A Method to Overcome Computer Word Size Limitation in Bit-parallel Pattern Matching

M. Oğuzhan Külekci

TÜBİTAK-UEKAE
National Research Institute of Electronics & Cryptology
41470 Gebze, Kocaeli, Turkey

kulekci@uekae.tubitak.gov.tr

Abstract. The performance of the pattern matching algorithms based on bit-parallelism degrades when the input pattern length exceeds the computer word size. Although several divide-and-conquer methods have been proposed to overcome that limitation, the resulting schemes are not that much efficient and hard to implement. This study introduces a new fast bit-parallel pattern matching algorithm that is capable of searching patterns of any length in a common bit-parallel fashion. The proposed bit-parallel length invariant matcher (BLIM) is compared with the Shift-Or and bit-parallel non-deterministic matching (BNDM) algorithms along with the standard Boyer-Moore and Sunday's quick search, which are known to be the very fast in general. Benchmarks have been conducted on natural language, DNA sequence, and binary alphabet random texts. Besides the length invariant architecture of the algorithm, experimental results indicate that on the average BLIM is 18%, 44%, and 6% faster than BNDM, which is accepted as one of the fastest algorithms of this genre, on natural language, DNA sequence and binary random texts respectively.

1 Introduction

Searching for exact or approximate occurrences of single or multiple patterns on text files is a fundamental issue that has been studied deeply during the last three decades. Besides its obvious usage in text editors, it gained great focus with the recent advances in genomics that require searching of long patterns with small alphabets. Additionally, network intrusion detection systems and anti-virus software also demand for high performance matchers.

The main idea behind the algorithms developed on off-line string searching may be grouped into two as automata theoretic approaches and sliding window techniques. In automata theoretic approaches beginning with the Aho-Corasick [1] finite state machine, most promising theoretical results have been obtained by using directed acyclic word graphs such as reverse factor and turbo reverse factor algorithms [2–4].

On the sliding window based approaches, the performance of searching patterns from right-to-left by Boyer-Moore (BM) [5] algorithm as oppose to the

left-to-right scanning of Knuth-Morris-Pratt (KMP) [6] had gained great success and many variants of the basic BM algorithm have been proposed, where the fastest ones are assumed to be the Sunday's quick search (QS) [7] and Horspool (HOR) [8] in general.

Baeza–Yates&Gonnet [9] introduced a new way of searching aiming to benefit from the computers intrinsic bit-parallelism for better speed-ups. Their Shift-Or (SO) algorithm preprocesses the input pattern in such a way that the searching on the text is performed with bitwise operations that are very fast in computer architecture. The downside of the SO was its inability to perform shifts while passing over the text. A similar bit-parallel algorithm supporting approximate matching was proposed by Wu&Manber [10], which was implemented as the *agrep* [11] file search utility software. Navarro&Raffinot [12] combined bit-parallelism with suffix automata (BNDM). By merging the shift mechanism of the sliding window techniques with bit-parallel implementation of the nondeterministic automata, BDNM is quite fast and also flexible especially on small alphabets.

The general problem in all bit parallel algorithms is the limitation defined on the length of the input pattern, which does not permit searching of strings longer than the computer word size effectively. While searching such long patterns, bit-parallel algorithms [12, 9, 13] divide the input pattern into pieces that are smaller than the computer word size and perform the search process on that smaller segments accordingly. In that case the performance of the original algorithm degrades considerably.

In this study, a different way of using bit parallelism is proposed that lets searching patterns independent of their lengths, thus, given name *bit-parallel length invariant matcher* (BLIM). As oppose to the previous bit-parallel algorithms that require the pattern length not to exceed the word size, and divide the string into smaller segments if that limitation is violated, BLIM defines a unique way of handling strings of any size. Proposed architecture supports using character classes and bounded gaps on the input patterns. It is also very suitable for matching computer word size number of multiple patterns of any length.

