A String Matching Algorithm Fast on the Average

Extended Abstract[×)]

by

Beate Commentz-Walter
- FB 10 - Informatik
Universitaet des Saarlandes

currently:
IBM-Germany
Scientific Center Heidelberg
Tiergartenstrasse 15
D-6900 Heidelberg

0. Introduction

In many information retrieval and text-editing applications it is necessary to be able to locate quickly some or all occurrences of user-specified words or phrases in one or several arbitrary text strings. Specifically , we consider retrieval from unformatted data, for example, a libary data base where there is for each book a record containing the signature, title, and abstract of book. Each such record we call a document. A user of the data base specifies one or several words or phrases, so called keywords, describing the information sought. The answer will be the documents which contain all or some of the user specified keywords.It takes too much time to scan each document of the data base for every user seperately. Therefore, we introduce a sort of secondary index (compare Scheck lit /12/) containing keyword fragments. Searching the index with the user specified keywords yields a superset of the documents required. This

----

×) For detailed version compare lit (4). The work reported here was done at the Heidelberg Scientific Center of IBM-Germany. It is part of a project dealing with subjects like Automatic Indexing, Clustering, and retrieval structures of unformatted data base.

superset contains documents where the fragments match but the keywords do not. These documents we want to reject. Therefore, we scan the documents of the superset for the user specified keywords.

Aho, Corasick lit /2/ describe an efficient algorithm doing this job. Their algorithm first preprocesses the keywords in time linear in the "total lengths of the keywords i. e. in the sum of the length of the keywords. Then their algorithm searches for the keyword occurrences in the document in time linear in document length (worst case).

The idea of this algorithm is based on the ideas of the Knuth-Morris-Pratt algorithm lit /10/ and those of finite state machines.

If there is only one keyword to search for in some document, Boyer, Moore lit /3/ give an algorithm the preprocessing phase of which also runs linearly and the search phase of which is faster on the average than Aho-Corasick's algorithm.

In the case of large alphabets the Boyer-Moore algorithm takes time about $|D|/|W|$ on the average to search for all occurrences of the keyword W in document D (where $|S|$ denotes the length of string S).

With the modification due to Galil lit /7/ the search phase of the Boyer-Moore algorithm behaves linearly in the document length even in the worst case. This is proved by Knuth, Morris, Pratt lit /10/, Guibas, Odlyzko lit /8/, and Galil lit /7/.

We give an algorithm B for a set of keywords. Its search phase behaves similar to the Boyer-Moore algorithm, sublinear on the average. It does not maintain linear search time for the worst case. Modification to B yield algorithm B1, which does maintain linear search time. But for practical purposes algorithm B is more useful. The overhead of algorithm B1 is very high.

These algorithms B and B1 combine the idea of the Aho-Corasick and Boyer-Moore algorithms.

This short paper concentrates on algorithm B .

Chapter I describes the structure of the preprocessed set of keywords.

Chapter II describes the search phase of B.

Chapter III describes the preprocessing phase of B.

Chapter IV considers B's running time.

Chapter V outlines the modification for B1.

Acknowledgement:

I want to to thank G. Jaeschke and G. Walch for stimulating discussions, hints and critical remarks, M. Zoeppritz for editing my English and R.Scherner for typing the paper.

I. The Structure of the Preprocessed Set of Keywords

To represent some given set of keywords in a useful way, we consider the data structure of a trie:

A trie is a tree T such that:

1. Each node v of T, except the root r is labeled by some character a = l(v), an element of some alphabet A.

2. The root r is labeled by ε, denoting the empty word.

3. If the nodes v' and v" are brothers (sons of the same node v), v' ≠ v" then l(v') ≠ l(v").

We say a path $v_1, \ldots, v_m$ of T where v_{i+1} is son of v_i represents the word $l(v_1)\, l(v_2) \ldots l(v_m)$.
This word we denote by $w(v_m)$ iff $v_1 = r$, the root.

Moreover, for each node we denote its depth by

$$d(v) = \begin{cases} 0 & \text{if } v = r\text{, the root} \\ d(v')+1 & \text{if } v \text{ son of } v' \end{cases}$$

and by

$$d(T) = \max \{d(v);\ v \in T\}$$

we denote the depth of trie T.

