ELSEVIER

Information Processing Letters 71 (1999) 107–113

Information Processing Letters

www.elsevier.com/locate/ipl

Fast practical multi-pattern matching

Maxime Crochemore [a,1], A. Czumaj [b], L. Gąsieniec [c], T. Lecroq [d,*], W. Plandowski [e,2], W. Rytter [c,e,2]

[a] *Institut Gaspard-Monge, Université de Marne-la-Vallée, 77454 Marne-la-Vallée Cedex 2, France*
[b] *Department of Mathematics and Computer Science, University of Paderborn, Fürstenallee 11, 33102 Paderborn, Germany*
[c] *Department of Computer Science, The University of Liverpool, Chadwick Building, Peach Street, Liverpool L69 7ZF, UK*
[d] *Laboratoire d'Informatique Fondamentale et Appliquée de Rouen, Atelier Biologie Informatique Statistique Sociolinguistique, Faculté des Sciences et des Techniques, Université de Rouen, 76821 Mont-Saint-Aignan Cedex, France*
[e] *Institute of Informatics, Warsaw University, ul. Banacha 2, 00-913 Warsaw 59, Poland*

Received 5 October 1998; received in revised form 6 July 1999
Communicated by L. Boasson

Abstract

The multi-pattern matching problem consists in finding all occurrences of the patterns from a finite set X in a given text T of length n. We present a new and simple algorithm combining the ideas of the Aho–Corasick algorithm and the directed acyclic word graphs. The algorithm has time complexity which is linear in the worst case (it makes at most $2n$ symbol comparisons) and has good average-case time complexity assuming the shortest pattern is sufficiently long. Denote the length of the shortest pattern by m, and the total length of all patterns by M. Assume that M is polynomial with respect to m, the alphabet contains at least 2 symbols and the text (in which the pattern is to be found) is random, for each position each letter occurs independently with the same probability. Then the average number of comparisons is $O((n/m) \cdot \log m)$, which matches the lower bound of the problem. For sufficiently large values of m the algorithm has a good behavior in practice.

Keywords: Formal languages; Combinatorial problems

1. Introduction

We consider the multiple string matching problem: finding all occurrences of a finite set X of string patterns in a text $T = T[1..n]$ of length n (the positions in T are starting with 1). Let σ be the size of the alphabet, assume σ is a constant. Finding all occurrences of elements of X is a problem that appears in bibliographic search and in information retrieval. The first algorithm to solve this problem in $O(n)$ time was the Aho–Corasick (AC algorithm for short) [2], which can be viewed as a generalization of the Knuth–Morris–Pratt algorithm [12], designed for a single pattern. As for one pattern the Boyer–Moore algorithm [5] has a better behavior in practice, Commentz-Walter developed an algorithm combining the ideas of the Aho–Corasick and Boyer–Moore algorithms [6,7]. A complete version can be found in [1]. Later, Uratani [15], and Baeza-Yates and Régnier [3] developed similar algorithms. The

* Corresponding author. Email: Thierry.Lecroq@dir.univ-rouen.fr. Supported in part by programme "Génomes" of CNRS.
[1] Supported in part by programme "Génomes" of CNRS.
[2] Supported by the grant KBN 8T11C03915.

0020-0190/99/$ – see front matter
PII: S0020-0190(99)00092-7

reader is referred to the books of Crochemore and Rytter [9] and Gusfield [10] for explanations of these techniques. We consider a new algorithm using the ideas of the Aho–Corasick algorithm [2] and the Reverse Factor algorithm [8] for searching for one pattern. In the implementation of the algorithm two automata-theoretic tools are employed: an Aho–Corasick machine and a Directed Acyclic Word Graph (DAWG, in short).

The lower bound for the average number of comparisons needed to solve the problem was shown in [16].

Lemma 1.1 (Lower bound). *For alphabets of size at least* 2 *the average number of symbol comparisons for the pattern-matching algorithm is* $\Omega((n/m)\log m)$.

Our algorithm achieves the lower bound if the total size of all patterns is polynomial with respect to m, the length of the shortest pattern. Due to Lemma 1.1 this is optimal.

