

Fast exact string matching algorithms

Thierry Lecroq

LITIS

Faculté des Sciences et des Techniques

Université de Rouen

76821 Mont-Saint-Aignan Cedex

France

Abstract

String matching is the problem of finding all the occurrences of a pattern in a text. We propose a very fast new family of string matching algorithms based on hashing q -grams. The new algorithms are the fastest on many cases, in particular on small size alphabets.

Key words: String matching, hashing, design of algorithms

1 Introduction

The string matching problem consists in finding one or more usually all the occurrences of a pattern $x = x[0..m-1]$ of length m in a text $y = y[0..n-1]$ of length n . It can occur for instance in information retrieval, bibliographic search and molecular biology. It has been extensively studied and numerous techniques and algorithms have been designed to solve this problem (see [10,3]). We are interested here in the problem where the pattern is given first and can then be searched in various texts. Thus a preprocessing phase is allowed on the pattern.

Basically a string-matching algorithm uses a window to scan the text. The size of this window is equal to the length of the pattern. It first aligns the left ends of the window and the text. Then it checks if the pattern occurs in the window (this specific work is called an *attempt*) and *shifts* the window to the

Email address: Thierry.Lecroq@univ-rouen.fr (Thierry Lecroq).

URL: monge.univ-mlv.fr/~lecroq (Thierry Lecroq).

right. It repeats the same procedure again until the right end of the window goes beyond the right end of the text.

The brute force algorithm performs a quadratic number of symbol comparisons. There exist a lot of linear solutions (see [10,3]). Hashing provides a simple method to avoid a quadratic number of character comparisons in most practical situations. It has been introduced by Karp and Rabin [8]. Instead of checking at each position of the text if the pattern occurs, it seems to be more efficient to check only if the contents of the window “looks like” the pattern. In order to check the resemblance between these two words an hashing function h is used. The preprocessing phase of the Karp-Rabin algorithm consists in computing $h(x)$. It can be done in constant space and $O(m)$ time. During searching phase, it is enough to compare $h(x)$ with $h(y[j..j+m-1])$ for $0 \leq j < n - m$. If an equality is found, it is still necessary to check the equality $x = y[j..j+m-1]$ character by character. The time complexity of the searching phase of the Karp-Rabin [8] algorithm is $O(mn)$ (when searching for a^m in a^n for instance). Its expected number of text character comparisons is $O(n + m)$.

The algorithm of Wu and Manber [12] is an algorithm for searching for all the occurrences of the patterns of a finite set $X = \{x_0, x_1, \dots, x_{k-1}\}$ in a text y . It considers substrings of length q . The preprocessing phase of this algorithm consists in computing a shift for all the possible strings of length q . For that all the substrings B of length q of every pattern in X are hashed using a function h into values within 0 and $maxvalue$. Then $shift[h(B)]$ is defined as the minimum between $|x_i| - j$ and $lmin - q + 1$ when $B = x_i[j - q + 1] \dots x_i[j]$ for $0 \leq i \leq k - 1$ and $0 \leq j \leq |x_i| - 1$ where $lmin$ denotes the length of the shortest pattern in X . In practice, the value of q varies with $lmin$ and the size of the alphabet and the value of $maxvalue$ varies with the memory space available.

The searching phase of the algorithm consists in reading substrings B of length q . If $shift[h(B)] > 0$ then a shift of length $shift[h(B)]$ is applied. Otherwise, when $shift[h(B)] = 0$ the patterns ending with the substring B are examined one by one in the text. The first substring to be scanned is $y[lmin - q + 1..lmin]$. This method is incorporated in the *agrep* command.

In this article we present an adaptation of the Wu and Manber multiple string matching algorithm to single string matching algorithm. We propose then very efficient implementations of this algorithm that in many cases are much faster than the previous known fastest string matching algorithms.

This article is organized as follows: section 2 presents the new family of algorithms, section 3 shows experimental results and section 4 provides our conclusion.

