Improved Approximate Multiple-Pattern String Matching using Consecutive N-Grams

Vidya Saikrishna
Monash University,Clayton Campus,
Wellington Road,Clayton,
Victoria 3800, Australia

Sid Ray
Monash University, Clayton campus,
Wellington Road,Clayton,
Victoria 3800, Australia

ABSTRACT
String matching is to find all the occurrences of a given pattern in a large text, the strings being sequence of characters drawn from finite alphabet set. Multiple-Pattern string matching problem involves detection of all the patterns of the Multiple-Pattern set in the text. Shift OR algorithm which we call as the Standard Shift OR algorithm uses the concept of Bit Parallelism to perform approximate string matching. The algorithm as the name suggests performs approximate string matching which means that it finds out some false matches besides detecting correct matches. In other words the algorithm behaves as a filter. In this paper a modification of the standard Shift OR is proposed to improve the filtering efficiency of the standard Shift OR algorithm using the consecutive N-Grams of the patterns of the multiple-pattern set. The proposed method reads N characters of the text at once as compared to a single character in the standard Shift OR algorithm. The number of false matches reduces besides increasing the speed of matching. Extensive experiments have been performed with the algorithm on text and pattern of variable size and the results are compared with the standard Shift OR algorithm.

Keywords
String Matching, Bit Parallelism, Shift OR String Matching, N-Grams, Automaton.

1. INTRODUCTION
N-Grams of a word or a pattern where N can be substituted by a small integer value can be overlapping or consecutive. For example consider a word "Patter", the overlapping 2-Grams of word are "pa", "at", "tt" and "er". If consecutive 2-Grams are considered then they would be "pa", "tt" and "er" [1].The proposed algorithm works on considering the consecutive N-Grams of the patterns. In the previous Shift OR method of approximate string matching if we have two patterns say "hello" and "world" then words like "herld", "wello" etc will also get recognized by the algorithm. The words recognized are termed as the false candidates. The potential matches generated needs to be verified [1]. The filtering efficiency of standard Shift OR filter is improved by considering N-Grams of the pattern. Along with the reduction of false candidates there has been a significant improvement in the speed of matching.

In the recent years bit parallelism has played an important role in string matching, because 'w' length of the pattern can be processed in parallel [5][6]. This is done by creating bit vectors of the pattern characters, and then the matching takes place with the help of bit operations in parallel. Transformation into bits results in faster results as they can be performed in parallel. Bit parallelism although performs better as compared to other non-bit parallel algorithms, but it imposes a limitation on the pattern size. Traditional algorithms solved using bit parallelism has a pattern size which is equal to the word length of the computer system [11][12]. Therefore increasing the word size of the system will make string matching algorithm work for patterns of larger size. Recent architecture makes use of 64 bit word size.

String Matching using bit parallelism can be viewed as being solved for single pattern and multiple-pattern. In single pattern string matching problem, there is a single pattern whose occurrence is to be reported in the text. In multiple pattern string matching problems, we are given a set of patterns whose occurrences are to be reported in the text. The multiple pattern string matching problems have more practical applications in real life which include text retrieval, symbol manipulation, computational biology, data mining and network security [1].

2. MULTIPLE PATTERN MATCHING USING BIT PARALLELISM
There are some notations used to describe the bit parallel algorithms. Exponentiation is used to indicate bit repetition, e.g., 1304 is interpreted as 1110000. Bitwise operators are used to indicate the bit operations such as "|" represents bitwise OR, "<<" moves the bits to the left and inserts zeros from right, e.g., 101011<<2=101100 and "~" complements the bits.

The Bit parallel approach can be extended to search for multiple patterns inside the text. The method also works for larger pattern sets. For large pattern sets, the bit parallel approach can be beneficial in terms of execution speed and memory requirement. The bit parallel approach for multipattern sets uses the Shift OR Algorithm for locating the patterns inside the text.