We denote by $Fact(X)$, $Pref(X)$, the set of all the subwords, prefixes, of the words of X, respectively. For a string γ, positions $j \leqslant i$ and a symbol a define:

$$NEXT1(\gamma, a) = \text{the longest suffix of } \gamma \cdot a \text{ which is in } Pref(X),$$

$$NEXT2(j, i) = \text{the longest suffix of } T[j..i] \text{ which is in } Pref(X),$$

$$\gamma(i) = \text{the longest suffix of } T[1..i] \text{ which is in } Pref(X),$$

$$\Delta(i) = \max\{|u|\colon u \in Fact(X) \text{ and } u \text{ is a suffix of } T[1..i]\}.$$

2. The algorithm MULTI-STRING-MATCHING

Our algorithm is based on a cooperation of two processes, which we call PROCESS1 and PROCESS2. The first process scans the text from left-to-right, then shifts a potential ending position of the pattern at least by $m/2$. For each position i PROCESS1 remembers $\gamma = \gamma(i)$. If $|\gamma| \leqslant m/2$ then PROCESS1 passes control to PROCESS2 which does a "backward" search starting from the position $i + SHIFT$, where $SHIFT = m - |\gamma|$. This jump is sufficiently large so that with high probability PROCESS2 ends quickly. The algorithm MULTI-STRING-MATCHING presented below reports for each position i the longest pattern from X whose occurrence ends at i, if there is any.

Algorithm MULTI-STRING-MATCHING(X, T);
Comment: searching a pattern from X in the text T
$i := 0$; $\gamma :=$ *empty_word*; *Invariant*: $\gamma = \gamma(i)$

REPEAT until stopped

PROCESS1:
  while $|\gamma| \geqslant m/2$ **do**
    if $\gamma \in X$ **then** REPORT A MATCH;
    $i := i + 1$; **if** $i > n$ **then STOP**;
    $\gamma := NEXT1(\gamma, T[i])$;

PROCESS2:
    $crit\_pos := i - |\gamma| + 1$;
    $SHIFT := m - |\gamma|$;
    $i := i + SHIFT$; **if** $i > n$ **then STOP**;
    $\gamma := NEXT2(crit\_pos, i)$;

There are several crucial facts related to the implementation of $NEXT1$ and $NEXT2$ and to the efficiency of the processes:

(P1) PROCESS2 makes at most $\min\{\Delta(i), i - crit\_pos\}$ symbol comparisons in one stage and its total complexity is of the same order as the number of comparisons.

(P2) If PROCESS1 starts with $i = i'$ and ends with $i = i''$ in a given iteration then it makes at most $i'' - i'$ comparisons in this iteration.

(P3) On average PROCESS2 and PROCESS1 scan in one stage only a logarithmic number of positions.

(P4) There are $O(n/m)$ iterations of the algorithm MULTI-STRING-MATCHING.

Observation. The factor $m/2$ is a compromise between two processes, in this way each of them on average contributes half of the total work.

Fig. 1 shows several iterations of PROCESS1, and how the control is passing to PROCESS2, the final value of γ in PROCESS1 is the starting value of γ in PROCESS2. The string γ satisfies at the beginning of each process the invariant $\gamma = \gamma(i)$, where i is the current position. Fig. 2 shows how the control is passing from PROCESS2 to PROCESS1.

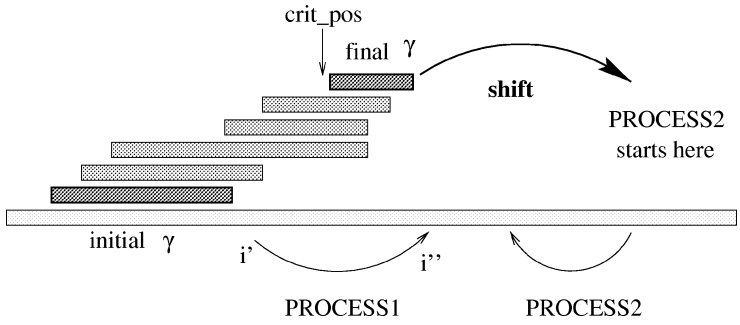

Fig. 1. PROCESS1 starts at position i' and stops at i'', when the shift becomes large, it is passing control to PROCESS2 with critical position equal to $i'' - |\gamma| + 1$ and $i = i'' + SHIFT$.