2 The new algorithm

The idea of the new algorithm is to consider substrings of length q . Substrings B of such a length are hashed using a function h into integer values within 0 and 255. For $0 \leq c \leq 255$

$$\mathit{shift}[c] = \begin{cases} m - 1 - i & \text{with } i = \max\{0 \leq j \leq m - q + 1 \mid h(x[j..j + q - 1]) = c\} \\ m - q & \text{when such an } i \text{ does not exist} \end{cases}$$

The searching phase of the algorithm consists in reading substrings B of length q . If $\mathit{shift}[h(B)] > 0$ then a shift of length $\mathit{shift}[h(B)]$ is applied. Otherwise, when $\mathit{shift}[h(B)] = 0$ the pattern x is naively checked in the text. In this case a shift of length sh is applied where $sh = m - 1 - i$ with $i = \max\{0 \leq j \leq m - q \mid h(x[j..j + q - 1]) = h(x[m - q + 1..m - 1])\}$.

The key features of the algorithm to be as fast as possible are:

- Set $y[n..n + m - 1]$ to x in order to avoid testing the end of the text but exit the algorithm only when an occurrence of x is found. If this is not possible (because memory space is occupied, it is always possible to store $y[n - m..n - 1]$ in z then set $y[n - m..n - 1]$ to x and check z at the end of the algorithm without slowing it.
- Unroll the loops as frequently as possible, i.e. writing q consecutive instructions when computing $h[B]$ for a substring B rather than a loop which is much more time consuming.

The algorithm for $q = 3$ is presented in Figure 1.

3 Experimental results

To evaluate the efficiency of the new string matching algorithms we perform several experiences with different algorithms on different data sets.

3.1 Algorithms

We have tested 17 algorithms:

- The brute force algorithm (BF).
- One implementation of the Boyer–Moore algorithm: with the best matching shift with fast loop (**BM2fast**) [4].

```

Algorithm NEW3( $x, m, y, n$ )
  ▷ Preprocessing
  for  $a \in \Sigma$  do  $shift[a] \leftarrow m - 2$ 
   $h \leftarrow x[0], \quad h \leftarrow 2h + x[1], \quad h \leftarrow 2h + x[2]$ 
   $shift[h \bmod 256] \leftarrow m - 3$ 
  for  $i \leftarrow 3$  to  $m - 2$  do
     $h \leftarrow x[i - 2], \quad h \leftarrow 2h + x[i - 1], \quad h \leftarrow 2h + x[i]$ 
     $shift[h \bmod 256] \leftarrow m - 1 - i$ 
   $h \leftarrow x[m - 3], \quad h \leftarrow 2h + x[m - 2], \quad h \leftarrow 2h + x[m - 1]$ 
   $sh1 \leftarrow shift[h \bmod 256], \quad shift[h \bmod 256] \leftarrow 0$ 
  ▷ Searching
   $y[n..n + m - 1] \leftarrow x, \quad j \leftarrow m - 1$ 
  while TRUE do
     $sh \leftarrow 1$ 
    while  $sh \neq 0$  do
       $h \leftarrow y[j - 2], \quad h \leftarrow 2h + y[j - 1], \quad h \leftarrow 2h + y[j]$ 
       $sh \leftarrow shift[h \bmod 256]$ 
       $j \leftarrow j + sh$ 
    if  $j < n$  then
      if  $x = y[j - m + 1..j]$  then REPORT( $j - m + 1$ )
       $j \leftarrow j + sh1$ 
    else RETURN

```

Figure 1. The new string matching algorithm with $q = 3$.

- The Tuned-BM algorithm [7] (TBM) with 3 unrolled shifts.
- The SSABS algorithm [11] (SSABS).
- The Zhu-Takaoka algorithm [13] (ZT).
- The Fast Search algorithm [2] (FS).
- One algorithm based on an index structure recognizing all the factors of the reverse of x : the Backward Oracle Matching [1] algorithm (BOM2) where the factor oracle is implemented in quadratic space with a transition matrix.
- For short patterns, four algorithms using bitwise operations:
 - The Backward Nondeterministic Dawg Matching algorithm [9] (BNDM).
 - The Simplified Backward Nondeterministic Dawg Matching algorithm which main loop starts with a test and loop-unrolling [6] (SBNDM2).
 - The Fast Average Optimal Shift Or algorithm [5] (FAOSO). It consist in considering sparse q -grams of x and unrolling u shifts, thus $q(\lceil m/q \rceil + u) \leq w$ should holds where w is the number of bits of a machine word.
- The new algorithms (New q) for $3 \leq q \leq 8$. It was not fastest when computing $h \bmod 256$ to consider h as an `unsigned char` and computing implicitly the `mod` operation than having h as an integer and computing explicitly the `mod` operation.

