at least $m - 2\pi_w$ shifted positions with respect to the number of comparisons performed, and this amortizes the excess of comparisons done in the previous step. In the case of a sequence of overlapping occurrences of the pattern in the text, the gain in the last step amortizes the initial excess while in the intermediate steps the shift $s = \pi_w$ is always equal to the number of comparisons. Thus, we may conclude that RC beats BM with respect to the worst-case behaviour.

5. The Preprocessing of the Pattern

As pointed out in [12] the preprocessing of the pattern for the algorithm BM, i.e., the computation of the shift functions δ_1 and δ_2, can be done in linear time. Now we show that the preprocessing of the pattern for RC can be computed in quadratic time in the worst case and in linear time on the average. The worst case quadratic time is due to the computation of the shift function Δ_1 while Δ_2 can be computed in linear time.

To compute Δ_1 we start by computing, for each character c in Σ, the index locc(c) of the last occurrence of c in $w_{[0:m-2]}$ (setting locc(c) $= -1$ in case c does not occur in $w_{[0:m-2]}$). Meanwhile we link downward all the occurrences of each character of the pattern setting link(i) $= -1$, when w_i is the first occurrence of a character, and setting link(-1) $= -1$ as a sentinel. This can be done in linear time as follows:

```
for c := firstchar to lastchar do locc(c) := −1;
link(−1) := −1;
for i := 0 to m − 2 do
  begin
    link(i) := locc(w_i);
    locc(w_i) := i
  end
```

After the execution of this algorithm the following properties hold:

$$c \notin w_{[\mathrm{locc}(c)+1:m-1]} \textbf{ and } \big(\mathrm{locc}(c) = -1 \textbf{ or } w_{\mathrm{locc}(c)} = c\big)$$

and

$$w_i \notin w_{[\mathrm{link}(i)+1:i-1]} \textbf{ and } \big(\mathrm{link}(i) = -1 \textbf{ or } w_{\mathrm{link}(i)} = w_i\big).$$

Recall that $\Delta_1(c, s) = \min(k \mid R(k))$, where $R(k)$ is the assertion

$$(k \geq m \textbf{ or } w_{m-k-1} = c) \textbf{ and } (k \geq m - s \textbf{ or } w_{m-k-s-1} = w_{m-s-1}).$$

Moreover, let $Q(i, j, c)$ be the assertion

$$\begin{aligned} Q(i,j,c) \equiv_{\mathrm{def}} & (i = -1 \textbf{ or } w_i = c) \textbf{ and } (j = -1 \textbf{ or } w_j = w_{m-s-1}) \\ & \textbf{and } \forall k((1 \leq k < m - i - 1) \Rightarrow \neg R(k)) \\ & \textbf{and } \forall k((m - i - 1 \leq k < m - j - s - 1) \\ & \qquad \Rightarrow w_{m-k-s-1} \neq w_{m-s-1}). \end{aligned}$$

Then, using $Q(i, j, c)$ as a loop invariant, Δ_1 can be computed as follows:

```
for c := firstchar to lastchar do
  for s := 1 to m do
    begin
      i := locc(c);
      j := link(m − 1 − s);
       {Q(i, j, c)}
      while i − j ≠ s and j ≥ 0 do
       {Q(i, j, c) and i − j ≠ s and j ≥ 0}
       if i − j > s then {Q(i, j, c) and i − j > s and j ≥ 0}
         i := link(i)
       else {Q(i, j, c) and i − j < s and j ≥ 0}
         j := link(j);
      {Q(i, j, c) and (i − j = s or j = −1)}
      while i − j > s do i := link(i);
       {Q(i, j, c) and (i − j = s or (i − j < s and j = −1))}
       {R(m − i − 1) and ∀k((1 ≤ k < m − i − 1) ⇒ ¬ R(k))}
      Δ1(c, s) := m − i − 1
    end
```

Note that, although the innermost while loop requires time $O(m)$ in the worst case, on the average it requires time $O(|\Sigma|)$ (because on the average

$\Delta_1(c, s) = O(|\Sigma|^2)$ and $i - \text{link}(i) = O(|\Sigma|)$). Thus Δ_1 is computed in time $O(m^2|\Sigma|)$ in the worst case and in time $O(m|\Sigma|^2)$ on the average.

