ISSN 2250-3153

Multiple Pattern String Matching Methodologies: A Comparative Analysis

Zeeshan Ahmed Khan[1], R.K Pateriya[2]

[1]Research Scholar
[2]Assistant Professor

Department of Computer Science & Engineering
Maulana Azad National Institute of Technology, Bhopal, 462051, INDIA

[1]zeeshan.khan12@gmail.com
[2]rk_pateriya@yahoo.co.in

Abstract**—**String matching algorithms in software applications like virus scanners (anti-virus) or intrusion detection systems is stressed for improving data security over the internet. String-matching techniques are used for sequence analysis, gene finding, evolutionary biology studies and analysis of protein expression. Other fields such as Music Technology, Computational Linguistics, Artificial Intelligence, Artificial Vision, have been using string matching algorithm as their integral part of theoretical and practical tools. There are various problems in string matching appeared as a result of such continuous, exhaustive use, which in turn were promptly solved by the computer scientists. The more practical solutions to the real world problems can be solved by the multiple pattern string matching algorithms. String Matching Algorithms like Aho-Corasick, Commentz-Walter, Bit parallel, Rabin-Karp, Wu-Manber etc. are to be focused in this paper. Aho-Corasick algorithm is based on finite state machines (automata). Commentz Walter algorithm is based on the idea of Knutt-Morris-Pratt and finite state machines. Bit parallel algorithm like shift-or makes use of wide machine words (CPU registers) to parallelize the work. Rabin-Karp uses hashing to find any one of a set of pattern strings in a text. Wu-Manber looking text in blocks instead of one by one character combining idea of Aho-Corasick and Boyer-Moore. Each algorithm has certain advantages and disadvantages. This paper presents the comparative analysis of various multiple pattern string matching algorithms. A comparison of Aho-Corasick, Commentz-Walter, Bit-Parallel(Shift-OR), Rabin-Karp, Wu-Manber etc. type of string matching algorithms is presented on different parameters.

Index Terms**— String matching, Aho-Corasick, Commentz Walter, Bit parallel, Rabin-Karp, Wu-Manber, FSM.**

I. Introduction

String matching is a technique to find out pattern from given text. Let Σ be an alphabet. Elements of Σ are called symbols or characters. For example, if Σ = {a,b}, then abab is a string over Σ. String ab, ba, aba, bab are set of patterns over given string. The patterns strings are denoted by P[1....m]. The text string is denoted by T [1...n]. If P occurs with shift s in T, then we call s a valid shift; otherwise, we call s an invalid shift. The string matching problem is the problem of finding all valid shifts with which a given pattern P occurs in a given text T Figure 1 shows this definition[1].

These problems find applications in information retrieval systems like search engines, bioinformatics, computer security, and DNA sequence analysis.

"When I got to the bottom of the stairs, I saw that the stairs were high and steep. I took a deep breath and began to climb the stairs. When I reached the top of the stairs I looked around me."

stairs ?

Input Text String and Pattern

"When I got to the bottom of the stairs, I saw that the stairs were high and steep. I took a deep breath and began to climb the stairs. When I reached the top of the stairs I looked around me."

Output

Fig 1. Basic Defination

Two examples of multiple-pattern matching problems:

English:

Text: CPM_annual_conference_announce

Set of Symbols: { *announce*, *annual*, *annually* }

DNA:

Text: AGATACGATATATAC

Set of Symbols: { *ATATATA*, *TATAT*, *ACGATAT* }

String matching algorithms can be categorized basically in two types exact and approximate string matching algorithms. Exact string matching algorithms are further divided into single and multiple pattern string matching. Each category can be applied to various areas of application. Multiple pattern matching algorithms have more practical and realistic applications like

DNA sequencing[10][11], intrusion detection/prevention system (IDS/IPS)[13][14][15], data mining, search engines[12], detecting plagiarism[8][9], Music Technology[7] etc.Multiple pattern string matching algorithms are mainly discussed in this paper.