These algorithms have been coded in C in an homogeneous way to keep the comparison significant. The programs have been compiled with `gcc` with the full optimization option `-O3`. The machine we used has an Intel Pentium processor at 1300MHz running Linux Red Hat version 2.4.20-8. The running times for the search of 100 patterns have been measured using the `clock` function.

3.2 Data

We give experimental results for the running times for the above algorithms for different types of text: random texts on binary alphabet and alphabet of size 8, a genome and a text in natural language (English). We consider short patterns (odd length within 5 and 31) and long patterns (length power of two from 2^5 to 2^{10}). For each length we made the search for 100 patterns randomly chosen from the text.

We use 4 different texts:

- Binary alphabet and alphabet of size 8: The texts are composed of 4,000,000 characters and were randomly built. The symbol distribution is uniform.
- Genome: A genome is a DNA sequence composed of the four nucleotides, also called base pairs or bases: Adenine, Cytosine, Guanine and Thymine. The genome we used for these tests is a sequence of 4,638,690 base pairs of *Escherichia coli*. We used the file `E.coli` of the Large Canterbury Corpus¹.
- Natural language: We used the file `world192.txt` (The CIA world fact book) of the Large Canterbury Corpus. The alphabet is composed of 94 different characters. The text is composed of 2,473,400 characters.

3.3 Results

The results for short patterns (length less than 32) are presented in tables 1 to 4. The results for long patterns (length more than 32) are presented in tables 5 to 8.

For short patterns the new algorithms perform very well: on a binary alphabet, it is the fastest on patterns from length 11 to 21 with $q = 6$ and on patterns from length 23 to 31 with $q = 7$. On the considered genome sequence, it is the fastest on patterns of length from 7 to 9 with $q = 3$, on patterns of length from 11 to 21 with $q = 4$ and on length from 23 to 31 with $q = 5$. On an alphabet of size 8, they compete with BNDM2 while they are a bit slower on the considered English text.

¹ <http://www.data-compression.info/Corpora/CanterburyCorpus/>

Table 1
Results for short patterns on a binary alphabet

	5	7	9	11	13	15	17	19	21	23	25	27	29	31
BF	41.29	25.84	21.54	20.21	20.26	20.10	20.25	20.50	20.16	20.08	20.08	20.09	20.05	20.07
BM2fast	26.03	8.93	4.15	2.98	2.89	2.65	2.59	2.35	2.26	2.16	2.15	2.00	1.87	1.96
TBM	27.72	12.25	7.74	6.85	6.12	6.65	6.40	6.37	6.14	6.00	6.58	6.22	5.99	6.59
SSABS	30.40	10.86	7.25	6.45	5.93	6.24	6.09	5.88	5.75	5.99	5.86	6.02	5.87	5.98
ZT	30.07	10.10	5.51	4.02	3.50	3.18	3.21	2.91	2.80	2.70	2.57	2.55	2.45	2.45
FS	28.34	9.80	5.08	3.05	2.56	2.25	2.31	2.13	2.02	1.96	1.91	1.77	1.68	1.73
BOM2	24.78	10.12	4.49	3.00	2.27	1.85	1.78	1.64	1.37	1.24	1.16	1.07	1.01	0.96
BNDM	25.13	9.28	3.98	2.53	2.28	2.03	1.82	1.57	1.43	1.31	1.21	1.13	1.08	1.01
SBNDM	31.53	10.34	4.52	2.74	1.90	1.60	1.37	1.22	1.12	0.99	0.92	0.86	0.79	0.73
SBNDM2	28.39	9.38	3.66	2.20	1.58	1.30	1.13	0.96	0.91	0.84	0.75	0.71	0.65	0.62
FAOSO	10.48	4.70	3.74	3.90	4.84	4.93	1.15	1.13	2.49	2.47	1.57	2.93	2.97	2.19
NEW3	27.24	8.05	3.16	1.82	1.50	1.20	1.20	1.21	1.22	1.20	1.17	1.01	0.98	1.00
NEW4	27.26	7.86	2.80	1.31	0.94	0.89	0.81	0.77	0.72	0.69	0.68	0.65	0.64	0.64
NEW5		8.24	2.99	1.32	0.79	0.70	0.64	0.62	0.57	0.54	0.53	0.51	0.50	0.50
NEW6		9.61	2.86	1.30	0.79	0.74	0.62	0.58	0.52	0.50	0.48	0.47	0.46	0.45
NEW7			2.93	1.32	0.98	0.77	0.67	0.59	0.54	0.48	0.48	0.45	0.45	0.44
NEW8			2.02	1.70	1.37	0.87	0.70	0.61	0.56	0.52	0.50	0.48	0.48	0.46