The method uses a bit vector B[c] which is initialised in a way such that the i-th bit is 0 if the character appears in any of the patterns in position i [1]. The automaton has a transition from state i to state i + 1 on character c if i-th bit in B[c] is 0. Another vector D is used which is initialized to all 1's. When the character c is read from the text D is updated as D = (D<<1) | B[c]. After the update, i-th bit in D is 0 if (i – 1)th bit was 0 (the previous state i − 1 was active) and ith bit is 0 in B[c] (there is a transition from state i − 1 to i on c)[1][11][12].

The assumption in this method is that all the patterns p1p2…..pr have equal size m and m≤w, where w is word size of the computer.

2.1 Algorithm Shift OR (text=$t_1 \ldots t_n$, patterns=p1... pk) [1][11][12]
The text characters are represented as $t_1 \ldots t_n$ and there are k patterns numbered from 1…k with each pattern consisting of

m characters. The notation pattern[i][j] is interpreted as the j-th character in pattern number i.

a. Initialization

m= pattern length, s=1, count=0, position=0, n=text length

b. Preprocessing

[Text[i]] ← 1^m
For j= 0…k do
For i= 0… m-1 do
B[pattern[j][i]] ← B[pattern[j][i] & ~ (s<<1)
end For
End For

c. Filtering

While pos< n do
D = D <<1 & 1^m
D=D | B [text [pos]]
if D> 1^{m-1} do pos← pos + 1
Else do count ← count +1
Report occurrence at position pos← pos-m +1
D ← 1^m
pos← pos +1
End else
End while

Example: Text= "hhello" Pattern = {"hello", "world"}
The Bit Vectors are set in the following manner [11][12].
B[h]=11110, B[e]=11101,B[l]=10011,B[o]=01101 , B[w]=11110, B[r]=11011, B[d]=01111
The Automaton recognizing the set of patterns is shown in fig. 1

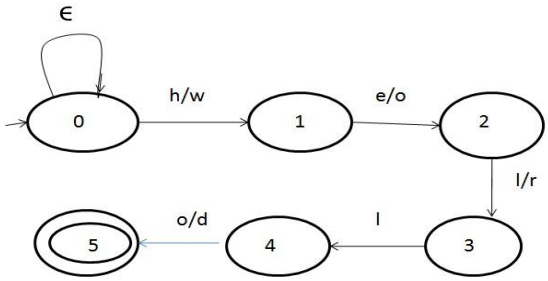

Fig 1: Non Deterministic Finite Automaton recognizing occurrence of character class pattern

The character class pattern is "[h,w],[e,o],[l/r],[l],[o/d]"
Table 1 shows bit parallel simulation of above automata.

Table 1 : Multiple pattern search example

1	Text = hhello D 11110 B[h] 11110 OR D 11110 D[0]=0 , so shift To next state	3.	Text = hhello D 11100 B[e] 11101 OR D 11101 D[1]=0, so shift to next State
2.	Text = hhello D 11100 B[h] 11110 OR	4.	Text = hhello D 11010 B[l] 10011 OR
	D 11110 D[1]=1 , so it remains in the same state		D 11011 D[2]=0, so shift to next state
5.	Text = hhello D 10110 B[l] 10011 OR D 10110 D[3]=0, so shift to next state	6.	Text = hhello D 01100 B[o] 01101 OR D 01101 D[4]=0, so shift to next State, which is the final state And the pattern is recognized.

The method used for multiple pattern searches is based on filtering approach. The filter method works in three phases. In the first phase, the pattern is preprocessed. In the second phase, matching takes place and in the third phase the matches generated by the method needs to be verified for more accurate results [12].

2.2 Analysis of Shift OR Algorithm [11][12]

- If the Text Length is assumed to be n, then the patterns are processed in O (n) time complexity.
- All the patterns are assumed to be of uniform length and less than or equal to the word size of the system.
- The method is a filter where the potential matches need to be verified.
- Number of False Matches for Shift OR Method

We assume there is a pattern set P= (p1, p2……pk) of k patterns. All the patterns are assumed to be having equal length m. We are calculating the false matches for the worst case, where all the patterns are assumed to be having distinct characters in all pattern positions. In this case:

(i) Total Number of correct Matches (CM) = K, as recognized by the Automaton.