String matching algorithms are also used in Intrusion Detection Systems (IDSs) that have become widely recognized as powerful tools for identifying, deterring and deflecting malicious attacks over the network. Essential to almost every intrusion detection system is the ability to search through packets and identify content that matches known attacks. Space and time efficient string matching algorithms are therefore important for identifying these packets at line rate[4].

The paper is organized as follows. Section II survey on the most significant algorithms for multiple string matching algorithms like Aho-Corasick, Commentz Walter, Bit-Parallel(Shift-OR), Rabin-Karp, Wu Manber. Section III comparative study of various algorithmsis described. Finally Section IV concludes comparative survey by putting strong points in favour of some algorithm in particular case of usage.

II. VARIOUS MULTI-PATTERN STRING MATCHING ALGORITHMS

A. Aho-Corasick Multi-Pattern Algorithm

The Aho-Corasick algorithm[1] (AC Algorithm) was proposed in 1975 and remains, to this day, one of the most effective pattern matching algorithms when matching patterns sets. Initially, the AC algorithm combines all the patterns in a set into a syntax tree which is then converted into a non-deterministic automaton (NFA) and, finally, into a deterministic automaton (DFA) as shown in Figure 2. The resulting FSM is then used to process the text one character at a time, performing one state transition for every text character. Whenever the FSM reaches designated "final" states that correspond to the identification of a pattern match as in Figure 3

The pseudocode for the matching phase of the algorithm is given below:

```
procedureAC(y,n,q0)
        /*Input:
        .y←array ofnbytesrepresentingthetextinput
        .n←integerrepresentingthetextlength
        .q0 ←initialstate
        */
    state←q0
        fori=1→ndo     //Matching
            whileg(state,y[i])= faildo
            //while g(state, y[i]) is undefined
              state←f(state)
          end while
          state←g(state,y[i])
          ifo(state)=∅then
              outputi
    //This an accepting state, i.e. state ∈ A
          end if
      end for
end procedure
```

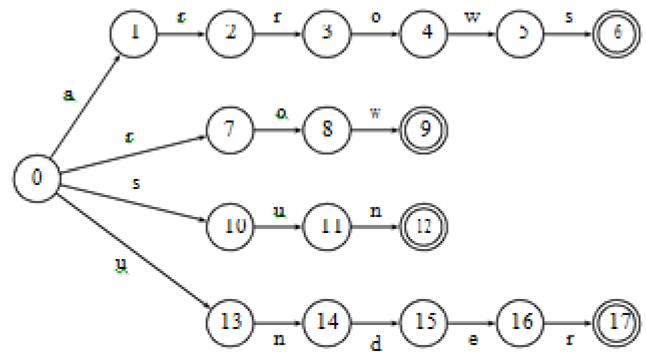

Fig.2. Finite State Automata for the keywords {arrows, row, sun, under

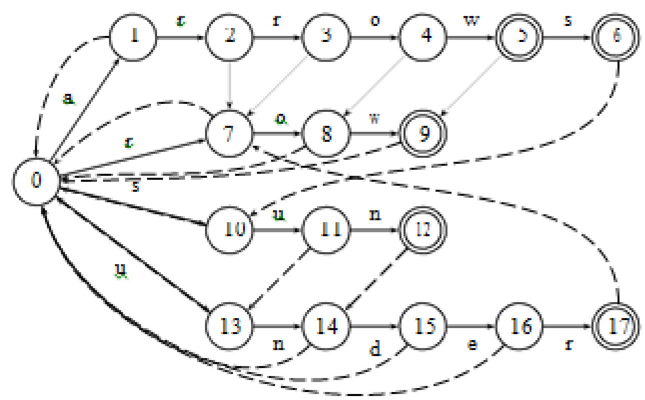

Fig.3. Failure Function for the above Automata

State(i)	1	2	3	4	5	6	7	8
Failure(i)	0	7	7	8	9	10	0	0

State(i)	9	10	11	12	13	14	15	16	17
Failure(i)	0	0	13	14	0	0	0	0	0

TABLE 1
FAILURE FUNCTION FOR ALL THE STATES OF THE ABOVE AUTOMATA

(i)	Output(i)
5	{row}
6	{arrows}
9	{row}
12	{sun}
17	{under}

//usethefailurefunction
TABLE 2
OUTPUT FUNCTION FOR THE ABOVE AUTOMATA

The AC algorithm has the significant advantage that every text character is examined only once, i.e. the lookup cost is O(N) where N the length of the text, regardless of the number of patterns or their length. A major disadvantage of the AC algorithm is the high memory cost required to store the transition rules ofthe underlying deterministic finite automaton.