The performance of BLIM is compared with other bit-parallel algorithms SO and BNDM, with standard BM, QS, and HOR, and also with reverse and turbo reverse factor algorithms. It is observed that BLIM is the fastest on DNA sequence searching. It shows a better performance than other bit-parallel alternatives SO and BNDM on natural language text staying very competitive with BM and QS on long strings. Thus, besides overcoming the pattern length limitation of bit-parallelism, it brings a significant improvement in speed, especially in gene searching.

2 BLIM Algorithm

Let $T = t_0t_1t_2 \dots t_{n-1}$ be the text of length n, $P = p_0p_1p_2 \dots p_{m-1}$ be the pattern of length m that will be scanned on T, and W denotes computer word

size. The alphabet of the text will be shown by Σ, and the number of characters in the alphabet by $|\Sigma|$.

Given a pattern P, let's imagine an alignment matrix A which consist of W number of rows, where each row_i, $0 \leq i < W$, contains the pattern right shifted by i characters. Thus, A has $ws = W + m - 1$ columns. Note that, throughout the study ws value will be referred as window size from now on. A sample alignment matrix corresponding to pattern $P = abaab$ is shown in Table 1 assuming computer word size $W = 8$.

	0	1	2	3	4	5	6	7	8	9	10	11
0	a	b	a	a	b							
1		a	b	a	a	b						
2			a	b	a	a	b					
3				a	b	a	a	b				
4					a	b	a	a	b			
5						a	b	a	a	b		
6							a	b	a	a	b	
7								a	b	a	a	b

Table 1. The alignment matrix generated for the sample pattern $P = abaab$ with the assumption that computer word size $W = 8$.

The main idea of BLIM is to slide that ws length alignment matrix over the text, check if any possible placements of the input pattern exist in the current window via bitwise operations, and after completing the investigation of the current text portion $T[i \ldots i + ws - 1]$ right shift the window by an amount that is computed according to the immediate text character $T[i + ws]$ following the current window[1].

When the window is located on text $T[i \ldots i + ws - 1]$, BLIM visits the characters $T[i + j]$ $(0 \leq j < ws)$ in an order that was previously computed at the preprocessing stage. At each character visit, the possibility of any occurrences of the pattern is checked with a bitwise *and* operation by using a mask matrix that was again precomputed. Current window is slid right by the shift amount specified by the text character $T[i + ws]$ after the current investigation is over. Thus, a shift vector of alphabet size needs to be calculated at preprocessing also. In summary, at preprocessing stage BLIM calculates the mask matrix, the shift vector and decides on the scan order according to input pattern.

2.1 Preprocessing Stage I: The Mask Matrix

Mask matrix consists of alphabet size $|\Sigma|$ rows, and ws columns. Each $Mask[ch][pos]$, where $ch \in \Sigma$, and $0 \leq pos < ws$, is a bit vector of W bits

[1] Performing shift according to the immediate text character following the current window was first proposed by Sunday in quick search algorithm [7], which is one of the fastest algorithms in BM family.

as $b_{W-1}b_{W-2}\ldots b_1b_0$. The ith bit in $Mask[ch][pos]$ is set to 0, if observing that ch at position pos is not possible for the i character right shifted placement of the input pattern P. Otherwise, it is equal to 1, which indicates observing that ch at that pos does not violate the placement of the i character right shifted version of the pattern. The formal definition for the actual value of the b_i in $Mask[ch][pos]$ is as follows:

$$b_i = 0 \text{ , if } (0 \leq pos - i < m \text{ and } (ch \neq p_{pos-i})$$
$$b_i = 1 \text{ , otherwise}$$

An example mask matrix that is generated for the sample pattern $P = abaab$ is shown in Table 2 (in hexadecimal notation) assuming the alphabet of the text is $\Sigma = \{a, b, c, d\}$ with a computer word size $W = 8$.