Table 2
Results for short patterns on the *E. coli* genome

	5	7	9	11	13	15	17	19	21	23	25	27	29	31
BF	22.75	22.16	22.74	22.52	22.89	22.55	22.47	22.50	22.44	22.09	22.04	22.04	22.06	22.03
BM2fast	2.82	2.01	1.75	1.60	1.45	1.41	1.28	1.27	1.24	1.17	1.13	1.11	1.10	1.06
TBM	3.11	2.11	1.83	1.82	1.81	1.79	1.79	1.84	1.90	1.80	1.80	1.82	1.82	1.82
SSABS	3.37	2.36	2.23	2.29	2.19	2.27	2.19	2.31	2.27	2.19	2.20	2.29	2.25	2.21
ZT	3.45	2.45	2.05	1.72	1.53	1.41	1.32	1.25	1.19	1.16	1.12	1.09	1.05	1.03
FS	3.16	2.11	1.83	1.80	1.69	1.67	1.54	1.57	1.54	1.52	1.54	1.52	1.54	1.51
BOM2	3.37	2.31	1.88	1.54	1.34	1.18	1.08	1.01	0.92	0.87	0.81	0.76	0.73	0.70
BNDM	3.45	2.39	1.95	1.57	1.36	1.20	1.07	0.99	0.90	0.83	0.79	0.75	0.71	0.70
SBNDM	4.79	2.73	1.99	1.55	1.39	1.22	1.08	0.99	0.87	0.80	0.77	0.74	0.69	0.66
SBNDM2	3.86	1.87	1.37	1.13	0.96	0.92	0.81	0.73	0.68	0.64	0.65	0.56	0.56	0.56
FAOSO	2.59	2.52	1.30	1.54	1.70	1.62	1.61	1.60	2.27	2.33	1.59	1.54	1.60	1.40
NEW3	2.86	1.19	0.89	0.72	0.63	0.58	0.54	0.53	0.51	0.51	0.48	0.49	0.53	0.48
NEW4	3.56	1.45	1.03	0.70	0.63	0.57	0.54	0.52	0.51	0.50	0.49	0.51	0.47	0.49
NEW5		1.94	1.28	0.88	0.71	0.62	0.54	0.51	0.53	0.49	0.49	0.38	0.42	0.44
NEW6		3.06	1.70	1.17	0.85	0.70	0.59	0.56	0.53	0.54	0.53	0.49	0.50	0.49
NEW7			2.42	1.47	1.06	0.85	0.70	0.58	0.56	0.51	0.51	0.50	0.52	0.50
NEW8			3.62	1.95	1.31	0.99	0.78	0.66	0.58	0.56	0.59	0.53	0.53	0.50

For long patterns, the new algorithms are the fastest from length 32 to 256 on the binary alphabet, from length 32 to 128 on the genome, from length 64 to 128 on the alphabet on size 8 and the English text.