(ii) Total number of matches recognized by the automaton (TM)= K^m

(iii) Total Number of false matches(FM1) = Total Matches – Total number of

Correct matches.

FM= K^m – K

(iv) In addition to these there are other false matches detected. Considering the following text and the pattern

Text: "heabcdello" and the pattern "hello".

The Shift OR method will detect one pattern match in the above text. Counting the false matches for such case.

FM2= m (∑*- k) where ∑ denotes the size of the input alphabet.

(v) Total False Matches(FM)= FM1 + FM2

FM= K^m – K + m (∑*-k)

= K^m – K + m∑* - m.k

= K{ K^{m-1} –m-1} + m∑*

3 . SHIFT OR CONSECUTIVE 2-GRAMS (SOC2G) PATTERN MATCHING

Let the text and the pattern be denoted as $t_1.....t_n$ and $p_1.....p_m$ respectively. All the patterns are assumed to have the same length. The size of the automaton is reduced to $\lfloor m/2 \rfloor + 1$ states as compared to m+1 states in previous shift OR method. This reduction in the size of the automaton is the main cause of improvement in the matching speed. Leena Salmela [1] emphasizes on achieving the multiple-pattern approximate string matching by the construction of overlapping N-Grams whereas the method used in this paper emphasizes on having a solution through consecutive N-Grams of the pattern with a detailed methodology to achieve it.

Considering two patterns "aabbcc" and "ccaabb" where the pattern length is 6 , the SOC2G method will result in the creation of a two dimensional array of size 256*256 bytes in memory. The increased speed is at the cost of increasing the memory requirement. The consecutive 2-Grams of the first pattern are "aa", "bb" and "cc" and that of the second pattern are "cc", "aa" and "bb". If the pattern length is m, then the number of bits in the bit vector will be $\lfloor m/2 \rfloor$. The 2-Grams of the pattern are treated as a single character and the bit vectors are set accordingly. The i^{th} bit is set to 0 if there is an occurrence of 2-Gram in the i^{th} position of the pattern and nonoccurrence denotes 1. For the example above the bit vectors of the 2-Grams of the pattern are set in the following manner:
B[aa] = 100, B[bb] =001 and B[cc] =010

Fig. 2.a shows the automaton recognizing the patterns in SOC2G algorithm.

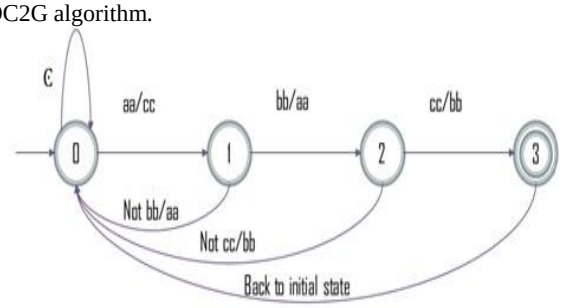

Fig 2.a: Automaton recognizing patterns "aabbcc" and "ccaabb" in SOC2G

As clear from the fig. 2.a the number of states would reduce to 4 as compared to 7 in the standard Shift OR method as shown in fig. 2.b.

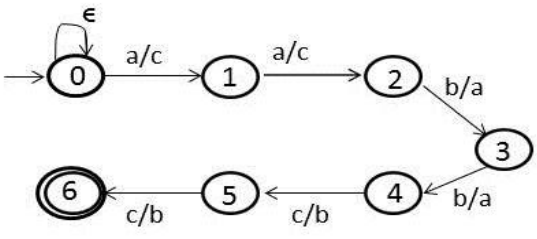

Fig 2.b: Automaton recognizing patterns "aabbcc" and "ccaabb" in Standard Shift OR

The automaton reads two characters of the text at a time and whenever a state is reached from where forward transition is not possible, this forces the automaton to reach to the initial state. This change in the automaton would reduce the generation of false candidates. For example the automaton would not read the group of characters "aaffgggggbbcc" in the text.