	0	1	2	3	4	5	6	7	8	9	10	11
a	FF	FE	FD	FB	F6	ED	DB	B7	6F	DF	BF	7F
b	FE	FD	FA	F4	E9	D3	**A7**	4F	AF	3F	7F	FF
c	FE	FC	F8	F0	E0	C1	83	87	8F	1F	3F	7F
d	FE	FC	F8	F0	E0	C1	83	87	8F	1F	3F	7F

Table 2. The mask matrix generated for sample pattern $P = abaab$, assuming alphabet $\Sigma = \{a, b, c, d\}$, and computer word size $W = 8$.

For the sample pattern $P = abaab$, Table 3 depicts the calculation and meanings of bits in $Mask[b][6]$ explicitly. It is seen that right shift of the string by 0, 1, 2, 5, and 7 characters are appropriate, as specified by the corresponding bits in $Mask[b][6]$. These possible alignments can be viewed on rows 0, 1, 2, 5, and 7 of Table 1. Note that, the observation of b at $T[i+6]$ does not infer with the alignments beginning at $T[i]$, $T[i+1]$, and $T[i+7]$. Thus, the corresponding bits are set to 1.

Bit Index i	7	6	5	4	3	2	1	0
pos-i	-1	0	1	2	3	4	5	6
p_{pos-i}		a	b	a	a	b		
Bit Value	1	**0**	1	**0**	**0**	1	1	1

Table 3. Calculation of the bit vector $Mask[b][6] = A7$ for $P = abaab$.

2.2 Preprocessing Stage II: The Shift Vector

BLIM uses the shift mechanism proposed in quick search of Sunday [7]. That is; the immediate text character following the window determines the actual shift

amount. If that character is included in the pattern then $Shift[ch] = ws - k$, where $k = max\{i; p_i = ch\}$, else $Shift[ch] = ws + 1$. On the same example pattern $P = abaab$, remembering that $ws = 12$, and $\Sigma = \{a, b, c, d\}$, the shift values of the letters are given in Table 4.

Character	Shift Value
a	$9 = 12 - max(0, 2, 3)$
b	$8 = 12 - max(1, 4)$
c	$13 = 12 + 1$
d	$13 = 12 + 1$

Table 4. Computing the shift values of letters for sample pattern $P = abaab$.

If one encounters the character a at $T[i + 12]$ while the window is located at $T[i \dots i + 11]$, then the pattern $abaab$ may begin at $T[i + 9]$, $T[i + 10]$, and $T[i+12]$. Thus, the safe shift of a should be 9. Similarly, if $b = T[i+12]$, then the shift value is 8. The characters c and d do not occur in pattern P. Observation of those bad characters at $T[i + 12]$ directly yields the next attempt to begin from $T[i + 13]$, which means a shift value of 13.

Note that such a shift mechanism does not benefit from the characters investigated in the current window. This memoryless shift function may be replaced with a more sophisticated two-dimensional shift matrix such that $Shift[ch][pos]$ would indicate the safe shift amount when one observes ch at position pos. At each character visit, the shift value should be selected as the maximum of the current value and the value of the visited letter. Although theoretically less number of characters are visited in that case, the experiments showed that the time elapsed to update shift value at each character visit downgrades the speed. Thus, it is preferred to use the simple version instead.

2.3 Preprocessing Stage III: Scan Order

The characters of the ws length window should be visited in such an order that checking all possible alignments in the current window be accomplished with minimum number of character accesses. If we traverse the window from right-to-left or left-to-right, we have to perform unnecessary checks in case of a leftmost or rightmost occurrence of the pattern. Actually, the optimum solution requires calculating the next position after each character visit, which slows down the algorithm considerably.

As a simple and efficient way of computing the scan order, BLIM algorithm checks the characters at positions $m - i, 2m - i, \dots, km - i$, where $km - i < ws$, for $i = 1, 2, ..., m$ in order. As an example, assuming computer word size $W = 8$, the scan order of the sample pattern $P = abaab$ is $4, 9, 3, 8, 2, 7, 1, 6, 11, 0, 5, 10$. The main idea behind this ordering is to investigate the window in such a way that at each character access maximum number of alignments is checked.

