Improved Approximate Multiple Pattern String Matching using Consecutive Q Grams of Pattern

Syed Danish Ali
Department of Computer Science & Engineering
All Saints' College of Technology

Zuber Farooqui
Department of Computer Science & Engineering
All Saints' College of Technology

ABSTRACT

String matching is to find all the occurrences of a given pattern in a large text both being sequence of characters drawn from finite alphabet set. This problem is fundamental in computer Science and is the basic need of many applications such as text retrieval, symbol manipulation, computational biology, data mining, and network security. Bit parallelism method is used for increasing the processing speed of String matching algorithm. Standard Shift OR algorithm is used to perform approximate string matching. The algorithm is a filter which finds out false matches besides detecting correct matches. To improve the efficiency of basic Shift OR algorithm by reducing the number of false matches that is detected along with the correct matches by the algorithm, proposed Shift OR with consecutive q grams has been implemented. In the algorithm instead of reading a single character at a time, it read q characters at once. Extensive experiments have been done with the algorithm and the results are compared with basic version of shift OR algorithms. The number of false matches also reduced considerably. The gain is due to the improved filtering efficiency caused by q-grams.

Keywords: Approximate String matching, Bit parallelism, Shift OR String Matching, q Grams**.**

1. INTRODUCTION

Q grams of 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 2grams of word are "pa", "at", "tt" and "er" . If consecutive 2grams are considered then they would be "pa", "tt" and "er"[1].The proposed algorithm works on considering the consecutive q grams of the patterns. In the previous Shift OR method of approximate string matching if there are two patterns say "hello" and "world" then words like "herld", " wello" etc will also get detected by the algorithm. The words recognized are termed as the false candidates. The potential matches generated needs to be verified. The filtering efficiency of the of the shift OR filter is improved by considering q grams of the pattern which results in the reduction of false candidates. [1][9].

In the recent years bit parallelism plays 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[13][14]. 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, a set of patterns are given whose occurrence's are to be reported in the text. The multiple pattern string matching problems are having more practical applications in real life.

2.MULTIPLE PATTERN MATCHING USING BIT PARALLELISM

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 [9]. The automaton has a transition from state i to state i + 1 on character c if ith 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)[9][13][14].

The assumption in this method is that all the patterns $p_1p_2.....p_r$ have equal size m and m<=w, where w is word size of the computer.

Algorithm Shift OR(text=t1…tn, patterns=p1,..,pk)[9][13][14]

1.Initialization

```
        m= pattern length, s=1, count=0,
        position=0,
```