However, in practical applications, when s is large ($s > |\Sigma|^2$) the shift function $\Delta_1(c, s)$ can be replaced by the function $\delta_1(c)$ still keeping the average shift of order $|\Sigma|^2$. Indeed, δ_1 is used in place of Δ_1 only when the previous shift s is large ($s > |\Sigma|^2$) and so the average between the two shifts is greater than $0.5|\Sigma|^2$. The computation of the shift function $\Delta_1(c, s)$ for $1 \le s \le |\Sigma|^2$ requires time $O(m|\Sigma|^3)$ in the worst case and in time $O(|\Sigma|^4)$ on the average.

Computation of Δ_2 can be done in linear time as follows. We first compute, for all k such that $1 \le k \le m$, the function $h_{\min}(k)$ that is defined as the minimum integer h such that $h \ge k - 1$ and $w_{[h-k+1:m-1]}$ is k-periodic. The computation of $h_{\min}(k)$ can be done in linear time by the following algorithm:

```
w_{-1} := #; {set a sentinel at the beginning of the pattern}
k := 1;
i := m - 1;
repeat
  while w_{i-k} = w_i do i := i - 1;
  h_min(k) := i;
  q := k + 1;
  while h_min(q - k) - (q - k) > i do
    begin
      h_min(q) := h_min(q - k);
      q := q + 1
    end;
  i := i + (q - k);
  k := q
  if i = m then i := m - 1
until k > m
```

This algorithm is essentially the algorithm we used in [5] to compute the function $h_{\max}(k)$, except that the pattern is scanned in the reverse order. Thus we refer to [5] for the correctness proof and for the exposition of the algorithm design.

Then we compute, for all i such that $0 \le i \le m - 1$, the function $k_{\min}(i)$ by using the algorithm:

```
for i := 0 to m - 1 do k_min(i) := 0;
for k := m downto 1 do
  if h_min(k) ≥ k then k_min(h_min(k)) := k
```

The next step is the computation for all i such that $0 \leq i \leq m - 1$ of the function $r_{\min}(i)$, by using the algorithm:

```
for i := m - 1 downto 0 do
  begin
    if h_min(i + 1) = i then r := i + 1;
    r_min(i) := r
  end
```

Last, to choose the order $h_1, \ldots, h_m$ and to compute the corresponding shift function Δ_2, we use the algorithm:

```
h_1 := m - 1;
i := 2;
for k := 1 to m do
  if h_min(k) ≠ m - 1 and k_min(h_min(k)) = k then
    begin
      h_i := h_min(k);
      Δ_2(i) := k;
      i := i + 1
    end;
i := m;
for j := m - 2 downto 0 do
  if k_min(j) = 0 then
    begin
      h_i := j;
      Δ_2(i) := r_min(j);
      i := i - 1
    end;
h_{m+1} := -1;
Δ_2(m + 1) := r_min(0)
```

All those algorithms are linear in m and so Δ_2 is computed in linear time.

6. Notes about the Algorithm Design Method

Both algorithm C in [5] and algorithm RC in this paper have been obtained using the same design method. The same method has been used in [5] to obtain the algorithm to compute $h_{\max}(k)$ and, in the present paper, to obtain the algorithm to compute $h_{\min}(k)$ (which is essentially the algorithm for $h_{\max}(k)$ in [5]) and to obtain the algorithm to compute $\Delta_1(c, s)$. Here we sketch a trace of this method.

Standard top-down design of algorithms, as exposed in [1], mainly consists in decomposing the overall problem into subproblems using cor-

rectness rules both as decomposition schemes to guide the algorithm design and as proof rules to be sure that the decomposition is correct. In particular, iterative decomposition of a problem can be done using, as a decomposition scheme, the proof rule of the while statement,

$$\frac{P \Rightarrow R, \{R \textbf{ and } B\}S\{R\}, R \textbf{ and } \neg B \Rightarrow Q}{\{P\} \textbf{ while } B \textbf{ do } S\{Q\}}$$

or, taking account that an initialization statement is often needed to start a loop, the compound proof rule,

$$\frac{\{P\}T\{R\}, \{R \textbf{ and } B\}S\{R\}, R \textbf{ and } \neg B \Rightarrow Q}{\{P\} \textbf{ begin } T; \textbf{ while } B \textbf{ do } S \textbf{ end } \{Q\}},$$

where T is the initialization statement and R is a suitable "loop invariant."