2.4 Main Search Loop

BLIM algorithm is sketched in Figure 1 as a whole. It has a very simple main loop in which the current text portion is investigated very fast by using only the bitwise *and* operator. When the window is located at $T[i \dots i+ws-1]$, the $flag$ variable is first set to the corresponding mask value of the text character $T[i + ScanOrder[0]]$ at position $ScanOrder[0]$. Remember that $ScanOrder[0]$ is always equal to $m-1$. The next character to be visited is defined by $ScanOrder[1]$. The *and* operation is performed between the current flag and the mask of $T[i+ScanOrder[1]]$ defined for position $ScanOrder[1]$. The traversal of the characters continues till the $flag$ becomes 0 or all the characters are visited. If $flag$ becomes 0, this implies pattern does not exist on the window. Otherwise, one or more occurrences of the pattern are detected at the investigated text factor. In that case, the 1 bits of the $flag$ tells the positions of occurrences exactly, e.g., say if the $3rd$ bit is 1, then pattern is detected at $T[i+3]$. After completing the search on the current text factor, the window is slid right by the shift amount of the character at $T[i+ws]$.

```
Calculate Mask;
Calculate Shift;
Calculate ScanOrder;
Pad the text with m arbitrary characters;//to avoid segmentation fault
i=0;
ws=W+m-1;
while (i<n){
      flag = Mask[T[i+ScanOrder[0]]][ScanOrder[0]];
      for (j=1 ; flag & j<ws ; j++){
          flag &= MaskMatrix[T[i+ScanOrder[j]]][ScanOrder[j]];
      }
      if (flag){
          Check bits of the flag to find out detected items;
      }
      i += Shift[T[i+ws]];
}
```

Fig. 1. The BLIM algorithm

A sample run of the BLIM algorithm is depicted in Table 5 assuming that pattern `abaab` is to be searched on a text factor $T[0 \dots 11]$ = `ababaabaabab`. The corresponding mask values can be reached from Table 2. At the end of that sample run, it is observed that $flag$ is 00100100, which indicates pattern is detected at positions $T[2]$ and $T[5]$ conforming to bit indices (from least significant bit to most signicicant bit) that are 1. Note that after completion of the run, the window will be slid right according to the immediate text character following the window, e.g., the shift is 9, if $T[12]$ =`a`, or 13 if $T[12]$ =`c` or `d`.

	j	ScanOrder[j]	Mask[ch][pos]	flag
ababaabaabab	0	4	Mask[a][4] = 11110110	11110110
ababaabaabab	1	9	Mask[b][9] = 00111111	00110110
ababaabaabab	2	3	Mask[b][3] = 11110100	00110100
ababaabaabab	3	8	Mask[a][8] = 01101111	00100100
ababaabaabab	4	2	Mask[a][2] = 11111101	00100100
ababaabaabab	5	7	Mask[a][7] = 10110111	00100100
ababaabaabab	6	1	Mask[b][1] = 11111101	00100100
ababaabaabab	7	6	Mask[b][6] = 10100111	00100100
ababaabaabab	8	11	Mask[b][11] = 11111111	00100100
ababaabaabab	9	0	Mask[a][0] = 11111111	00100100
ababaabaabab	10	5	Mask[a][5] = 11101101	00100100
ababaabaabab	11	1	Mask[b][1] = 11111101	00100100

Table 5. A sample run of BLIM on text factor `ababaabaabab` for searching `abaab`.

3 Analysis

The preprocessing of BLIM has a quadratic time complexity due to the computation of the mask matrix that is of size $|\Sigma| \times ws$. The calculation of shift vector and the scan order array are linear with the length of the input pattern.

The best shift that the algorithm can perform is $ws + 1 = W + m$, and worst is $ws - m + 1 = W$. At each attempt the algorithm must check at least $\lceil \frac{ws}{m} \rceil$ positions on the text window $T[i \dots i + ws - 1]$ under investigation, and $ws - 1$ positions at most in case of a mismatch. Thus, the best-case complexity is $O(\lceil \frac{n}{ws+1} \rceil \times \lceil \frac{ws}{m} \rceil) \approx O(\frac{n}{m})$ assuming that best shift is performed at each attempt with minimal position checks. The worst case is when the window is slid over the text with the least shift value, and maximum character visits performed at each trial, which corresponds to a complexity of $O(\lceil \frac{n}{W} \rceil \times ws)$.