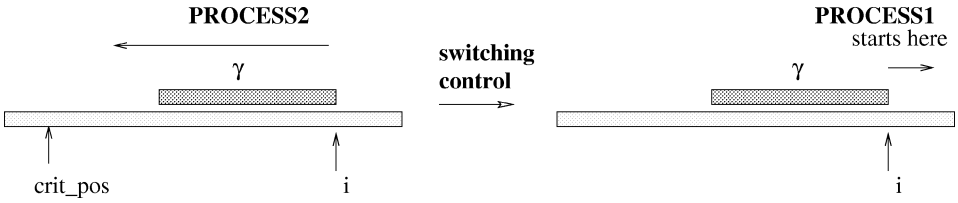

Fig. 2. PROCESS2 is passing control to PROCESS1.

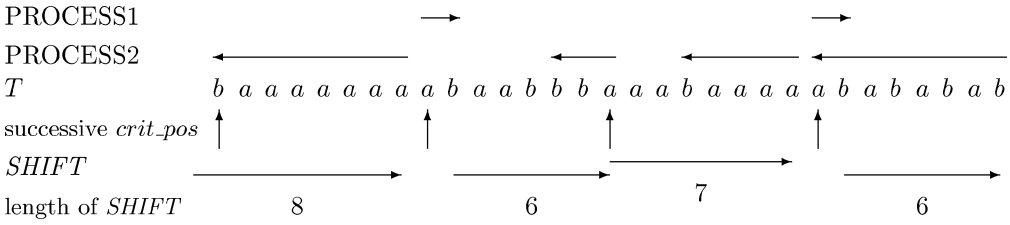

Fig. 3. The schematic history of the algorithm for the patterns $X = \{aaaaaaaa,\ abababab\}$. The positions 11, 12, 13, 17 and 18 are not inspected at all. PROCESS1 reports one occurrence of *aaaaaaaa* and one of *abababab*.

3. Complexity of the algorithm MULTI-STRING-MATCHING

We analyze worst-case and average-case complexities of the algorithm of Section 2. The good worst-case performance follows from properties (P1) and (P2), and average-case performance is good due to (P3). On average most positions are not scanned at all, see Fig. 3. We start with the (much simpler) worst-case analysis which reduces to the proof that the same position can be scanned at most twice.

Observation. It is important (for the worst-case complexity) that PROCESS2 does not need to scan to the left of the position *crit_pos*. The reason is that the third situation in Fig. 4 is impossible. We have:

$$\text{if } \left|\gamma(i)\right| \leqslant m/2 \quad \text{then } \left|\gamma(i+m-|\gamma(i)|)\right| \leqslant m.$$

Theorem 3.1. *The algorithm* MULTI-STRING-MATCHING *makes in total at most* $2n$ *symbol comparisons.*

Proof. Observe that a symbol at a given position of T can be inspected only once by PROCESS1. If it

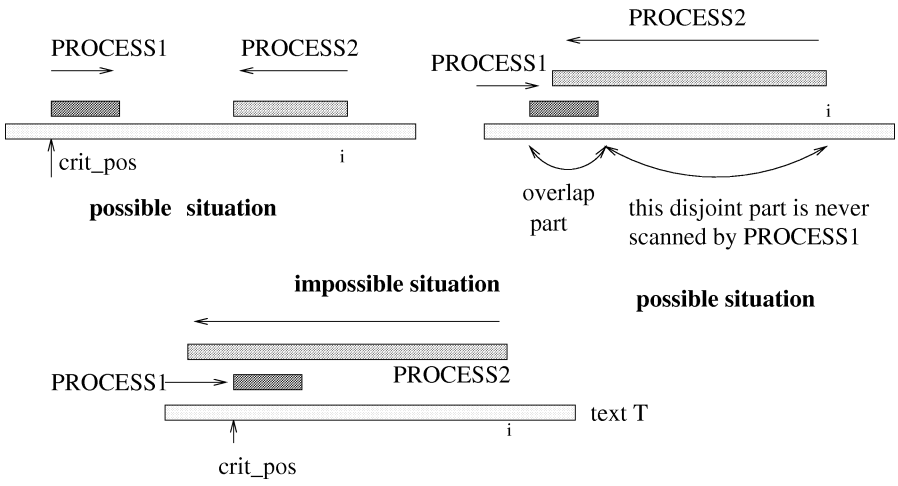

Fig. 4. Possible and impossible relations between the segments γ produced by consecutive instances of PROCESS1 and PROCESS2.