4 Conclusion

In this article we presented simple and though very fast adaptations and implementations of the Wu-Manber exact multiple string matching algorithm to the case of exact single string matching algorithm. Experimental results show that the new algorithm are very fast for short patterns on small size alphabets comparing to the well known fast algorithms using bitwise techniques. The

Table 3
Results for short patterns on an alphabet of size 8

	5	7	9	11	13	15	17	19	21	23	25	27	29	31
BF	18.62	18.96	19.22	19.17	19.11	19.12	19.10	19.15	19.29	19.14	19.16	19.15	19.13	19.15
BM2fast	1.12	0.81	0.76	0.67	0.65	0.61	0.62	0.57	0.58	0.55	0.55	0.55	0.52	0.52
TBM	1.10	0.85	0.73	0.72	0.65	0.61	0.64	0.62	0.61	0.60	0.62	0.60	0.61	0.61
SSABS	1.23	0.96	0.88	0.84	0.80	0.77	0.73	0.72	0.71	0.73	0.74	0.72	0.72	0.74
ZT	1.83	1.45	1.13	1.01	0.85	0.75	0.71	0.66	0.61	0.63	0.57	0.58	0.57	0.55
FS	1.29	0.91	0.83	0.74	0.71	0.68	0.69	0.63	0.63	0.63	0.64	0.63	0.62	0.62
BOM2	1.92	1.31	1.06	0.87	0.75	0.68	0.63	0.57	0.56	0.55	0.51	0.48	0.54	0.47
BNDM	1.92	1.42	1.13	0.92	0.82	0.72	0.64	0.62	0.58	0.55	0.51	0.50	0.49	0.50
SBNDM	2.40	1.75	1.41	1.15	0.93	0.82	0.75	0.66	0.60	0.58	0.54	0.52	0.49	0.48
SBNDM2	1.75	0.90	0.68	0.60	0.52	0.50	0.46	0.42	0.45	0.41	0.42	0.41	0.39	0.41
FAOSO	1.73	1.50	0.85	0.68	0.70	0.65	0.66	0.66	1.21	1.21	1.07	1.13	1.11	1.16
NEW3	1.61	0.94	0.74	0.61	0.52	0.52	0.50	0.47	0.46	0.45	0.45	0.42	0.44	0.42
NEW4	2.32	1.21	0.87	0.63	0.56	0.48	0.50	0.45	0.44	0.43	0.45	0.40	0.41	0.44
NEW5		1.63	1.08	0.74	0.60	0.50	0.50	0.47	0.45	0.45	0.44	0.44	0.42	0.40
NEW6		2.77	1.52	0.93	0.74	0.58	0.49	0.49	0.47	0.45	0.45	0.42	0.44	0.42
NEW7			2.40	1.28	0.93	0.74	0.60	0.52	0.47	0.46	0.46	0.44	0.43	0.44
NEW8				3.38	1.56	1.06	0.79	0.68	0.58	0.50	0.49	0.47	0.43	0.42

Table 4
Results for short patterns on an English text

	5	7	9	11	13	15	17	19	21	23	25	27	29	31
BF	11.99	11.63	11.67	11.54	11.57	11.55	11.51	11.51	11.54	11.51	11.51	11.50	11.52	11.51
BM2fast	0.68	0.40	0.35	0.31	0.29	0.28	0.27	0.27	0.27	0.26	0.26	0.26	0.25	0.25
TBM	0.81	0.40	0.36	0.30	0.29	0.28	0.27	0.26	0.26	0.27	0.25	0.26	0.25	0.25
SSABS	0.66	0.41	0.37	0.33	0.31	0.29	0.28	0.28	0.28	0.26	0.27	0.27	0.27	0.26
ZT	1.22	0.81	0.64	0.54	0.46	0.42	0.39	0.36	0.35	0.34	0.33	0.33	0.33	0.32
FS	0.69	0.43	0.36	0.32	0.31	0.29	0.28	0.28	0.26	0.27	0.27	0.27	0.25	0.25
BOM2	0.92	0.66	0.56	0.46	0.40	0.38	0.34	0.33	0.31	0.31	0.29	0.28	0.28	0.27
BNDM	0.96	0.67	0.52	0.48	0.41	0.38	0.35	0.33	0.32	0.30	0.29	0.28	0.28	0.27
SBNDM	1.37	0.75	0.60	0.50	0.45	0.42	0.38	0.34	0.33	0.26	0.31	0.29	0.27	0.31
SBNDM2	1.30	0.53	0.41	0.30	0.30	0.28	0.26	0.22	0.22	0.23	0.20	0.21	0.18	0.21
FAOSO	1.03	0.66	0.49	0.33	0.31	0.31	0.32	0.33	0.66	0.69	0.68	0.69	0.68	0.68
NEW3	1.24	0.52	0.48	0.36	0.34	0.28	0.27	0.28	0.26	0.29	0.28	0.23	0.23	0.22
NEW4	1.61	0.78	0.54	0.38	0.33	0.33	0.29	0.28	0.25	0.25	0.24	0.25	0.26	0.24
NEW5		1.07	0.73	0.45	0.38	0.33	0.30	0.29	0.29	0.26	0.26	0.27	0.24	0.28
NEW6		1.74	0.93	0.60	0.46	0.36	0.33	0.30	0.26	0.27	0.24	0.28	0.24	0.27
NEW7			1.36	0.73	0.56	0.41	0.36	0.32	0.29	0.29	0.29	0.28	0.28	0.27
NEW8				2.40	0.95	0.66	0.52	0.43	0.38	0.32	0.30	0.29	0.28	0.27