BLIM can handle character classes and bounded gaps in input patterns as well. If the pattern P contains more than one character at a position, then the corresponding cells of all those characters in the mask matrix will be filled accordingly. As an example, let's say we are searching for pattern `P=ab[ab]ab`, which means third character is either a or b. Then, while calculating the mask matrix, $P[2]$ will be treated not as a single letter, but a list of characters. When a *gap* is defined on the pattern, that means the list is composed of whole alphabet for that position.

Another advantage of BLIM is its ability to perform multiple pattern matching easily. Up to W number of patterns, whatever their sizes are, can be scanned simultaneously on the text. The main idea is to reserve each row of the alignment matrix for one of the input patterns in Table 1, and compute the mask matrix accordingly. Actually, the proposed architecture is capable of building the mask matrix according to any specified alignment matrix. Thus, it is able to handle character classes and bounded gaps on multi-patterns as well. Note that, the length invariant structure of BLIM lets to scan multi-patterns of any size again.

That is more important as the total size of multiple strings is more probable to exceed the computer word size, which restricts the other bit-parallel algorithms.

4 Experimental Results

BLIM is compared with bit-parallel non-deterministic matching (BNDM), reverse factor (BDM), turbo reverse factor (TBDM), standard Boyer-Moore (BM), Sunday's quick search (QS), Horspool (HOR), and Shift-Or (SO) algorithms on natural language text, DNA sequences and binary alphabet random texts. Note that the comparisons of the algorithms shown in figures 2, 3, and 4 do not include HOR, BDM and TBDM plots. That is because QS dominates HOR in general, and the performances of BDM and TBDM in practice are poor when compared with BLIM and BNDM.

The implementations of the tested algorithms are taken from the Charras&Lecroq's handbook of exact string matching algorithms [14][2]. Slight modifications performed on QS and HOR algorithms, which let them run faster.

All text files used in experiments are 30 MB in size. During the tests, the files are totally read into the memory, and each test pattern is scanned on the data 10 times by each of the tested algorithms. The minimum timing for each algorithm is recorded to minimize the effect of the system[3]. That testing scheme is repeated at least 5 times for each data type with different 30MB files on a PC with a Intel Pentium4 2.8 Ghz CPU and 2 GB memory running Windows XP operating system.[4]

Natural language test data is a collection of Turkish text from a daily newspaper. For each length of 2 to 50, 20 distinct strings are randomly selected from the text. Thus, a total of 980 test patterns are used in natural language searching experiments. The benchmark results are depicted in figure 2.

On natural language text, BLIM is on the average 18% faster than BNDM. For short patterns of length smaller than 5, SO behaves better than BLIM. Note that, although it is generally accepted that bit-parallel approaches do not give good results on natural language text, it is observed that for patterns longer than 12 characters, BLIM is very competitive with QS and BM.

The DNA sequence data used in experiments is extracted from the first human genome. Again, 20 random patterns are chosen for each length of 5 to 200 in steps of 5 making a total of 800 test patterns. Figure 3 shows the performance of the tested algorithms on DNA sequence searching.

On DNA sequence searching, BM and QS are not very effective especially on longer patterns as expected. The performance of BLIM is approximately 44%

[2] The codes can also be reached from the web page `http://www-igm.univ-mlv.fr/~lecroq/string`.

[3] Note that time elapsed to read the data file into memory is not included in recorded timings, which means the results show the pure performance of the tested algorithms.

[4] Same experiments are also performed on Intel Xeon 3.0 Ghz machine with 3 GB of memory running Windows XP SP2, and on a Linux machine with 2 GB memory. Similar performance lines are obtained for each algorithm in each trial.