Usually a naive (and inefficient) iterative solution of the problem is easy to find using either of those rules. Thus we are faced with the problem of how to improve the algorithm efficiency.

The statement S is often the sequential composition of two statements: a statement S_1 that produces some information (making tests and computing values) and a statement S_2 that consumes the information (using it to update the values of some program variables). In the simplest cases, either S_1 or S_2 may be empty. Thus, the top-down design of the naive algorithm supplies, besides the loop invariant R, also another assertion V that asserts the correctness of the information computed by S_1 and used in S_2 to update the values of the program variables. Let

begin
  $\{P\}\ T\ \{R\}$;
  while B **do**
    begin
      $\{R$ **and** $B\}\ S_1\ \{V\}$;
      $\{V\}\ S_2\ \{R\}$;
    end $\{R$ **and** $\neg B\}$
end$\{Q\}$

be the proof of the naive algorithm. Being the algorithm naive, the assertion V reflects the naive idea we used to characterize the computation of a single step of the iterative algorithm and the correctness of $\{R$ **and** $B\}S_1\{V\}$ and $\{V\}S_2\{R\}$ is straightforward. However, the algorithm being correct, the assertion V is somewhat of intermediate between the strongest postcondition $V_1 = \text{sp}(S_1, R \textbf{ and } B)$ of the statement S_1 with respect to the precondition $\{R$ **and** $B\}$ and the weakest precondition $V_2 = \text{wp}(S_2, R)$ of the statement S_2 with respect to the postcondition $\{R\}$. The assertions V_1 and V_2 can be computed using the calculus of the strongest postcondition and the calculus of the weakest precondition as

given in [7]. Such a computation is quite tedious and mechanical. However, the computation being mechanical prevents our making all those simplifications that we have used (either consciously or unconsciously) in the design and proof of the naive algorithm. Thus, we obtain an assertion V_1 that characterizes the "whole" information computed by S_1 and an assertion V_2 that characterizes the "only" information used by S_2. The proof of the naive algorithm becomes

begin
  $\{P\}\ T\ \{R\}$;
  while B **do**
    begin
      $\{R$ **and** $B\}\ S_1\ \{V_1\}$;
      $\{V_2\}\ S_2\ \{R\}$
    end $\{R$ **and** $\neg B\}$
end$\{Q\}$

where $V_1 \Rightarrow V_2$ or, equivalently, between $\{V_1\}$ and $\{V_2\}$ there is a null statement.

The effect of the null statement is a loss of information and, often, such a loss of information represents a wasting of the computational effort done to compute it. There are two ways to avoid such a wasting of computational effort. We may reconsider the design of the statement S_1, looking for a statement S_1' that, starting from the precondition R **and** B computes only the information that is strictly needed for S_2, i.e., a statement S_1' such that $\mathrm{sp}(S_1', R$ **and** $B)$ is as closed as possible to $\{V_2\}$. This is the strategy we used to improve the slow loop of BM.

On the other hand, we may try to use the lost information to speed up the subsequent computations. This can be done as follows. We compute $\mathrm{sp}(S_2, V_1)$, thus obtaining a new invariant assertion R' that is stronger than R. Then we look for a statement S_1' that computes the same information V_1 computed by S_1, but starting from the precondition R' **and** B (that carries over all the information computed in the previous iteration). This is the strategy we used to improve the fast loop of BM.

Appendix A

The derivation of formulas (3), (4), and (5)

Formula (3). The value of $\min\{k \mid k \notin \mathrm{Neg}(c, i)\}$ is $k \leq i$ when $c = w_{i-k}$ and $c \neq w_{i-h}$ for all h such that $1 \leq h < k$ and it is $k = i + 1$ when $c \neq w_{i-h}$ for all h such that $1 \leq h \leq i$.