is inspected by PROCESS1 and later by PROCESS2 then it will be never inspected again. Also it can be inspected at most twice by PROCESS2, but in this case it is never inspected by PROCESS1. Possible situations are shown in Fig. 3, which shows that there can be an overlap of the segments γ in consecutive instances of PROCESS1 and PROCESS2, but the disjoint part is never later scanned by PROCESS1. Hence, the algorithm MULTI-STRING-MATCHING performs at most $2n$ inspections of text characters. □

The text T is random in the sense that each letter occurs at a given position independently with the same probability $1/\sigma$, where $\sigma \geqslant 2$ is the size of the alphabet.

Lemma 3.2.

$$\textit{Probability}\{\Delta(i) > (3k+1)\cdot \log_\sigma m\} \leqslant \frac{1}{m^{k+1}}.$$

Proof. For $C = 3k+1$ we have $|\textit{Fact}(X)| \leqslant m^{2k} = A$, on the other hand there are $B = \sigma^{C\cdot\log_\sigma m}$ different words of length $C\cdot\log_\sigma m$. The considered probability is at most $A/B = 1/m^{k+1}$. □

Theorem 3.3. *Assume $M \leqslant m^k$. The algorithm* MULTI-STRING-MATCHING *makes on average* $O(n\log_\sigma m/m)$ *inspections of text characters.*

Proof. There are no more than $O(n/m)$ shifts in the algorithm. Hence it is enough to show (P3). Denote $K = (3k+1)\cdot\log_\sigma m$. PROCESS2 ends with high probability after K steps, if not it makes at most m^k steps (the upper bound on the size of a longest pattern) with probability at most $1/m^{k+1}$. Hence average work is $O(K)$ which is logarithmic.

A similar argument works for PROCESS2. If $\Delta(i+K) < K$ then it scans (from position i to a position preceding $i+K$) at most K symbols. The probability of the opposite inequality is small and on average PROCESS2 ends after $O(K)$ steps. This completes the proof. □

4. Experimental results

In order to verify the good practical behavior of the MULTI-STRING-MATCHING algorithm we have tested it against the Commentz-Walter algorithm. The two algorithms were implemented in C. We implemented the simple Commentz-Walter algorithm which is quadratic in the worst case (see [11]). Tests on these two algorithms have been performed with three kinds of alphabet: binary alphabet, alphabet of size 4, and alphabet of size 8.

For each alphabet size, we randomly build a text of 50,000 characters. Then, we first made experiments with sets of patterns of the same length: for each length