B. Commentz Walter Multi-Pattern Algorithm

The popular GNU fgrep utility uses the Commentz-Walter algorithm for multiple string search[2].Commentz-Walter algorithm combines the Boyer-Moore technique with the Aho-Corasick algorithm. In preprocessing stage, differing from Aho-Corasick algorithm, Commentz-Walter algorithm constructs a converse state machine from the patterns to be matched. Each pattern to be matched adds states to the machine, starting from right side and going to the first character of the pattern, and combining the same node.

In searching stage, Commentz-Walter algorithm uses theidea of Boyer-Moore algorithm. The length of matching window is the minimum pattern length. In matching window, Commentz-Walter scans the characters of the pattern from right to left beginning with the rightmost one. In case of a mismatch (or a complete match of the whole pattern) it uses a precomputed shift table (CWShift[]) to shift the window to the right.

For pattern set {search, ear, arch, chart}, Figure 4 shows the Commentz-Walter state machine and the goto function. Table 3 shows the output function and Table 4 shows the shift distance.

If the text string is "strcmatecadnsearchof", Figure 5 shows the searching process. Forseeingclearly,we drawthe keycharactersinredandgreen,andwiththesameintention in Figure4and Figure 5.

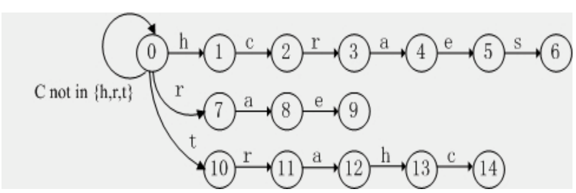

Fig.4.Commentz-Walterstate machineandgotofunction

nodes	4	6	9	14
out	{arch}	{search}	{ear}	{chart}

TABLE 3.
COMMENTZ-WALTER OUTPUT FUNCTION

	a	c	e	h	r	others
0	1	1	2	3	1	3
7	1	1	1	2	1	3
8	1	1	1	1	1	2
9	2	2	2	2	2	2
others	3	3	3	3	3	3

TABLE 4.
CWSHIFT TABLE. CWSHIFT[0A]=1, CWSHIFT[0E]=2, …, CWSHIFT[9R]=2, ETC.

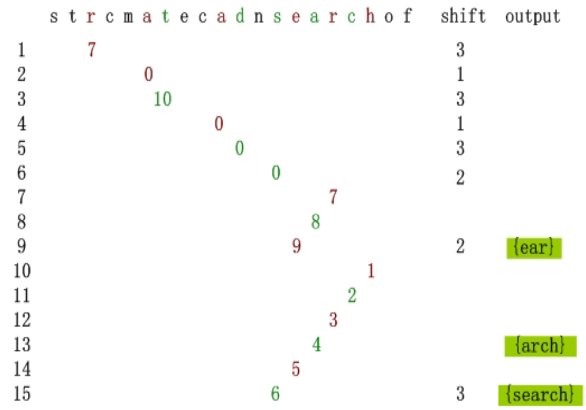

Fig.5.Commentz-Waltersearchingprocess.

In Figure 3, the length of matching window is 3. On step 1, when input is 'r', the state is 7, when input is 't', the state is 0, CWShift[7t]=3, the text shift 3 characters. On step 2, when input is 'a', the state is 0, CWShift[0a]=1, the text shift 1 character. On step 7, when input is 'r', the state is 7, when inputis'a',thestateis 8,wheninput is 'r', thestateis 9, the outputis 'ear', wheninputis 's',thestateis0, CWShift[9s]=2, the textshift 2 characters.