Let $p = 1/|\Sigma|$ be the probability that two characters chosen at random in $|\Sigma|$ are equal and $q = 1 - p = (|\Sigma| - 1)/|\Sigma|$ be the probability that

they are different. Then $\min\{k \mid k \notin \mathrm{Neg}(c, i)\}$ is equal to k with probability

$$Q(k) = \begin{cases} pq^{k-1} & \text{if } k \le i \\ q^i & \text{if } k = i + 1. \end{cases}$$

Thus, the average $\tilde{N}(i)$ of $\min\{k \mid k \notin \mathrm{Neg}(c, i)\}$ is

$$\begin{aligned} \tilde{N}(i) &= \sum_{k=1}^{i} kpq^{k-1} + (i+1)q^i \\ &= p \sum_{k=0}^{i-1} (k+1)q^k + p(i+1)q^i - p(i+1)q^i + (i+1)q^i \\ &= p \sum_{k=0}^{i} (k+1)q^k + (1-p)(i+1)q^i \\ &= p \sum_{k=0}^{i} q^k + p \sum_{k=0}^{i} kq^k + (i+1)q^{i+1} \end{aligned}$$

that, using the summation formulas

$$\sum_{k=0}^{i} x^k = \frac{1 - x^{i+1}}{1 - x}, \qquad \sum_{k=0}^{i} kx^k = \frac{ix^{i+2} - (i+1)x^{i+1} + x}{(1-x)^2},$$

gives

$$\begin{aligned} \tilde{N}(i) &= p\frac{1 - q^{i+1}}{1 - q} + p\frac{iq^{i+2} - (i+1)q^{i+1} + q}{(1-q)^2} + (i+1)q^{i+1} \\ &= 1 - q^{i+1} + \frac{iq^{i+2} - (i+1)q^{i+1} + q}{p} + (i+1)q^{i+1} \\ &= 1 - q^{i+1} + (i+1)q^{i+1} + \frac{iq^{i+2} - iq^{i+1} - q^{i+1} + q}{p} \\ &= 1 + iq^{i+1} + \frac{-iq^{i+1}(1-q) - q^{i+1} + q}{p} \\ &= 1 + \frac{-q^{i+1} + q}{p} \\ &= \frac{p - q^{i+1} + q}{p} \\ &= \frac{1 - q^{i+1}}{p} = \sum_{j=0}^{i} \left(\frac{|\Sigma| - 1}{|\Sigma|}\right)^j = |\Sigma|\left[1 - \left(\frac{|\Sigma| - 1}{|\Sigma|}\right)^{i+1}\right]. \end{aligned}$$

Formula (4). The value of $\min\{k|k \notin \mathrm{Pos}(c,i)\}$ is $k \leq i$ when $c \neq w_{i-k}$ and $c = w_{i-h}$ for all h such that $1 \leq h < k$ and it is $k = i+1$ when $c = w_{i-h}$ for all h such that $1 \leq h \leq i$. Then $\min\{k|k \notin \mathrm{Pos}(c,i)\}$ is equal to k with probability

$$P(k) = \begin{cases} qp^{k-1} & \text{if } k \leq i \\ p^i & \text{if } k = i+1. \end{cases}$$

Thus, the average $\tilde{P}(i)$ of $\min\{k|k \notin \mathrm{Pos}(c,i)\}$ is

$$\tilde{P}(i) = \sum_{k=1}^{i} kqp^{k-1} + (i+1)p^i$$

that is equal to $\tilde{N}(i)$ with p and q interchanged. Thus

$$\tilde{P}(i) = \frac{1-p^{i+1}}{q} = \sum_{j=0}^{i} \frac{1}{|\Sigma|^j} = \frac{|\Sigma|-1}{|\Sigma|}\left(1 - \frac{1}{|\Sigma|^{i+1}}\right).$$

Formula (5). The inequality $\tilde{N}(i) \leq i+1$ follows from $\tilde{N}(i) = \sum_{j=0}^{i}((|\Sigma|-1)/|\Sigma|)^j$ since all the addenda are less than or equal to one. The inequality $\tilde{N}(i) \leq |\Sigma|$ follows from $\tilde{N}(i) = |\Sigma|[1 - ((|\Sigma|-1)/|\Sigma|)^{i+1}]$ since $1 - ((|\Sigma|-1)/|\Sigma|)^{i+1}$ is less than one.