3.1 Shift OR Consecutive 2 Gram Algorithm (SOC2G)

The algorithm works in three phases. The first is the initialization phase which consists of initialization of the variables used in the algorithm. The second is the pre-processing phase which consists of setting the bit vectors. The third is the filtering phase which consists of matching the set of patterns against the text.

a. Initialization

In initialization phase different variables have been initialized which we have used in the algorithm. The variable m is initialized with the length of patterns (assuming that all the patterns have same length). Another variable count is used which denotes the number of matches. The variable count is initially set to zero before the filtering process begins. The variable S is used to maintain the state condition and S is initialized with 1.

b. Pre-Processing Phase

In pre-processing phase we are making bit vectors for each 2-Gram of the given patterns. This phase itself consists of number of steps as follow:

- In the first step all possible combination of alphabets, in given text file, of length two (as we are implementing 2-Grams) are initialized with all ones as presence of denotes nonoccurrence. For this we use array of size 256x256.
- In this step for each of the patterns, consecutive two characters are read and the bit vector is set in the manner that occurrence at i^{th} position inserts a 0 at i^{th} position of the bit vector. In case of odd length pattern we store the last alphabet of each pattern in an array called odd [] and then same procedure is applied on the rest even length pattern.
- Now for each 2-Gram of each pattern we are initializing a variable 't' as follows t = ~(s<<h) & int (pow (2,z) - 1), where h is initially zero for each pattern and is increased by one for each 2-Gram of a pattern.

  Bit vector of the 2-Gram = Previous Bit vector of the 2-Gram & t.

  In this way we have created bit vector of each 2-Gram.

c. Filtering Phase

In filtration phase we are making use of the bit vectors formed in previous phase to check the presence of the patterns in the given text. We don't search each pattern separately rather we search simultaneously for all patterns. This is done by making the 2-Grams of patterns equivalent. For example, 1^{st} 2-Gram of all the patterns are considered equivalent, 2^{nd} 2-Gram of all the patterns are considered equivalent and so on.

Further, we have made different stages equal to the number of 2-Grams in case of even length pattern and number of 2-Grams plus one in case of odd length pattern, with last state as final state in each case.

A integer array D [] is initialized with the bit vector of the 2-Gram encountered during the scanning of the text file from the beginning (position equal to zero). The bit of D, at position equal to the present state i.e. D [present state number], is checked. If that bit is zero, then we move to next state and check the bit vector of next 2-Gram(position in text file is increased by two), else position in text file is decreased by the state and we go back to initial state. A match is encountered when we reach the final state. At final state we increase the value of count and go back to initial state. This is repeated till the end of file.

Table 2 shows the working of the SOC2G algorithm

Table 2 Shift OR with consecutive 2-Grams

1.	Text= "sdaabbccfd" Reading 2 characters "sd" D 1 1 0 B[sd] 1 1 1 OR D 1 1 1 As D[0]=1 ,so it remains in the same state and the text is shifted one character to read the next 2 characters.
2.	Text = "sdaabbccfd" Reading next two characters "da" D 1 1 0 B[da] 1 1 1 OR D 1 1 1 As D[0]=1 ,so it remains in the same state and the text is shifted one character to read the next 2 characters.
3.	Text = "sdaabbccfd" Reading next two characters "aa" D 1 1 0 B[aa] 1 0 0 OR D 1 1 0 As D[0]=0 , the automaton moves to the next and the text is also shifted two characters to read next two characters.
4.	Text = "sdaabbccfd" Reading next two characters "bb" D 1 0 0 B[bb] 0 0 1 OR D 1 0 1 As D[1]=0 , the automaton moves to the next and the text is also shifted two characters to read next two characters.
5.	Text = "sdaabbccfd" Reading next two characters "bb" D 0 1 0 B[cc] 0 1 0 OR D 0 1 0 As D[2]=0 , the pattern is recognized , State Vector D gets reinitialized to 1 1 1. The text is shifted two characters.