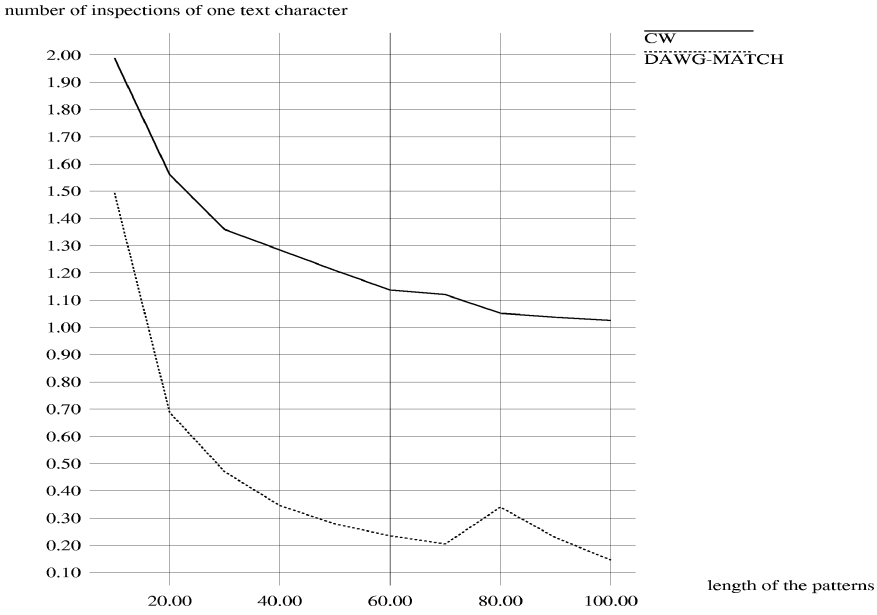

Fig. 5. Results for an alphabet of size 4.

Table 1
Results for an alphabet of size 4

Length of patterns	Commentz-Walter	MULTI-STRING-MATCHING
10	1.9887	1.4938
20	1.5612	0.6884
30	1.3593	0.47
40	1.2843	0.3457
50	1.2094	0.2785
60	1.1371	0.2351
70	1.1205	0.205
80	1.0521	0.3402
90	1.0376	0.2285
100	1.0258	0.1462
10–50	1.83	1.34
50–100	1.2	0.27

of pattern we randomly build a set of 100 patterns of the same length.

After that we build sets of patterns of different length (the length is random in a interval): one set with 100 patterns of lengths between 10 and 50, and one set of 100 patterns of lengths between 50 and 100. Then, for the two algorithms, we count the number of inspections per one text character. The results for an alphabet of size 4 are presented in Fig. 5 and in Table 1.

From these results it appears that for small alphabet and sufficiently long shortest pattern (with respect to M) the MULTI-STRING-MATCHING algorithm is better than the simple Commentz-Walter algorithm. This is due to the fact that the Commentz-Walter algorithm computes its shifts with the suffixes it recognizes in the text, but when the set of patterns is big the probability that those suffixes reappear close to the right end of at least one pattern is very large; so, the shifts computed by the Commentz-Walter are small.

5. The preprocessing phase

Our aim in this paper is to produce an efficient searching phase algorithm. For completeness we sketch shortly how the preprocessing phase can be done.

Lemma 5.1. *Assume the alphabet is of a constant size. We can preprocess a set of patterns in* $O(M)$ *time in such way that*
(1) $NEXT1(\gamma, a)$ *can be computed in* $O(1)$ *time,*
(2) $\gamma = NEXT2(j, i)$ *can be computed in* $O(\min\{\Delta(i), i - crit\_pos\})$ *time.*
(3) *The properties* (P1) *and* (P2) *are satisfied.*

Proof. We precompute the Directed Acyclic Word Graph (DAWG) (see [4]) of the reverse of the words of X. Then each query $NEXT2$ can be answered in the required time by scanning the text $T[j..i]$ right-to-left and simultaneously "walking" on the DAWG. We refer the reader to [9].

In order to process fast queries of the form $NEXT1$ we build an Aho–Corasick machine (see [2]). Then γ is represented by a state of the Aho–Corasick machine and $NEXT1(\gamma, a)$ corresponds directly to one step of this machine. Obviously each such step can be simulated in O(1) time for a constant sized alphabet. This completes the proof. □

When there is a large disparity of lengths of patterns it is possibly better to take more complicated *SHIFT*. Define:

$$STRONGER\_SHIFT(\gamma) = \min\{|u|;\ \gamma \cdot u \in X\}.$$

Observe that

$$m/2 \leqslant SHIFT \leqslant STRONGER\_SHIFT(\gamma) \leqslant m$$

if $SHIFT = m - |\gamma|$ and $|\gamma| \leqslant m/2$. Hence there is no dramatic difference between *SHIFT* and *STRONGER_SHIFT*.

Remark. The table *STRONGER_SHIFT* can be precomputed in linear time (with respect to M) using the Aho–Corasick machine "enriched" with the failure table.

The algorithm can be modified as follows.