A multiple string matching algorithm that compares from the end of the pattern, like Boyer-Moore, using a finite state machine, like Aho-Corasick.In computer science, the Commentz-Walter algorithm is a string searching algorithm invented by Beate Commentz-Walter. Like the Aho–Corasick string matching algorithm, it can search for multiple patterns at once. It combines ideas from Aho–Corasick with the fast matching of the Boyer–Moore string search algorithm. For a text of length n and maximum pattern length of L, its worst-case running time is O(nL), though the average case is often much better.

The pseudocode for the matching phase of the algorithm is given below:

```
procedure CW(y, n, m, p, root)
        /*. Input:
        . y ← array  of n bytes representing the text input
        . n ← integer representing the text length
        . m ← array  of keyword lengths
        . p ← number  of keywords
        . root ← root node of the trie
        */
    v ← root     //The current node
    i ←min{m[0], m[1], ..., m[p − 1]}
    //i points to the current position in y
    j ← 0      //j indicates depth of the current node v
    while i ≤ n do          //Matching
    while v has child v0  labeled y[i − j] do
    v ← v0
    j ← j + 1
        if out(v) = ∅ then
                output i – j
               //Path from v to root matches y[i − j] to y[i]
        end  if
```

ISSN 2250-3153

```
        end  while    //Shifting
  i ← i+ min { shif t2(v), max { shif t1(v), char(y[i − j]) − j − 1 } }
        j ← 0
        end  while
end  procedure
```

C. Wu-Manber (WM) Multi-Pattern Algorithm

Wu-Manberalgorithmisahighperformancemulti-pattern matchingalgorithm basedonBoyer-Moorealgorithm.Itonly usesthe bad-character shift, andconsiderthecharacters from thetextinblocksof sizeBinsteadof onebyone,expands the effectofBad-charactershift.Wu-Manberalgorithm alsouses thehashingtabletoindexthe patternsintheactualmatching phase,thussavingalotoftime.Thebestperformanceof Wu-Manberalgorithm is*O(B n/ m)*.Therunningtimeofthe Wu-Manberalgorithm doesnotincreaseinproportiontothe sizeofthe patternset.The performanceoftheWu-manberis dependent ontheminimum lengthofthe patterns.Inpractice Wu-Manberalgorithm isoneofthe bestperformancein average case [4].

Inpreprocessingstage,Wu-Manberalgorithm buildthree tables,aSHIFTtable,a HASH table,and aPREFIXtable. TheSHIFTtableissimilar, but not exactlythesame,tothe regularshifttableinaBoyer-Mooretypealgorithm.It is used todeterminehowmanycharactersinthetextcan beshifted (skipped)whenthetextisscanned. The HASHandPREFIX tablesareusedwhentheshiftvalueis 0.Theyareusedto determinewhich patternis acandidatefor thematchandto verifythematch. For patternset {search,hear,arch,chart},Table6shows the SHIFT table 5 and Table6 shows theHASH tablefor B=2.

BC	ar	ch	ea	ha	he	rc	se	others
shift	0	0	1	1	2	1	2	3

TABLE 5
WU-MANBER SHIFT TABLE

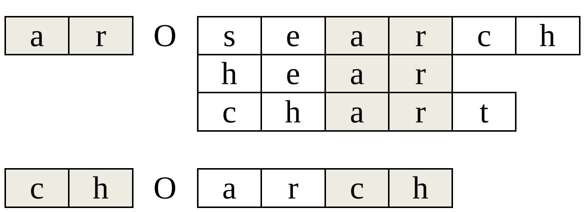

TABLE 6
WU-MANBER HASH TABLE

If the text string is "strcmatecadnsearchof", Figure 6 showsthe searchingprocess.

Fig.6.Wu-Manbersearchingprocess.

Wu-Manber combines the advantages of both Aho-Corasick and can index the document once as well as skip over words in a jump table. Wu-Manber algorithm performs well under scenarios with small or medium number of patterns, but it does not perform well under scenarios with large number of patterns, because the likelihood of shifting the text sliding window is decreased quickly with the increase of the number of patterns.