Now, let $K = W_1, \ldots, W_r$ be the set of keywords on some alphabet A which we want to search for in some document D.
Similar to Aho, Corasick lit /2/ we represent K by a trie T. But in contrast, we base our trie T on the reversed keywords:

Exactly for h = 1...r there is one node v_h of T representing the reversed keyword W_h^R.

i.e.

$$w(v_h) = w_{h,1}, \ldots, w_{h,|W_h|}$$

$$\text{where } W_h = w_{h,|W_h|}, \ldots, w_{h,1}$$

To each node we add an output function

$$out(v) = \{ W \;;\; W^R = w(v),\ W \text{ in } K \}$$

To this trie we add the functions, shift1 and shift2, which map each trie node to an integer. Their purpose will become obvious from the description of the search phase of algorithm B. (Compare ChapterII):

The definition of shift1 and shift2 is based on sets of nodes:
For each $v \neq r$ of T:

$$\text{set1}(v) = \{v';\ w(v) \text{ is proper suffix of } w(v') \text{ i.e. } w(v') = u\, w(v) \text{ for some non empty word } u\}$$

and

$$\text{set2}(v) = \{\ v';\ v' \text{ is element of set1}(v) \text{ and out}(v') \neq \emptyset\ \}$$

Now shift1 and shift2 are defined by:

$$\text{shift1}(v) = \begin{cases} 1 & \text{if } v = r \\ \min(\ \{k\ ;\ k = d(v')-d(v),\ v' \text{ is element of set1}(v)\} \cup \{\ \text{wmin}\ \}\ ) & \text{else} \end{cases}$$

$$\text{shift2}(v) = \begin{cases} \text{wmin} & \text{if } v = r \\ \min(\ \{k\ ;\ k = d(v')-d(v),\ v' \text{ is element of set2}(v)\} \cup \text{shift2}(v' \text{'s father}\}) & \text{else} \end{cases}$$

Let

$$\text{wmax} = \max\ \{|W_h|;\ l = 1,\ldots,r\}$$

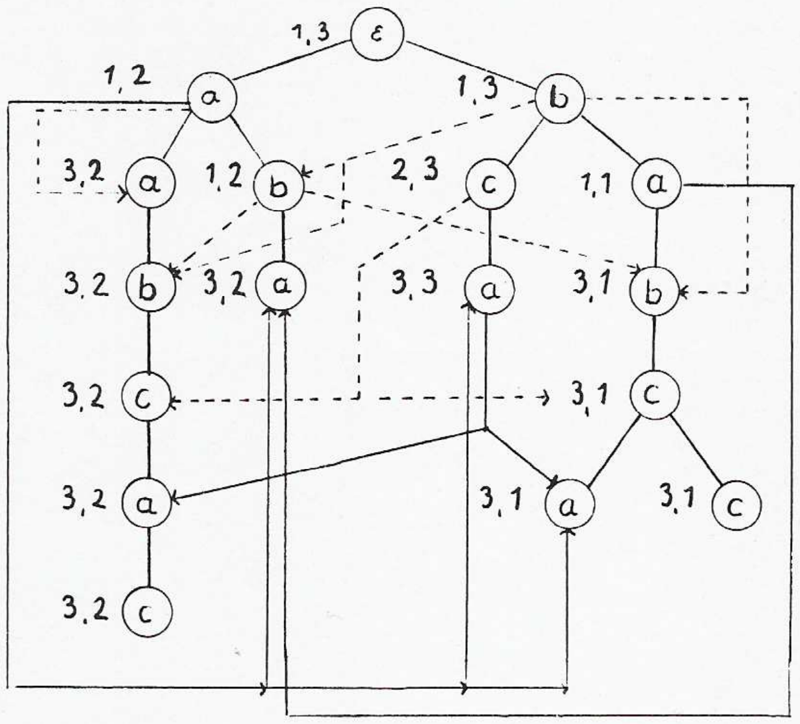
$$\text{wmin} = \min\ \{|W_h|;\ l = 1,\ldots,r\}$$

Finally we add a function

$$\text{char}: A \longrightarrow N \text{ where}$$

$$\text{char}(a) = \min(\ \{d(v);\ l(v) = a\}\ \ \{\text{wmin} + 1\}\ )$$

<u>Example</u>: $k = \{\ \text{cacbaa, acb, aba, acbab, ccbab}\ \}$, wmin = 3



For each node v ≠ r --→ and → point to the nodes of set1(v) where → points to set2(v).