2.**Preprocessing**

```
            [text[i]] <- $1^m$
             for j= 0…k  do
             for  i= 0… m-1 do   B[ptn[j][i]] <-
             B[ptn[j][i] & ~ (s<<1)
             end for
            end for
3.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 [13][14.

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 figure 2

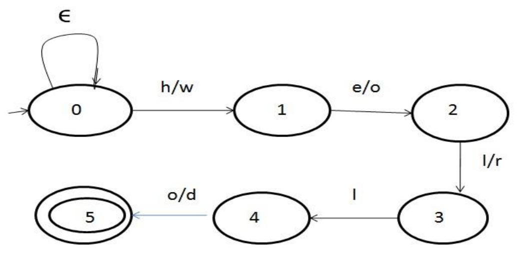

Figure 1 : NFA finding occurrence of character class pattern.

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

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 D 11110 D[1]=1 , so it remains in the same state	4.	Text = hhello D 11010 B[l] 10011 OR 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.

Table 1 : Multiple pattern search example

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 [14].

ANALYSIS OF Shift OR ALGORITHM[13][14]

- 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 needs to be verified.
- Number of False Matches for Shift OR Method

It is assume that there is a pattern set P=(p1,p2……pk) of K patterns. All the patterns are assumed to be having equal length m. False matches are calculated 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

$$FM= K^m – K + m(\sum^*-k)$$
$$= K^m – K + m\sum^* - mk$$
$$= K\{ K^{m-1} –m-1\} + m\sum^*$$

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 efficiency.

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 efficiency 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 ith bit is set to 0 if there is an occurrence of 2gram in the ith position of the pattern and non occurrence 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

The automaton recognizing the set of the patterns is shown in figure 1

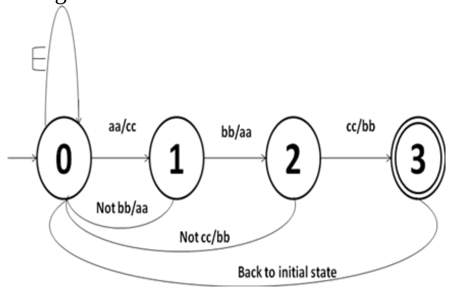

Figure 2 Automaton recognizing patterns "aabbcc" and "ccaabb"

As clear from the diagram the number of states would reduce to 4 as compared to 7 in the standard shift OR method. 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.

3.1(a) Initialization Phase

In initialization phase different variables have been initialized which is 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 .

3.1(b) Pre-Processing Phase

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

1. In the first step all possible combination of alphabets, in given text file, of length two (as implementing 2grams) are initialized with all ones as presence of denotes non occurrence. For this array of size 256*256 is used.
2. 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 ith position inserts a 0 at ith position of the bit vector. In case of odd length pattern the last alphabet of each pattern is stored in an array called odd[] and then same procedure is applied on the rest even length pattern.
3. Now for each two gram of each pattern 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 2gram = Previous Bit vector of the 2gram & t.

In this way bit vector is created of each two gram.

3.1(c) Filtering Phase

In filtration phase making the use of the bit vectors formed in previous phase to check the presence of the patterns in the given text. Each pattern is not searched separately rather than that search for all the patterns is done simultaneously. This is done by making the 2 grams of patterns equivalent. For example, 1st 2 gram of all the patterns are considered equivalent, 2nd 2gram of all the patterns are considered equivalent and so on.