The inequality $\tilde{P}(i) \geq 1$ follows from $\tilde{P}(i) = \sum_{j=0}^{i}(1/|\Sigma|^j)$ since the first addendum is equal to one. The inequality $\tilde{P}(i) \leq 2$ follows from $\tilde{P}(i) = (|\Sigma|/(|\Sigma|-1))(1 - (1/|\Sigma|^{i+1}))$ since $|\Sigma|/(|\Sigma|-1)$ is less than two (recall that $|\Sigma| \geq 2$) and $1 - 1/|\Sigma|^{i+1}$ is less than one.

To prove the inequality $(1 - 1/e)\min(i+1, |\Sigma|) \leq \tilde{N}(i)$ we proceed as follows. Since

$$\tilde{N}(0) = 1 \qquad \text{and} \qquad \lim_{i \to \infty} \tilde{N}(i) = |\Sigma|$$

and the function $\tilde{N}(i)$ is convex, then the minimum of the ratio

$$\rho = \frac{\tilde{N}(i)}{\min(i+1, |\Sigma|)}$$

is for $i + 1 = |\Sigma|$. Thus

$$\rho \geq \frac{\tilde{N}(|\Sigma|-1)}{|\Sigma|} = 1 - \left(\frac{|\Sigma|-1}{|\Sigma|}\right)^{|\Sigma|}.$$

Since the function $1 - ((x-1)/x)^x$ is decreasing in the interval $[1, \infty]$ and

$\lim_{x \to \infty} 1 - ((x-1)/x)^x = 1 - 1/e$, then we conclude that

$$\rho \geq 1 - 1/e.$$

Appendix B: Computation of the Average of $\Delta 1$

The average shift $\tilde{\Delta}1$ can be expressed as

$$\tilde{\Delta}1 = \sum_{s=1}^{m} sQ(s),$$

where $Q(s)$ is the probability distribution of the shifts performed in the fast loop.

To compute an (approximate) value of the probability distribution $Q(s)$ we proceed as follows. For all $s, s' = 1, \ldots, m$, the probability that $\Delta_1(c, s') = s$ is the product of the probability $p(s, s')$ that the assertion

$$(s = m \textbf{ or } w_{m-s-1} = c) \textbf{ and } (s \geq m - s' \textbf{ or } w_{m-s-s'-1} = w_{m-s'-1})$$

holds, times the product of the probabilities $1 - p(j, s')$ that the assertions

$$(j = m \textbf{ or } w_{m-j-1} = c) \textbf{ and } (j \geq m - s' \textbf{ or } w_{m-j-s'-1} = w_{m-s'-1})$$

do not hold for $j = 1, \ldots, s-1$. The value of $p(s, s')$ is

$$p(s, s') = \begin{cases} 1 & \text{if } s = m \\ p & \text{if } m - s' \leq s < m \\ p^2 & \text{if } 1 \leq s \leq m - s' - 1. \end{cases}$$

Thus the probability $P(s, s')$ that $\Delta_1(c, s') = s$ can be expressed as follows in terms of p, $q = 1 - p$ and $r = 1 - p^2$:

$$P(s, s') = p(s, s') \prod_{j=1}^{s-1} (1 - p(j, s'))$$

$$= \begin{cases} q^{m-1} & \text{if } s = m \text{ and } s' = m \\ pq^{s-1} & \text{if } 1 \leq s < m \text{ and } s' = m \\ r^{m-s'-1}q^{s'} & \text{if } s = m \text{ and } 1 \leq s' < m \\ pr^{m-s'-1}q^{s+s'-m} & \text{if } m - s' \leq s < m \text{ and } 1 \leq s' < m \\ p^2 r^{s-1} & \text{if } 1 \leq s \leq m - s' - 1 \text{ and } 1 \leq s' < m. \end{cases}$$