The two integers beside each node v denote the functions shift1(v), shift(2)v.

II. The Search Phase of Algorithm B for String Matching Fast on the Average

The input for the search phase of algorithm B is some document D and, for some keyset K, the preprocessed trie T and the functions out, shift1, shift2, and char.

The output of the search phase of algorithm B is a list of pairs (W,i) where W is a word and i is an integer representing the occurrence of W, i.e.

(W,i) element of the output of B

iff

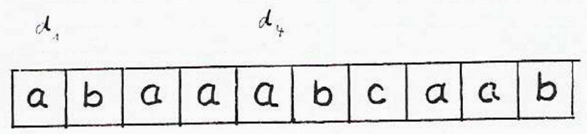
W is a keyword of K and $d_{i-|W|+1},\dots,d_i = W$.

Aho, Corasick lit /2/ also represent the set of keywords K by a trie T. Searching for the occurrences of any keyword W in any document D they compare the letters of the trie T with the letters of the document D left to right until mismatch occurs.

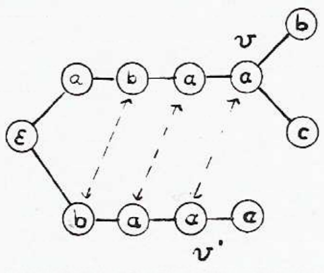
If mismatch occurs, the root is "shifted right along the document" by a number of letters calculated from the matching letters just scanned.

Example:

documents:

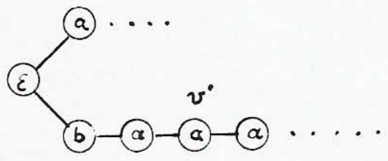


trie:



Mismatch occurs at d_4 and node v as non of its sons is labeled by a

shift:



w(v') is the maximal praefix of some keyword, which is suffix of w(v)

For detail compare lit /2/.
The Boyer-Moore algorithm lit /3/ starts putting the keyword (only one) beneath the left end of the document. It differs from the Aho-Corasick algorithm in that it compares the letters of the document with the letters of the keyword from right to left. If mismatch occurs, the keyword is shifted right by a number of letters calculated from the matching

letters and the mismatch character. This right to left scan and left to right shift yields a sublinear behaviour on the average, lit /3/, and the linear worst case behaviour is easy to preserve, lit /7,8,10/.

We combine these ideas:
We base our trie on the reversed keywords. Let wmin denote the minimal length of some keyword. The algorithm B starts putting the root r of T underneath d_{wmin+1}. Next it "scans" the document right to left until mismatch occurs. (For detail compare the algorithm mentioned below). Assuming we have just scanned the matching document letters $d_{i-m+1},\dots,d_i$ and a mismatch occured at letter d_{i-m} we then shift the trie root right by some number of letters S calculated from the document letters $d_{i-m},\dots d_i$.

The search phase of algorithm B in detail:

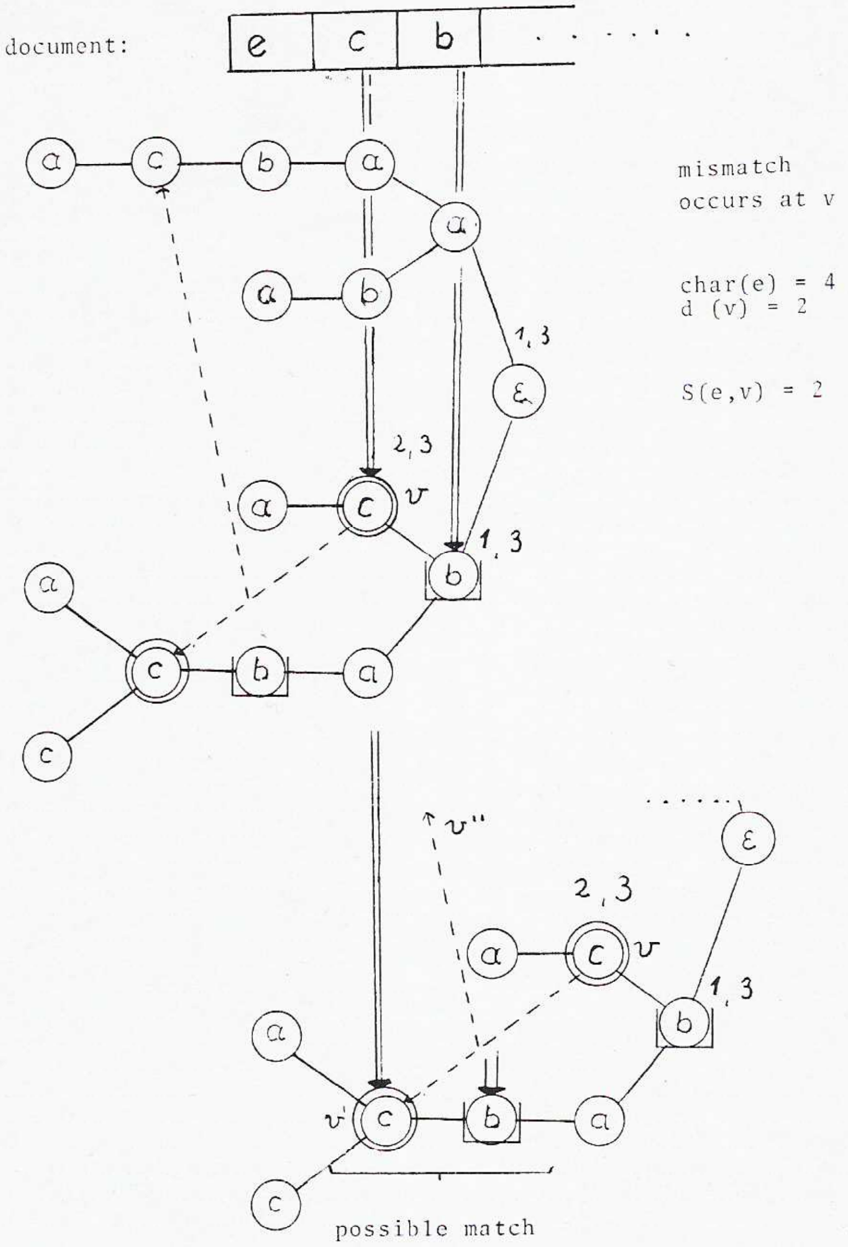
Initial phase:

```
        v ← root r      (v is the "present" node of T)

        i ← wmin        (i points to the document letter
                        above the nodes of depth 1 .)

        j ← 0           (j indicates the depth of the
                        present node v.)

While i ≤ length document do

Scan phase:
begin
 while there is some son v' of v labeled by $d_{i-j}$ do
  begin

   v ← v'
   j ← j + 1
   output: (W,i) for each W of out(v)
  end
shift phase:
begin
 i ← i + S(v,$d_{i-j}$)
 j ← 0
end end
```

where $S(v, d_{i-j})$ is the length of the shift defined by

$$S(v,d_{i-j}) = \min(\max(\text{shift1}(v), \text{char}(d_{i-j})-j-1), \text{shift2}(v)).$$

Example:



Obviously, each pair (W,i) found by B represents some occurrence of the keyword W. So it remains to show, that B finds each occurrence of some keyword in the document D.

Due to the construction of B's search phase it is sufficient to show that no shift is too long.

i. e. $d_{i-j+1},\dots,d_i = w_t{}^R(v)$ for some v of T implies there is no i' such that:

1.) $i < i' < S(v, d_{i-j})$

and 2.) $d_{i'-|W|+1},\dots,d_{i'} = W$

for some keyword W.

Due to the construction of $S(v,d_{i-j})$ this is easy to show.

III. The Preprocessing Phase of Algorithm B

The input of the preprocessing phase of B is the set of keywords $K = \{W_1,\dots,W_r\}$. Its output is the trie T of the reversed keywords and the functions out, shift1, shift2, and char.

We shall show that the time used by the preprocessing phase is linear in the total length of the keywords i.e. in the sum of the lengths of the keywords $W_1,\dots,W_r$.