Further, different stages have been made equal to the number of 2grams in case of even length pattern and number of 2grams 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 2gram encountering during the scanning of the text file form the starting(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 move to next state and check the bit vector of next 2gram(position in text file is increased by two), else position in text file is decreased by the state and then go back to initial state. A match is encountered when final state has been reached. At final state the values of count is increased and go back to initial state. This is repeated till the end of file.

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.

Table 2 Shift OR with 2 Grams

4. RESULT AND ANALYSIS

The algorithm 'standard shift or' and our algorithm 'shift or with 2grams' have been tested with different pattern length, number of patterns and different text file of different sizes. On the basis of this, the following test cases have been made. The result produced in each test case is shown below in tabular form, graphical form, and graphs comparing these two algorithms.
The experiment has been performed using the following experimental conditions:
Processor: Intel(R) Xeon (R) CPU E3-1240 V2 @ 3.40 GHz
RAM : 8.00 GB
System Type: 64 Bit Operating System
OS: Windows 7 Professional

Table 3 Comparison of Shift OR and Shift OR 2 Grams

No. of PATTERNS OF SAME LENGTH	No. of EXACT MATCHES	No. of MATCHES FOUND (in shift or)	No. OF MATCHES FOUND (in shift or with 2grams)	TIME (in Sec) (in shift or)	TIME (in Sec) (in shift or with 2grams)
5	20301552	21316602	20591553	0.250	0.297
10	20573061	21896644	20591553	0.234	0.281
15	20884570	23201734	23028276	0.234	0.67
20	30541111	38656846	34348418	0.234	0.577

4.1 Different text files of variable size:

In this case four random text files of size 40MB, 80MB, 120MB, and 160MB have been taken. And pattern set of 5 patterns of length 5 character have been taken.
The following table is showing the result for the input set For Shift OR algorithm and Shift OR with 2 grams.

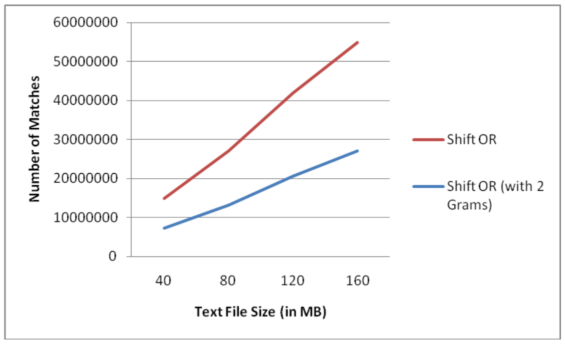

Figure 2: Comparing Number of matches

Here different text sizes are taken for same pattern set, and observed that if the size of the text is increased, the possibility of false matches is gradually reduced by upto10% using shift or (with 2 grams) when comparing it with standard shift or.

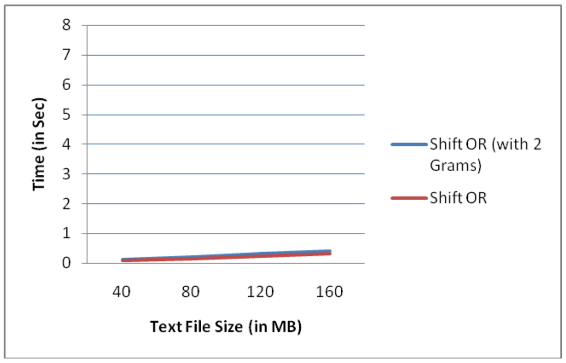

Figure 3: Comparing Time

4.2 Different patterns set consisting different number of patterns of same length:

Table 4 Comparison of Shift OR and Shift OR 2 Grams

SIZE OF TEXT (in MB)	No. of EXACT COUNT	No. OF MATCHES FOUND (in shift or with 2grams)	No. OF MATCHES FOUND (in shift or)	TIME (in Sec) (in shift or with 2grams)	TIME (in Sec) (in shift or)
40	7243484	7343484	7602054	0.109	0.094
80	13262149	13282149	13749829	0.203	0.141
120	20571552	20591553	21316602	0.312	0.234
160	26961000	26980000	27930000	0.406	0.312

In this case four pattern set files consisting 5, 10, 15, and 20 patterns are taken of 5 characters each and a text file of 120MB is taken.
The following table is showing the result for the input set for Standard Shift or algorithm and Shift
OR with 2 Grams:

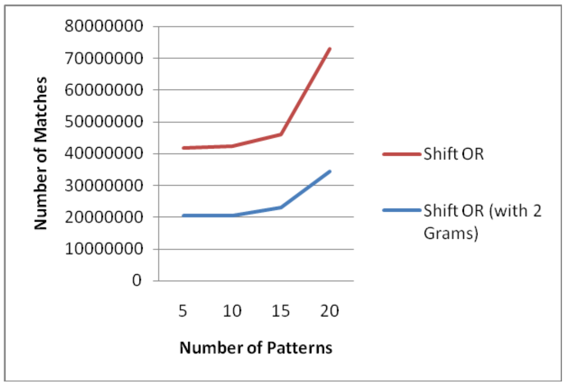

Figure 4: Comparing Number of Matches

For a same size text, here pattern set containing different number of patterns are taken, where each pattern set having same pattern length, It is found that as the number of patterns increases, the reduction of false matches obtained by shift or (with 2 grams) is gradually increased upto 20% as comparing it with the shift or method.

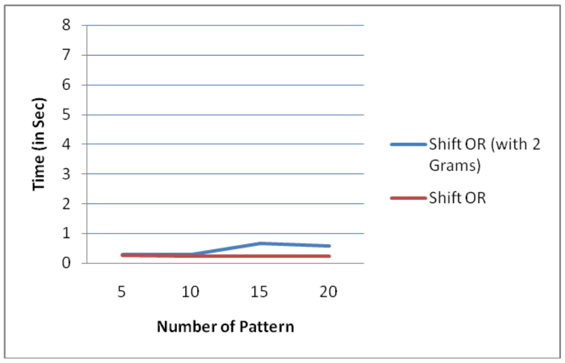

Figure 5: Comparing Time

4.3 Different patterns sets consisting patterns of different length and same number of patterns:

In this case 4 pattern set files are taken consisting 5, 10 15 and 20 characters in the pattern, consisting of 5 patterns each. And a text file of 120MB is taken.