Let $Q_0(s)$ be the probability distribution of the last shift performed before the fast loop is entered and let us suppose that k consecutive shifts are performed in the fast loop. Then the probability distribution $Q_1(s), Q_2(s), \ldots, Q_k(s)$, of the shifts performed in the fast loop can be computed using the equation

$$Q_t(s) = \sum_{s'=1}^{m} P(s, s')Q_{t-1}(s').$$

The sequence $\langle Q_1(s), Q_2(s), \ldots, Q_k(s)\rangle$ and the corresponding sequence $\langle \tilde{\Delta}1_1, \tilde{\Delta}1_2, \ldots, \tilde{\Delta}1_k\rangle$ has been computed, by means of an iterative procedure, for all the values of m and $|\Sigma|$ that we used in Table 1, starting from several initial distributions $Q_0(s)$.

Both the sequences converge very quickly to a limit that is independent from $Q_0(s)$. Moreover, the sequence $\langle \tilde{\Delta}1_1, \tilde{\Delta}1_2, \ldots, \tilde{\Delta}1_k\rangle$ oscillates around the limit with monotonically decreasing amplitude. Thus the limit $\tilde{\Delta}1$ is close to the average shift also for short sequences of consecutive iterations of the fast loop. In Table 2, the limit $\tilde{\Delta}1$ of the sequence $\langle \tilde{\Delta}1_1, \tilde{\Delta}1_2, \ldots, \tilde{\Delta}1_k\rangle$ is compared with the experimental value $\Delta 1$ of the average shift done in the fast loop.

When $m \to \infty$ the matrix $P(s, s')$ becomes the infinite idempotent matrix $P^{\infty}(s, s') = p^2 r^{s-1}$ (whose entries do not depend on s'). Thus

$$\lim_{m\to\infty} \tilde{\Delta}1 = \sum_{s=1}^{\infty} sp^2 r^{s-1}$$

By using the limits of the summation formulas in Appendix A,

$$\sum_{k=0}^{\infty} x^k = \frac{1}{1-x}, \qquad \sum_{k=0}^{\infty} kx^k = \frac{x}{(1-x)^2},$$

we obtain

$$\lim_{m\to\infty} \tilde{\Delta}1 = \frac{1}{p^2} = |\Sigma|^2.$$

REFERENCES

1. S. ALAGIĆ AND M. A. ARBIB, "The Design of Well-Structured and Correct Programs," Springer-Verlag, New York, 1982.
2. A. APOSTOLICO AND R. GIANCARLO, The Boyer–Moore–Galil string searching strategies revisited, *SIAM J. Comput.* **15** (1986), 98–105.

3. R. S. BOYER AND J. S. MOORE, A fast string searching algorithm, *Comm. ACM* **20** (1977), 762–772.
4. R. COLE, Tight bounds on the complexity of the Boyer–Moore pattern matching algorithm, manuscript, 1990.
5. L. COLUSSI, Correctness and efficiency of the pattern matching algorithms, *Inform. and Comput.* **95** (1991), 225–251.
6. L. COLUSSI, Z. GALIL, AND R. GIANCARLO, On the exact complexity of string matching, *in* "Proceedings, 31st IEEE Symposium on Foundations of Computer Science, 1990," pp. 135–143.
7. J. W. DEBAKKER, "Mathematical Theory of Program Correctness," Prentice-Hall, Englewood Cliffs, NJ.
8. Z. GALIL AND R. GIANCARLO, On the exact complexity of string matching (upper bounds), *SIAM J. Comput.* **21** (1992), 407–437.
9. Z. GALIL AND R. GIANCARLO, On the exact complexity of string matching (lower bounds), *SIAM J. Comput.* **20** (1991), 1008–1020.
10. L. J. GUIBAS AND A. M. ODLYZKO, A new proof of the linearity of the Boyer–Moore string matching algorithm, *SIAM J. Comput.* **9** (1980), 672–682.
11. C. A. R. HOARE AND N. WIRTH, An axiomatic definition of the programming language PASCAL, *Acta Inform.* **6**, No. 4 (1973), 335–355.
12. D. E. KNUTH, J. H. MORRIS, AND V. B. PRATT, Fast pattern matching in strings, *SIAM J. Comput.* **6** (1977), 189–195.