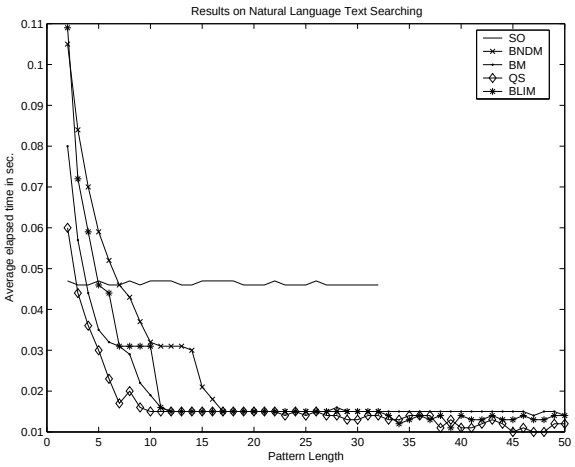

Fig. 2. Experimental results obtained on natural language text.

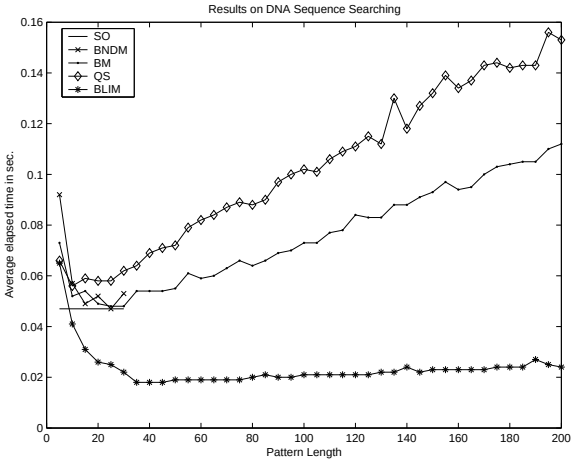

Fig. 3. Experimental results obtained on DNA sequence data.

better than BNDM. With that speed up, BLIM is by far the fastest technique on off-line searching of DNA sequences as BNDM is accepted as state-of-the-art off-line DNA matching in general.

Similar to DNA data, 20 random patterns with a binary alphabet $\Sigma = a, b$ are generated for each length of 5 to 200 again with a step of 5. That 800 binary alphabet strings are searched on randomly generated text files of 30 MB in size with the same alphabet.

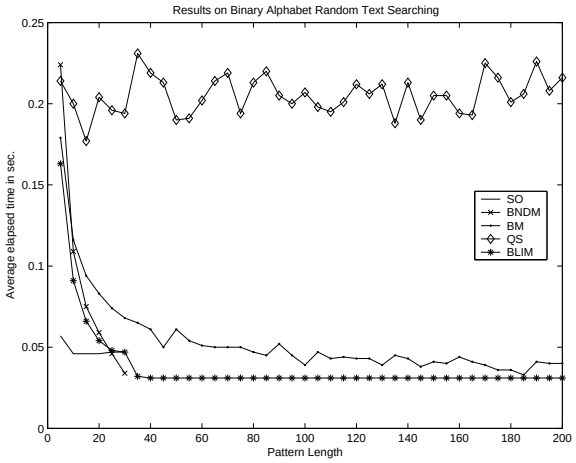

Fig. 4. Experimental results obtained on randomly generated binary alphabet text.

It is observed that BLIM is only 6% better than BNDM on binary alphabet random texts. SO is very effective on short binary sequence searching. On longer strings BLIM shows a better performance than BM. Note that the speed of QS on binary random data is very bad, when compared with BM. Actually, that is because on random binary data good-suffix shift mechanism of BM gets more important than bad character shift function. As QS does not include the good suffix shift mechanism, it degrades.

In summary, BLIM outperforms the other bit-parallel alternatives BNDM and SO on both natural language text and DNA sequences. Especially on DNA pattern matching it shows a very significant improvement being approximately 1.6 times faster than BNDM. On binary random texts, the leadership of SO continues, although for long patterns BLIM is the best candidate with its length-invariant structure. The over all average speed ratios of algorithms are depicted in Table 4.