Obviously, the time of computing the trie T and the functions out and char is linear in the total length of the keywords. It remains to analyse the computation of shift1 and shift2.

Consider some function on T's nodes:

$f(v') = v$ where $w(v)$ is maximal proper suffix of $w(v')$ in T.

This function coincides with the failure function of Aho-Corasick's pattern matching machine.

The inversion of f is given by

$$set1'(v) = \{v'; f(v') = v\}$$

Obviously set1'(v) is subset of set1(v). Moreover it contains the nodes v' of set1(v) where d(v')-d(v) is minimal. Hence due to lit /2/ the computation of shift1 is linear in the total length of the keywords.

The computation of shift2 can be done analogusly using

$$set2'(v)= \{v'; v' \text{ is element of } set1'(v) \text{ and } out(v') \neq 0\}.$$

IV. The Average and Worst Case Behavior of the Search Phases of Algorithm B

The running time of the search phase of algorithm B splits into two parts; the running time to perform the scan phase and the running time to perform the computation of the shift $S(v, d_{i-j})$ whenever necessary.

The total running time for the scan and shift phases is linear in the total number of character comparisons. Hence we measure the speed of algorithm B by the number of character inspections which are performed.

As in the Boyer-Moore algorithm lit /3/: If the size of the alphabet A is large, the search phase needs to inspect only about |D|/wmin letters of the document on the average.

Unfortunately, the search phase of algorithm B can perform |D| x wmax letter comparisons in the worst case.

Notice, the search phase of the usual Boyer-Moore algorithm with changes due to Galil lit /7/, lit /8/ and lit /10/ does at most c|D| letter comparisons in the worst case.

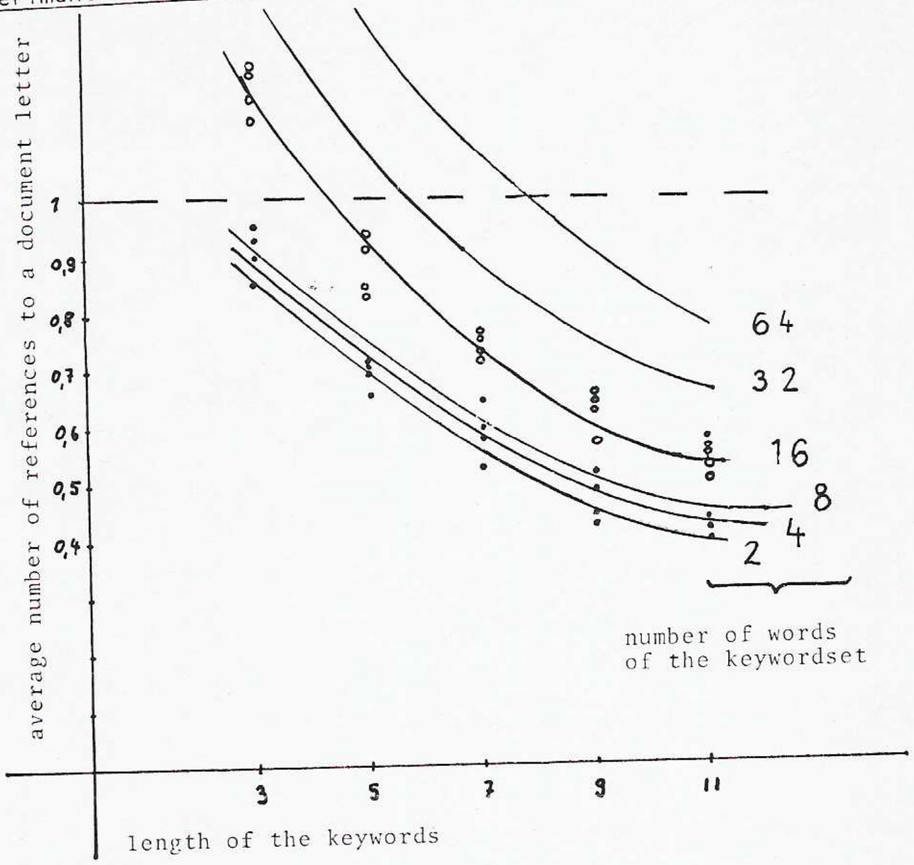
We did some experimental runs of algorithem B to get an estimate of its average behavior. Our experiments are based on 100 titles of English and German books on Computer Science and related subjects.These titles are our documents.