D. Rabin-Karp Multi-Pattern Algorithm

In computer science, the Rabin–Karp algorithm is a string searching algorithm created by Michael O. Rabin and Richard M. Karp in 1987 that uses hashing to find any one of a set of pattern strings in a text.[5]The Rabin-Karp string searching algorithm calculates a hash value for the pattern, and for each M-character subsequence of text to be compared. If the hash values are unequal, the algorithm will calculate the hash value for next M-character sequence. If the hash values are equal, the algorithm will do a Brute Force comparison between the pattern and the M-character sequence. In this way, there is only one comparison per text subsequence, and Brute Force is only needed when hash values match.

Consider an M-character sequence as an M-digit number in base b, where b is the number of letters in the alphabet. The text subsequence t[i .. i+M-1] is mapped to the number.

$$x(i) = t[i].b^{M-1} + t[i+1].b^{M-2} +...+ t[i+M-1]$$

Furthermore, given x(i) we can compute x(i+1) for the next subsequence t[i+1 .. i+M] in constant time, as follows:

$$x(i+1) = t[i+1].b^{M-1} + t[i+2].b^{M-2} +...+ t[i+M]$$

$x(i+1) = x(i).b$ //Shift left one digit
$- t[i].b^M$ // Subtract leftmost digit
$+ t[i+M]$ //Add new rightmost digit

In this way, we never explicitly compute a new value. We simply adjust the existing value as we move over one character.

Let's say that our alphabet consists of 10 letters. Our alphabet = a, b, c, d, e, f, g, h, i, j Let's say that "a" corresponds to 1, "b" corresponds to 2 and so on. The hash value for string "cah" would be:-

3*100 + 1*10 + 8*1 = 318

If M is large, then the resulting value (~bM) will be enormous. For this reason, we hash the value by taking it mod a prime number q. The mod function is particularly useful in this case due to several of its inherent properties:

- [(x mod q) + (y mod q)] mod q = (x+y) mod q
- (x mod q) mod q = x mod q

For these reasons:

$h(i) = ((t[i].b^{M-1} \bmod q) + (t[i+1].b^{M-2} \bmod q) + ... + (t[i+M-1] \bmod q)) \bmod q$

h(i+1) =(h(i).b mod q//Shift left one digit
-t[i]. b^M mod q//Subtract leftmost digit
+t[i+M] mod q)//Add new rightmost digit
mod q

For text of length n and p patterns of combined length m, its average and best case running time is O(n+m) in space O(p), but its worst-case time is O(nm). In contrast, the Aho–Corasick string matching algorithm has asymptotic worst-time complexity O(n+m) in space O(m).

```
Procedure  RabinKarpSet(string s[1..n], set of string subs, m):
 set hsubs := emptySet
  for each sub in subs
      insert hash(sub[1..m]) into hsubs
hs := hash(s[1..m])
for i from 1 to n-m+1
    if hs ∈ hsubs and s[i..i+m-1] ∈ subs
         return i
      hs := hash(s[i+1..i+m])
  return not found
```

Rabin-Karp is good for plagiarism[8][9], because it can deal with multiple pattern matching. It is not faster than brute force matching in theory, but in practice its complexity is O(n+m). With a good hashing function it can be quite effective and it's easy to implement.

E. Bit-Parallel(Shift OR) Multi-Pattern Algorithm

Bit-parallelism is a technique introduced by Baeza-Yates and Gonnet in which takes advantage of the intrinsic parallelism of the bit operations inside a computer word, allowing to cut down the number of operations that an algorithmperforms by a factor up to w, where w is the number of bits in the computer word.Bit-parallelism is particularly suitable for the efficient simulation of nondeterministic (suffix) automata.[6] in Figure 7 and Table 7 and 8.