Table 5
Results for long patterns on a binary alphabet

	32	64	128	256	512	1024
BF	20.19	20.22	20.20	20.21	20.22	20.15
BM2fast	1.92	1.42	1.23	1.11	0.97	0.90
TBM	6.24	6.18	6.02	6.24	6.14	6.19
SSABS	5.50	5.69	5.76	5.76	5.62	5.78
ZT	2.42	1.83	1.54	1.34	1.13	0.99
FS	1.69	1.26	1.07	0.99	0.81	0.71
BOM2	0.92	0.60	0.65	0.38	0.21	0.19
NEW3	1.36	1.29	1.22	1.37	1.27	1.30
NEW4	0.65	0.59	0.56	0.58	0.56	0.60
NEW5	0.49	0.46	0.45	0.43	0.46	0.42
NEW6	0.45	0.42	0.41	0.42	0.39	0.37
NEW7	0.45	0.38	0.44	0.35	0.32	0.29
NEW8	0.44	0.42	0.44	0.33	0.23	0.20

Table 6
Results for long patterns on the *E. coli* genome

	32	64	128	256	512	1024
BF	23.46	25.85	23.31	23.38	23.39	23.56
BM2fast	1.12	0.91	0.89	0.78	0.66	0.67
TBM	1.84	1.87	1.92	1.88	1.78	1.91
SSABS	2.31	2.33	2.46	2.31	2.40	2.39
ZT	1.11	0.98	0.99	0.94	0.86	0.78
FS	1.37	1.20	1.17	1.03	0.91	0.86
BOM2	0.71	0.52	0.53	0.31	0.20	0.18
NEW3	0.51	0.44	0.44	0.43	0.38	0.40
NEW4	0.49	0.42	0.49	0.38	0.28	0.28
NEW5	0.46	0.41	0.48	0.35	0.28	0.22
NEW6	0.49	0.40	0.48	0.37	0.26	0.24
NEW7	0.48	0.42	0.52	0.35	0.27	0.23
NEW8	0.50	0.44	0.52	0.37	0.25	0.22

Table 7
Results for long patterns on an alphabet of size 8

	32	64	128	256	512	1024
BF	18.24	19.17	19.11	19.17	18.85	18.78
BM2fast	0.54	0.50	0.53	0.47	0.42	0.48
TBM	0.61	0.61	0.60	0.62	0.62	0.62
SSABS	0.72	0.72	0.71	0.71	0.69	0.74
ZT	0.53	0.49	0.48	0.47	0.46	0.44
FS	0.60	0.58	0.58	0.56	0.51	0.46
BOM2	0.47	0.38	0.33	0.17	0.12	0.12
NEW3	0.42	0.41	0.40	0.38	0.35	0.37
NEW4	0.43	0.39	0.41	0.35	0.30	0.28
NEW5	0.39	0.37	0.43	0.30	0.25	0.20
NEW6	0.42	0.37	0.46	0.30	0.21	0.21
NEW7	0.41	0.39	0.44	0.31	0.22	0.18
NEW8	0.41	0.38	0.45	0.31	0.20	0.21