4 .RESULT AND ANALYSIS

We have tested the algorithm 'Standard Shift OR' and our proposed algorithm 'Shift OR with consecutive 2-Grams (SOC2G)' on patterns of variable length, variable number of patterns and different text file of different sizes. On the basis of this, we have made following test cases. The result produced in each test case is shown below in tabular form and graphical form.

The experiment has been performed using the following experimental conditions:

Processor : Intel Core i7-260 M CPU, 2.80 GHz
RAM : 8 GB
System Type: 64 Bit Operating System
OS : Windows 7 Professional

4.1 Different text files of variable size:

In this case we have taken four random text files of size 40MB, 70MB, 100MB, and 130MB. And we have taken pattern set of 20 patterns of length 10 character.
Table 3 shows the result for the input set of standard Shift OR and SOC2G algorithm.

Table 3 Comparison of Shift OR and Shift OR 2 Grams

Size of text file(MB)	No. of matches (SOC2G)	No. of matches (Standard Shift OR)	Time (M Sec) (SOC2G)	Time (M Sec) (Standard Shift OR)
40	1607987	2009984	4.532	5.219
70	2375887	2639875	8.438	8.798
100	3138003	4482862	11.375	11.453
130	4325446	6179209	14.391	14.486

Fig.3 and Fig.4 compare the number of Matches and Time respectively

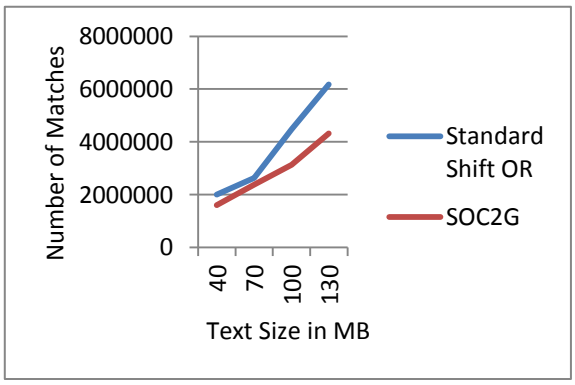

Fig 3: Comparing Number of matches

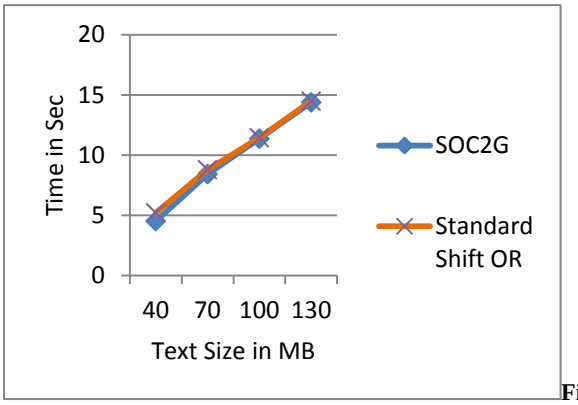

Fig 4: Comparing Time

4.2 Different Pattern Sets consisting of different numbers of patterns of same length

In this case we have taken four pattern set files consisting 5, 10, 15, and 20 patterns of 7 character each and we have taken a text file 130MB.
Table 4 shows the result for the input set of standard Shift OR and SOC2G algorithm.