Data Type	*Pattern Length*	***Average Time Ratio*** $\frac{SO}{BLIM}$	$\frac{BNDM}{BLIM}$	$\frac{BM}{BLIM}$	$\frac{QS}{BLIM}$
Natural Language Text	2-32	**1.83**	**1.18**	0.87	0.74
	33-50	NA	NA	**1.12**	0.90
DNA Sequence	5-30	**1.34**	**1.66**	**1.54**	**1.70**
	35-200	NA	NA	**3.75**	**5.18**
Random Text with Binary Alphabet	5-30	0.61	**1.16**	**1.30**	**2.52**
	35-200	NA	NA	**1.46**	**6.66**

Table 6. Average elapsed time ratio of BLIM according to other tested algorithms.Bold items represent the cases where BLIM is better.

5 Conclusion

Bit-parallel pattern matching algorithms require input pattern length to be less than or equal to computer word size. If pattern length exceeds that limitation, the string is divided into pieces that are smaller than the word size, and search process is performed on that segments accordingly. In that case the performance of the algorithms degrade.

This study presented a new way of using bit-parallelism that overcomes the computer word-size limitation and permits to search patterns of any length. The algorithm is capable of handling multiple patterns, and also using character classes and bounded gaps in patterns as well. A deep investigation of multi-pattern searching by BLIM is an on-going research, where initial results are promising.

In addition to its length invariant property, experimental results indicate that it is faster than previous bit-parallel approaches, SO and BNDM. Especially on DNA sequence searching, where up to now BNDM is known to be fastest in general, BLIM outperforms the BNDM with a 44% gain in speed. On long patterns it is 3.75 times faster than BM also. Thus, BLIM becomes a powerful tool in gene searching with both its significant improvement in speed and length-invariant structure.

References

1. Aho, A., Corasick, M.: Efficient string matching: An aid to bibliographic search. Communications of the ACM **18** (1975) 333–340
2. Crochemore, M., Czumaj, A., Gasieniec, L., Jarominek, S., Lecroq, T., Plandowski, W., Rytter, W.: Fast practical multi-pattern matching. Information Processing Letters **71** (1993) 107–113
3. Crochemore, M., Czumaj, A., Gasieniec, L., Jarominek, S., Lecroq, T., Plandowski, W., Rytter, W.: Speeding up two string matching algorithms. Algorithmica **12** (1994) 247–267
4. Crochemore, M., Rytter, W.: Text Algorithms. Oxford University Press (1994)
5. Boyer, R., Moore, J.: A fast string searching algorithm. Communications of the ACM **20** (1977) 762–772

6. Knuth, D., Morris, J., Pratt, V.: Fast pattern matching in strings. SIAM Journal of Computing **6** (1977) 323–350
7. Sunday, D.: A very fast substring search algorithm. Communications of the ACM **33** (1990) 132–142
8. Horspool, N.: Practical fast searching in strings. Software – Practice and Experience **10** (1980)
9. Baeza-Yates, R., Gonnet, G.: A new approach to text searching. Communications of the ACM **35** (1992) 74–82
10. Wu, S., Manber, U.: Fast text searching allowing errors. Communications of the ACM **35** (1992) 83–91
11. Wu, S., Manber, U.: Agrep – a fast approximate pattern-matching tool. In: USENIX Winter 1992 Technical Conference. (1992) 153–162
12. Navarro, G., Raffinot, M.: Fast and flexible string matching by combining bit-parallelism and suffix automata. ACM Journal of Experimental Algorithms **5** (2000) 1–36
13. Peltola, H., Tarhio, J.: Alternative algorithms for bit-parallel string matching. In: LNCS 2857, Proceedings of SPIRE'2003. (2003) 80–94
14. Charras, C., Lecroq, T.: Handbook of exact string matching algorithms. King's Collage Publications (2004)