Table 8
Results for long patterns on an English text

	32	64	128	256	512	1024
BF	11.92	11.91	11.90	11.92	11.98	11.99
BM2fast	0.26	0.22	0.27	0.22	0.22	0.25
TBM	0.25	0.23	0.23	0.18	0.13	0.09
SSABS	0.26	0.24	0.24	0.18	0.13	0.09
ZT	0.31	0.29	0.31	0.19	0.12	0.10
FS	0.26	0.24	0.26	0.18	0.13	0.10
BOM2	0.27	0.21	0.16	0.09	0.06	0.10
NEW3	0.25	0.21	0.25	0.17	0.11	0.10
NEW4	0.25	0.24	0.23	0.16	0.11	0.10
NEW5	0.24	0.24	0.24	0.16	0.10	0.09
NEW6	0.28	0.23	0.26	0.16	0.12	0.10
NEW7	0.24	0.23	0.27	0.18	0.11	0.10
NEW8	0.26	0.24	0.26	0.19	0.11	0.09

new algorithms are also fast on long patterns (length 32 to 256) comparing to algorithms using an indexing structure for the reverse pattern (namely the Backward Oracle Matching algorithm). This new type of algorithm can serve as filters for finding seeds when computing approximate string matching.

References

- [1] C. Allauzen, Crochemore, and M. Raffinot. Factor oracle: a new structure for pattern matching. In J. Pavelka, G. Tel, and M. Bartosek, editors, *Proceedings of SOFSEM'99, Theory and Practice of Informatics*, number 1725 in Lecture Notes in Computer Science, pages 291–306, Milovy, Czech Republic, 1999. Springer-Verlag, Berlin.
- [2] D. Cantone and S. Faro. Fast-search: A new efficient variant of the Boyer-Moore string matching algorithm. In K. Jansen, M. Margraf, M. Mastrolilli, and J. D. P. Rolim, editors, *Proceedings of the 2nd International Workshop on Experimental and Efficient Algorithms*, number 2647 in Lecture Notes in Computer Science, pages 47–58, Ascona, Switzerland, 2003. Springer-Verlag, Berlin.
- [3] C. Charras and T. Lecroq. *Handbook of exact string matching algorithms*. King's College London Publications, 2004.
- [4] M. Crochemore and T. Lecroq. A fast implementation of the Boyer–Moore string matching algorithm. Submitted.
- [5] K. Fredriksson and S. Grabowski. Practical and optimal string matching. In *Proceedings of SPIRE'2005*, LNCS 3772, pages 374–385, 2005.
- [6] J. Holub and B. Durian. Fast variants of bit parallel approach to suffix automata. Talk given in The second Haifa Annual International Stringology Research Workshop of the Israeli Science Foundation, 2005. <http://www.cri.haifa.ac.il/events/2005/string/presentations/Holub.pdf>.
- [7] A. Hume and D. M. Sunday. Fast string searching. *Software – Practice & Experience*, 21(11):1221–1248, 1991.
- [8] R.M. Karp and M.O. Rabin. Efficient randomized pattern-matching algorithms. *IBM J. Res. Dev.*, 31(2):249–260, 1987.
- [9] G. Navarro and M. Raffinot. Fast and flexible string matching by combining bit-parallelism and suffix automata. *ACM Journal of Experimental Algorithms*, 5:4, 2000.
- [10] G. Navarro and M. Raffinot. *Flexible Pattern Matching in Strings – Practical on-line search algorithms for texts and biological sequences*. Cambridge University Press, 2002.
- [11] S. S. Sheik, S. K. Aggarwal, A. Poddar, N. Balakrishnan, and K. Sekar. A fast pattern matching algorithm. *J. Chem. Inf. Comput. Sci.*, 44:1251–1256, 2004.
- [12] S. Wu and U. Manber. A fast algorithm for multi-pattern searching. Report TR-94-17, Department of Computer Science, University of Arizona, Tucson, AZ, 1994.
- [13] R. F. Zhu and T. Takaoka. On improving the average case of the Boyer–Moore string matching algorithm. *J. Inform. Process.*, 10(3):173–177, 1987.