Table 4 : Comparison of Standard Shift OR and SOC2G algorithm

No. of Patterns in the Pattern Set	No. of matches (Standard Shift OR)	No. of matches (SOC2G)	Time(M sec) (Standard Shift OR)	Time (M Sec) (SOC2G)
5	642500	578250	17.579	16.376
10	1284997	1027998	15.454	15.329
15	1798990	1439192	15.454	15.328
20	2569988	2184490	15.329	15.226

Fig. 5 and Fig. 6 compare the number of Matches and Time respectively

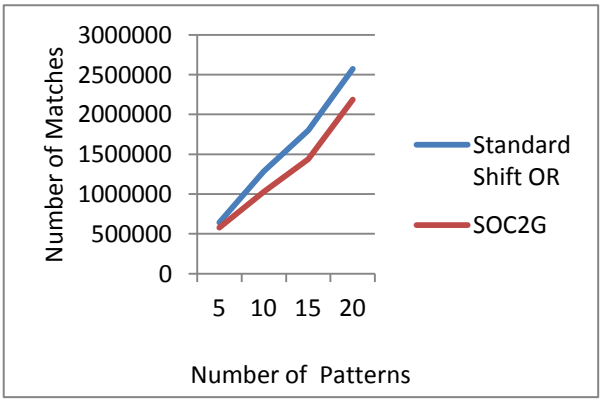

Fig 5: Comparing Number of matches

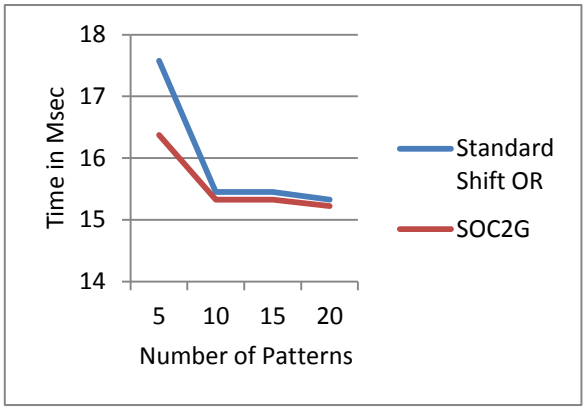

Fig 6: Comparing Time

4.3 Different pattern sets consisting patterns of different lengths and same number of patterns

In this case we have taken 3 pattern set files consisting 7, 10 and 20 letters in the pattern, consisting of 20 patterns each. And we have taken a text file 125MB.
Table 5 shows the result for the input set of standard Shift OR and SOC2G algorithm.

Table 5: Comparison of Standard Shift OR and SOC2G algorithm

Length of Pattern	No. of matches (Standard Shift OR)	No. of matches (SOC2G)	Time(M sec) (Standard Shift OR)	Time (M Sec) (SOC2G)
7	1336500	1336500	17.61	16.313
10	1336480	1336480	16.907	16.548
20	1336494	1336494	16.392	16.954