Text= "hhello"
Pattern = { "hello", "world"}

The Bit Vectors are set in the following manner.
B[h]=11110, B[e]=11101,B[l]=10011,B[o]=01101 ,
B[w]=11110, B[r]=11011, B[d]=01111

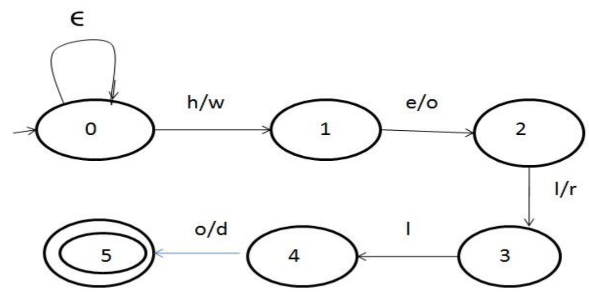

Fig 7. The Automaton recognizing the set of patterns is shown

1	Text = hhello D 11110 B[h] 11110 OR D 11110 D[0]=0 , so shift To next state	2	Text = hhello D 11100 B[e] 11101 OR D 11101 D[1]=0, so shift to next state
3	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 nextstate	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 7
EXAMPLE SHIFT-OR ALGORITHM

The pseudocode for the matching phase of the algorithm is given below:

procedureShift-Or(P,m,T,n)
for all $i \in \Sigma$ **do**
$B[i] = 1^W$;
for $i = 0$ to $m-1$ **do**
$B[P[i]] = B[P[i]] \,\&\, 1^{m-i-1}01^i$;
$D = 1^W$; $mm = 0^{W-m}10^{m-1}$; $i = 0$;
while $i < n$ **do**
$D = (D << 1) | B[T[i]]$;
if $((D \& mm)) \neq mm$ **then**
Pattern detected beginning at $T[i]$;

	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 8
THE ALIGNMENT MATRIX GENERATED FOR THE SAMPLE PATTERN
P = {ABAAB}WITH THE ASSUMPTION THAT COMPUTER WORD SIZE W = 8

Bit parallel Algorithms like shift-Or have advantage that it works at bit level of the string matching problems that results will be obtained faster compared with other techniques of string matching. Every coin has two side so as this algorithm, Patterns are required to be no longer than the computer word size. It means that maximum length of the pattern must not exceed 32 bits in case of 32 bit machine or 64 bits in case of 64 bit machine.

III. COMPARATIVE ANALYSIS OF SELECTED MULTIPLE PATTERN STRING MATCHING ALGORITHMS

In this paper, we analysed selected mulitiple pattern string matching algorithms on the basis of time complexity, search type, key ideas and approach parameters. Each algorithm has certain advantages and disadvantages.

The main advantage of the Aho-Corasick algorithm is that it runs in linear time to the input patters regardless of the number of patterns. However, the main disadvantage lies in devising a practical implementation due to the large memory needed to store the FSM. So one of the primary area of focus in the IDS area is in devising a performance and area efficient strategy for the Aho-Corasick algorithm.

Commentz Walter algorithm combines Aho-Corasick with the Boyer-Moore single pattern matching algorithm [3], which achieves sub-linear running time by skipping characters in the input text according to the “bad character” and “good suffix” heuristics.

In Wu-Manber Algorithm when the length of patterns is short, the search time is long. As the length of the patterns increasing, the search time shortens obviously. It shows that the search time of double-character patterns is a little longer than the single-character patterns. That is because these two kinds of patterns have the same shift distance “1”. When the shift distance is same, the matching time of double-characters is longer than the single-characters.

The Rabin-Karp algorithm achieves an average case running time of O(m+n) by using hashing. The worst case running time is still O(mn) however. The Rabin-Karp algorithm exploits the fact that if two strings are equal, their hash values are also equal. Rabin-Karp is simple and can be easily extended to two-dimensional pattern matching.

The bit-parallel algorithms are more efficient that other string matching algorithms for small and long patterns respectively. Their running time decreases as the pattern length increases and they produce similar running times in all cases with the exception of the binary alphabet.