For the alphabet we took:
ALP = A B C D E F G H I J K L M N O P Q R S T U V W X Y Z 0
1 2 3 4 5 6 7 8 9 and blank.
From the set of titles we choose sets of strings to function as keyword sets.
The number of keywords in a set was to be : 2,4,8,16,32,64.
The length of a keyword in a set was to be : 3,5,7,9,11
For each possible pair of number and length we choose four sets of keywords.For each keywordset and the 100 titles the average number of references to a document letter by algorithm B is computed.
For each pair of number and length of keywords we take the average on the four different keyword probes. In the figure below this mean value is plotted against the length of the keywords for each different number of keywords. In addition, we indicate the average number of references to a document for each probe by a dot for number of keywords = 4 and by a circle for number of keywords = 16.

The results of the experiments show that the average behavior of algorithm B is sublinear.

For experimental results of other versions of algorithm B compare lit /4/.

Experimential Results:



V. The Construction of Algorithm B1, Linear for the Worst Case

Algorithm B1 differs from B in "remembering" the document letters already scanned. As "memory" it uses the trie T and some additional functions.

For detailed description compare lit /4/.

Because of this "memory" the search phase of B1 behaves linear in the worst case. Moreover, on the average it is probably faster than B. Of course we have to pay a price for this improvement: Some constant increase in time and space needed for the overhead of preprocessing and search phases Anyway, B1's preprocessing phase remains linear in the total length of keywords. The proof is based on lit /11/.

Literature:

/1/ Aho, A.V., Hopcraft, J. E.and Ullman J.D.
"The Design and Analysis of ComputerAlgorithms"
Addison-Wesley Publ. Comp. Read. Mass.

/2/ Aho, A.V. and Corasick, M.
"Efficient String Matching: An Aid to Bibliographic Search"
Com. ACM, June 75, Vol. 18, No. 6

/3/ Boyer, R.S. and Moore, J.S.
"A Fast String Searching Algorithm"
Com. of the ACM, Vol. 20, No. 10, 1977, 262-272

/4/ Commentz-Walter, B.
"A String Matching Algorithm Fast on the Average"
Scientific Center Heidelberg, Technical Report, in print.

/5/ Fagin, R., Nievergelt, J., Pippenger, N. and Strong, H.R.
"Extendible Hashing - A Fast Access Method for Dynamic Files"
IBM, Research Rep. RJ 2305, 1978 (San Jose)

/6/ Galil, Z.
"Saving Space in Fast String-Matching"
IBM, Reseach Rep., RC 6670, 1977 (Yorktown Heights)

/7/ Galil, Z.
"On Improving the Worst Case Running Time of Boyer-Moore String Matching Algorithm"
Automata, Languages and Programming, 5th Colloquium EATCS, July 1978

/8/ Guibas, L.J. and Odlyzko, A.M.
"A New Proof of the Linearity of the Boyer-Moore String Searching Algorithms"
Proceedings 18th Annual IEEE Symposium on Foundations of Computer Science, 1977

/9/ Guibas, L.J., McCreight, E.M., Plass, M.F. and Roberts, J.R.
"A new Representation for Linear Lists"
9th Annual ACM Symposium on Theory of Computing, 1977

/10/ Knuth, D.E., Morris Jr., J.H. and Pratt, V.B.
"Fast Pattern Matching in Strings"
SIAM J. on Computing, Vol. 6, No. 2, 1977, 323-350

/11/ McCreight, E.M.
"A Space Economical Suffix Tree Construction Algorithm"
Journal of the ACM, Vol. 23, No. 2, 1976, 262-272

/12/Scheck, H.-J.
"The Reference String Indexing Method" Proceedings Information System Methodology, Venice 1978 Lecture Notes in Comp. Sc. 65 Springer Heidelberg 1976

/13/ Weiner, P.
"Linear Pattern Matching Algorithm"
Proceedings 14th Annual IEEE Symposium in Switching and Automata Theory, 1973, 1-11