Fig.7 and Fig.8 compare the number of Matches and Time respectively

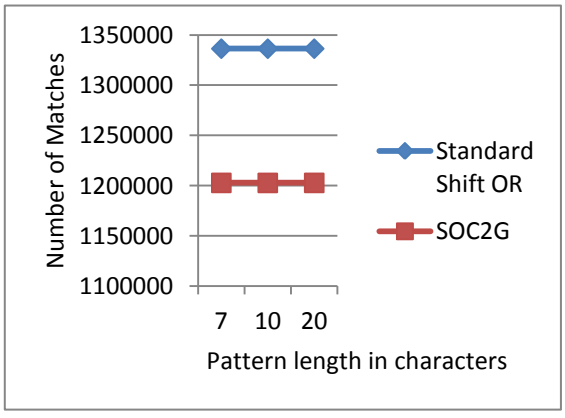

Fig 7: Comparing Number of matches

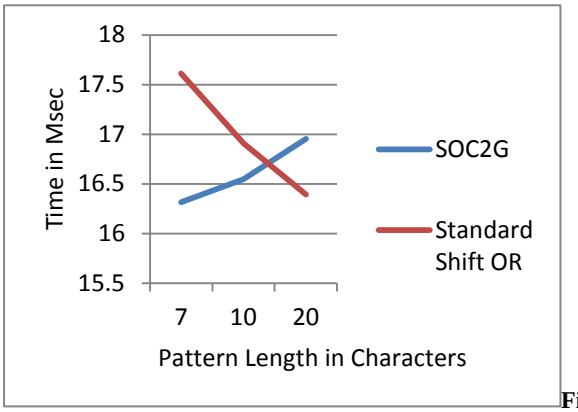

Fig 8: Comparing Time

5 . CONCLUSION & FUTURE WORK

We have presented efficient solutions for Shift or for multiple string matching algorithm using N-Grams and bit-parallelism. We have demonstrated that by using our algorithm the number of false matches has reduced considerably. The reduction in false matches is about 10%- 30% and there is a significant improvement as far as the time complexity is concerned. The speed is increased by about 10%. In certain texts where the probability of patterns matching with the text is high, the speed is enormously increased as two characters are read at a time.

In future we can further improve the speed of matching as well as the filtering efficiency by constructing 3-Grams of the pattern. The speed and efficiency is improved at the cost of having a memory capacity of 256x256x256 for constructing 3-Grams of the pattern.

6. REFERENCES

[1] Leena Salmela, J. Tarhio and J. Kytojoki, "Multiple Pattern String Matching with Q Grams", ACM Journal of Experimental Algorithmics , Vol. 11, Article No. 1.1, 2006.

[2] Rajesh Prasad, Suneeta Agarwal, Ishadutta Yadav, Bharat Singh "Efficient Bit-Parallel Multi-Patterns String Matching Algorithms for Limited Expression", Compute '10 , Proceedings of Third Annual ACM Bangalore Conference, Article No. 10, 2010.

[3] Heikki Hyyr¨o, Kimmo Fredriksson Gonzalo Navarro, "Increased Bit-Parallelism for Approximate and Multiple String Matching", ACM Journal of Experimental Algorithmics, Vol 10, 2006.

[4] Gonzalo Navarro and Mathieu Raffinot. "A Bit Parallel approach to Suffix Automata :Fast Extended String Matching", In M. Farach (editor), Proc. CPM'98, LNCS 1448. pp. 14-33, 1998.

[5] G. Navarro,M. Raffinot, "Fast and Flexible String Matching by combining Bit-Parallelism and Suffix Automata,ACM J. Experimental Algorithmics (JEA) Vol.5, Article No. 4 2000.

[6] M. Crochemore et al., "A Bit-Parallel Suffix Automaton approach for (δ, γ)-Matching in Music Retrieval", in Proc. 10th Internat. Symp. On String Processing and Information Retrieval (SPIRE'03), in: Lecture Notes in Computer. Sci., vol. 2857, pp. 211–223.

[7] R. Baeza-Yates, G. Gonnet, "A New Approach to Text Searching", Comm. ACM 35 (10) pp. 74–82, 1992.

[8] Hannu Peltola and Jorma Tarhio , Alternative Algorithms for Bit-Parallel String Matching, String Processing and Information Retrieval, LNCS 2857 pp. 80-93, 2003.

[9] AHO, A. AND CORASICK, M. 1975. "Efficient String Matching: an aid to Bibliographic Search", Communications of the ACM 18, 6, pp. 333–340, 1975.

[10] HYYR¨O, H. AND NAVARRO, G. 2002. "Faster Bit-Parallel Approximate String Matching", in Proc. 13th Combinatorial Pattern Matching (CPM '02). LNCS 2373. Berlin, Germany, Springer, New York.203–224, 2002.

[11] Vidya Saikrishna, Akhtar Rasool and Nilay Khare, "Spam Filtering through Multiple Pattern Bit Parallel String Matching Combining Shift AND & OR", International Journal of Computer Applications 61(5):40-45,. Published by Foundation of Computer Science, New York, USA, January 2013.

[12] Vidya Saikrishna, Akhtar Rasool and Nilay Khare, "Time Efficient String Matching Solution for Single and Multiple Pattern using Bit Parallelism", International Journal of Computer Applications 46(6):15-20,. Published by Foundation of Computer Science, New York, USA, May 2012.