Algorithm
MODIFIED MULTI-STRING-MATCHING(X, T);
Comment: searching a pattern from X in the text T
$i := 0$; $\gamma :=$ *empty_word*; *Invariant*: $\gamma = \gamma(i)$

REPEAT until stopped

PROCESS1:
while $|\gamma| \geqslant m/2$ **do**
if $\gamma \in X$ **then** REPORT A MATCH;
$i := i + 1$; **if** $i > n$ **then STOP**;
$\gamma := NEXT1(\gamma, T[i])$;

PROCESS2:
$crit\_pos := i - |\gamma| + 1$;
$i := i + STRONGER\_SHIFT(\gamma)$;
if $i > n$ **then STOP**;
$\gamma := NEXT2(crit\_pos, i)$;

However it does not improve asymptotic complexity and complicates the algorithm. The $2n$ bound on the number of comparisons and good asymptotic behavior of the modified algorithm remain the same (with the same proofs). In practice the algorithms do not differ much.

Remark. A similar algorithm is implemented in the package `vfgrep` (see [14]).

Acknowledgment

We thank anonymous referees for many useful comments, in particular for their help in gaining simplicity of the presentation.

References

[1] A.V. Aho, Algorithms for finding patterns in strings, in: J. van Leeuwen (Ed.), Handbook of Theoretical Computer Science, Vol. A, Algorithms and Complexity, Elsevier, Amsterdam, 1990, pp. 255–300.
[2] A.V. Aho, M. Corasick, Efficient string matching: An aid to bibliographic search, Comm. ACM 18 (6) (1975) 333–340.
[3] R.A. Baeza-Yates, M. Régnier, Fast algorithms for two-dimensional and multiple pattern matching, in: R. Karlsson, J. Gilbert (Eds.), Proc. 2nd Scandinavian Workshop in Algorithmic Theory, Lecture Notes in Computer Sci. 447, Springer, Berlin, 1990, pp. 332–347.
[4] A. Blumer, J. Blumer, A. Ehrenfeucht, D. Haussler, R. McConnel, Complete inverted files for efficient text retrieval and analysis, J. ACM 34 (3) (1987) 578–595.
[5] R.S. Boyer, J.S. Moore, A fast string searching algorithm, Comm. ACM 20 (10) (1977) 762–772.
[6] B. Commentz-Walter, A string matching algorithm fast on the average, Technical Report 79.09.007, IBM Heidelberg Scientific Center, Germany, 1979.

[7] B. Commentz-Walter, A string matching algorithm fast on the average, in: H.A. Maurer (Ed.), Proc. 6th Internat. Colloquium on Automata, Languages and Programming, Lecture Notes in Computer Sci., Springer, Berlin, 1979, pp. 118–132.
[8] M. Crochemore, A. Czumaj, L. Gąsieniec, S. Jarominek, T. Lecroq, W. Plandowski, W. Rytter, Speeding up two string matching algorithms, Algorithmica 12 (4/5) (1994) 247–267.
[9] M. Crochemore, W. Rytter, Text Algorithms, Oxford University Press, Oxford, 1994.
[10] D. Gusfield, Algorithms on Strings, Trees, and Sequences: Computer Science and Computational Biology, Cambridge University Press, Cambridge, 1997.
[11] A. Hume, Keyword matching, Internal Report of Bell Laboratories, NJ, 1990.
[12] D.E. Knuth, J.H. Morris Jr, V.R. Pratt, Fast pattern matching in strings, SIAM J. Comput. 6 (1) (1977) 323–350.
[13] T. Lecroq, A variation on Boyer–Moore algorithm, Theoret. Comput. Sci. 92 (1) (1992) 119–144.
[14] M. Raffinot, On the multi backward DAWG matching algorithm (MultiBDM), in: R. Baeza-Yates (Ed.), Proc. 4th South American Workshop on String Processing, Valparaiso, Chile, Carlton University Press, 1997, pp. 149–165.
[15] N. Uratani, A fast algorithm for matching patterns, Internal Report of IECEJ, Japan, 1988.
[16] A.C. Yao, The complexity of pattern matching for a random string, SIAM J. Comput. 8 (3) (1979) 368–387.