Algorithms		Aho-Corasick	Rabin-Karp	Bit-Parallel (Shift OR)	Commentz Walter	Wu-Manber
1	Time Complexity	Linear	Linear	Linear	Linear	Sub-linear
2	Search Type	Prefix	Prefix	Prefix	Prefix	Suffix
3	Key Ideas	Finite automat on that tracks the partial prefix match.	Compare the text and the patterns from their hash functions	Bit-parallelism and q-gram for prefix matching.	it uses is to shift by as much as determined by the longest proper prefix of the pattern	Determine the shift distance from a block of characters in the suffix of the search window.
4	Approach	Automat on-based	Hashing-based	Bit-parallelism based	Automaton + Heuristic based	Heuristics based

TABLE 8
COMPARISON OF VARIOUS MULTIPLE PATTERN STRING MATCHING ALGORITHM

IV. CONCLUSION

This paper is based on multiple pattern string matching algorithms. There are various scenarios where we can use a particular type of algorithm. Aho-Corasick builds an automata of size proportional to the sums of the lengths of all of the substring patterns, and then running it over an input string in a single pass and gives substring matches. The Commentz Walter algorithm has not emerged as a popular string pattern matching algorithm partly due to the difficulty in understanding it. The Wu-Manber algorithm had a better performance with bigger input texts than smaller one. The idea of the rabin-karp algorithm is that it first preprocesses the pattern and assigns a number for the pattern and then searches that number in the text repeatedly and in each step. It substracts a value from the current value and adds a value to the current value. The Rabin-Karp algorithm can be very slow if the text contains a lot of false matches. The complexity of Shift OR algorithm is linear in the length of the text and in the size ofset of patterns.

REFERENCES

[1] Aho, Alfred V.; Margaret J. Corasick (June 1975). "Efficient string matching: An aid to bibliographic search". Communications of the ACM 18 (6): 333–340
[2] Commentz-Walter B. A string matching algorithm fast on theaverage, Proc. 6th International Colloquium on Automata,Languages, and Programming (1979), pp. 118-132.
[3] Robert S. Boyer and J. Strother Moore. A fast string searching algorithm. Communications of the ACM, 20(10):762–772, 1977.

[4] Sun Wu, Udi Manber. A Fast Algorithm For Multi-Pattern Searching, Technical Report TR 94-17, University of Arizona at Tuscon, (May 1994).
[5] Cormen, Thomas H.; Leiserson, Charles E.; Rivest, Ronald L.; Stein, Clifford (2001-09-01). "The Rabin–Karp algorithm". Introduction to Algorithms (2nd ed.). Cambridge, Massachusetts: MIT Press. pp. 911–916.
[6] M. O. Kulekci, "BLIM: A New Bit-Parallel Pattern Matching Algorithm Overcoming Computer Word Size Limitation", Mathematics in Computer Science, 3(4), pp. 407-420, 2010.
[7] K. Lemstrom. String Matching Techniques for Music Retrieval. PhDthesis, University of Helsinki, Faculty of Science, Department of ComputerScience, 2000.
[8] T. Lancaster. Effective and Efficient Plagiarism Detection. PhD thesis,School of Computing, Information Systems and Mathematics, South BankUniversity, 2003.
[9] M. Mozgovoy, V. Tusov, V. Klyuev, "The Use of MachineSemantic Analysis in Plagiarism Detection", Proceedings of the9th International Conference on Humans and Computers, Japan,2006, p. 72-77.
[10] M.J.Atallah, J. Jiangtao Secure Outsourcing of SequenceComparisons International Journal of InformationSecurity, Vol 4, pp.277-287, 2005.
[11] Cornish-Bowden, A. (1985) Nomenclature for incompletely specified bases in nucleicacid sequences: recommendations 1984, Nucl. Acids Res, 13, pp. 3021-3030.
[12] Canfora, G. & Cerulo, L.:"A Taxonomy of Information RetrievalModels and Tools" Journal of Computing and Information Technology- CIT 12, 2004, 3, 175–194
[13] Martin Roesch, "Snort - lightweight intrusion detection for networks," in Proceedings of the13th Systems Administration Conference. 1999, USENIX.
[14] Vern Paxson, "Bro: A system for detecting network intruders in real-time," Computer Networks,vol. 31, no. 23-24, pp. 2435–2463, Dec. 1999.
[15] M. Fisk and G. Varghese. An analysis of fast string matchingapplied to content-based forwarding and intrusion detection.Technical Report CS2001-0670 (updated version), University of California - San Diego, 2002.