The following table is showing the result for the input set for Standard Shift or algorithm:

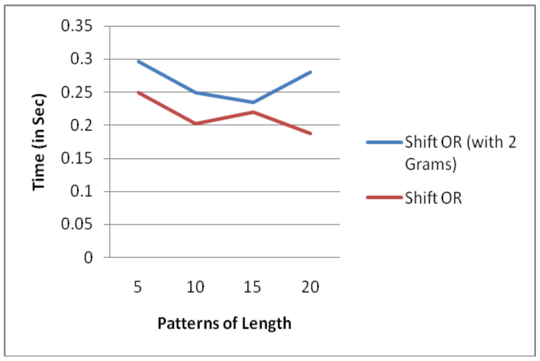

Figure 6: Comparing Time

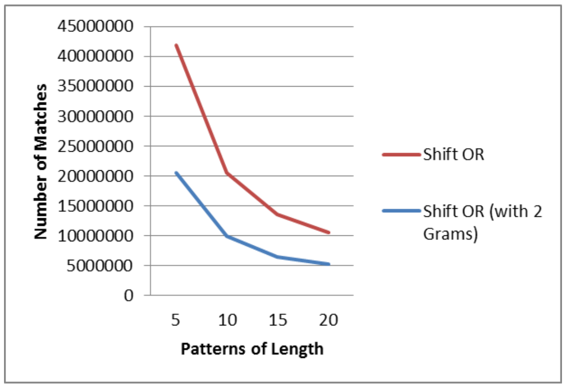

Figure 7: Comparing Number of Matches

For the same text file, here same number of patterns in different pattern sets is taken each of which containing different number of characters as pattern size, and analyzed that as the number of characters increases in pattern size, the possibility of reducing false matches in shift or (with 2 grams) is obtained upto 5% when it compares with standard shift or method.

Table 5 Comparison of Shift OR and Shift OR 2 Grams

PATTERNS OF LENGTH	No. of EXACT MATCHES	No. of MATCHES FOUND (in shift or)	No. OF MATCHES FOUND (in shift or with 2grams)	TIME (in Sec) (in shift or)	TIME (in -Sec) (in shift or with 2grams)
5	20591552	21316602	20591553	0.249	0.297
10	9721487	10585796	10005754	0.202	0.249
15	6051423	7105534	6525492	0.219	0.234
20	5026369	5365403	5220394	0.187	0.281

5. CONCLUSION & FUTURE WORK

An efficient solutions has been presented for Shift or for multiple string matching algorithm using q-grams and bit-parallelism. Our algorithm work in three phases: pre-processing, filtering and verifying. The algorithm works well even for very large pattern sets by using larger q-grams. It has been demonstrated that by using our algorithm the number of false matches has reduced considerably. The reduction in false matches is about 10% to 20% and as far as the time complexity is concerned, it may increase due to excessive hashing mechanism. It is also obvious that as the number of q grams will increase the memory and time will also increase. 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 the the filtering efficiency by constructing 3 grams of the pattern can be further improved. The efficiency is improved at the cost of having a memory capacity of 256*256*256 for constructing 3 grams of the pattern.

6.ACKNOWLEDGMENT

I would like to thanks to my guide Prof Zuber Farooqui who has give his precious time and guideline in order to complete this paper. This paper will never complete without the help of Department of Computer Science & Engineering, ASCT Bhopal.

7.REFERENCES

[1] Leena Salmela, J. Tarhio and J. Kytojoki, “Multiple Pattern String Matching with Q grams”, ACM Journal Name, Vol. V, No. N, Month 20YY

[2] Rajesh Prasad, Suneeta Agarwal, Ishadutta Yadav, Bharat Singh “Efficient Bit-Parallel Multi-Patterns String Matching Algorithms for Limited Expression”, Compute 2010 ACM

[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. Pages 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) 5 (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, 2003, pp. 211–223

[7] R. Baeza-Yates, G. Gonnet, A new approach to text searching, Comm. ACM 35 (10) (1992) 74–82.

[8] Hannu Peltola and Jorma Tarhio , Alternative Algorithms for Bit-Parallel String Matching, String Processing and Information Retrieval, 2003 - Springer

[9] Leena Salmela, J. Tarhio and J. Kytojoki “MultiPattern String Matching with Very Large Pattern Sets”, ACM Journal of Experimental Algorithmics, Volume 11, 2006.

[10] AHO, A. AND CORASICK, M. 1975. Efficient string matching: n aid to bibliographic search. Communications of the ACM 18, 6, 333–340.

[11] 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.

[12] NAVARRO, G. AND RAFFINOT, M. 2000. Fast and flexible string matching by combining bit parallelism and suffix automata. ACM Journal of Experimental Algorithmics (JEA). 5, 4.

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

[14] .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, May 2012. Published by Foundation of Computer Science, New York, USA