Retraction

Retraction: A Novel Approach for Pattern String Matching in Intrusion Detection System (J. Phys.: Conf. Ser. 1916 012007)

Published 23 February 2022

This article (and all articles in the proceedings volume relating to the same conference) has been retracted by IOP Publishing following an extensive investigation in line with the COPE guidelines. This investigation has uncovered evidence of systematic manipulation of the publication process and considerable citation manipulation.

IOP Publishing respectfully requests that readers consider all work within this volume potentially unreliable, as the volume has not been through a credible peer review process.

IOP Publishing regrets that our usual quality checks did not identify these issues before publication, and have since put additional measures in place to try to prevent these issues from reoccurring. IOP Publishing wishes to credit anonymous whistleblowers and the Problematic Paper Screener [1] for bringing some of the above issues to our attention, prompting us to investigate further.

[1] Cabanac G, Labbé C and Magazinov A 2021 arXiv:2107.06751v1

Retraction published: 23 February 2022
A Novel Approach for Pattern String Matching in Intrusion Detection System

K Azarudeen¹, G Vinoth Chakkaravarthy¹, Premkumar Murugiah², S Kharthikeyan

¹Department of Computer Science and Engineering, Velammal College of Engineering and Technology, Madurai – 625009, TamilNadu, India
²Department of Computer Science and Engineering, KPR Institute of Engineering and Technology, Coimbatore – 641407, TamilNadu, India kad@vcet.ac.in

Abstract. String matching algorithms are critical for web usage devices which screen the packets and moves on the basis of load. Intrusion detection or prevention systems, web cleansing, anti-virus and junk preventive systems all have a necessity for effective algorithms when managing string matching scenarios. Some of the algorithms viz. Aho-Corasick, Wu-Manber, Rabin-Karp, Commentz-Walter, Bit parallel etc. entail a lot of space and time. AhoCorasick algorithm is built on the basis of finite state machines. On the shadow of KnuttMorris-Pratt, Commentz Walter developed an algorithm which in turn uses finite state machines. Bit parallel algorithms like shift or takes the help of varied computer words, i.e., the registers in CPU, to compare the efforts. Rabin Karp implements hashing for discovering matching pattern strings in a given text. Wu-Manber looks text in chunks than of every character coalescing the work of Aho-Corasick and Boyer-Moore. Our algorithm reduces character comparisons and consumption of memory on the basis of graph evolution structure and search procedure by means of dynamic linked list. Theoretic study and trial results demonstrates that the projected work is much enhanced in terms of space and time consumption, when compared with previously known pattern-matching algorithms.

Keywords: String matching, Intrusion Detection Systems, Aho-Corasick, Wu-Manber, Bit parallel, Commentz-Walter, Pattern matching

1. Introduction
In computer science, text stays as the foremost way to express and transmit data in a simple and much effective way in wide range of scenarios. Text is understood as a comprehensible order of characters and is generally differentiated from non-character encoded data, like graphics or pictures. Pattern matching is a very significant part in text processing, which would target at discovering a part of text termed pattern \( P = p_1p_2 \ldots p_y \) in a given text \( T = t_1t_2 \ldots t_x \) where \( x > y \) and \( p_i, t_i \) are signs of the alphabet given. The process can be distributed into two ways of matching. The first is entitled exact pattern matching where we target at discovering a pattern \( P = p_1p_2 \ldots p_y \) in the text \( T = t_1t_2 \ldots t_x \) where all \( p_i \) are similar to \( t_i \). The second is entitled approximate pattern matching, also called as error-tolerant pattern matching. The foremost objective of the process is to discover a pattern in a text by revealing either or both the text to some kind of exploitation. Couple of examples of approximate pattern matching are, discovering a DNA subsequence after all potential mutations, and finding words even in the company of spelling mistakes. Various good explanations are offered at precise string matching of multiple...
patterns. The conventional algorithms of Aho-Corasick, Wu-Manber and Commentz-Walter algorithms have some inconsistencies. Many algorithms stay targeted at pattern collections with modest size but unsuitably fail to calculate much fine to patterns of larger collections.

Thus our idea is to build an algorithm which would scale up large pattern sets with minimal space occupancy and with lesser search time. The multi-pattern matching problem is supportive in data filtering to notice certain patterns, for example, web cleansing, anti-virus and spam preventive systems, and specific data mining problems. Many wide application zones moved the field of pattern matching like computational life science, signal processing, and text recovery. Subsequently, the field of pattern matching turned out to be a very dynamic research field, much more domains have resolved many complications implementing the algorithms on the basis of pattern matching techniques like image processing, network security, web search engines, spell checks, and speech analysis. The enormous procurement of data in biology has taken to reality a new research area in the name of bioinformatics. To be specific, bioinformatics attempts to hypothesise biology with respect to molecules in terms of physical-chemistry and it relates informatics systems to recognise and shape the information linked with these molecules on an enormous scale.

One of the peak essential tasks in bioinformatics is the detection of motifs in biological sequences in a way to express the function or the family of biochemical molecules such as RNA, DNA, and Protein. As this task depends on examining textual data, pattern matching algorithms are vital to hold such a problem.

1.1 Intrusion Detection Systems
In the following fragment, we discuss the contextual part of intrusion detection systems. This laterally with the knack in pattern matching procedures is utilised in intrusion detection systems which is significant to completely hold to know the impact of the work proposed. Intrusion detection conceals a wide series of digital security since intrusion detection system has a various kind of procedures. Basically, these are systems that mechanise and automate the course of pulling out the intelligence near past or existing actions which tries to negotiate the security properties of a source. Though characterization of an intrusion in current perspective is insecure, however relatively stands a conception which deviates reliant on the supervision or purpose of the system. Further precisely, the astuteness and facts extended through an intrusion detection system is reliant on in what way the system is utilised, and is as essential as the elected intrusion detection system itself. When the system realises the intrusion, irrespective of its distinctness, there is mutual intention that a system should keep an entry or mark of intrusion, usually by charting or producing an awareness to the suitable party. Moreover, the systems are constructed to turn not only as a critic of intrusions, but also to respond to them.

In the following sections of our proposed work, we analyse groups of intrusion detection systems which are well known to its community. Bearing in mind this grouping aids to slender attention of context in which any exploit or delay creates an intrusion. A comprehend of these classifications would simplify the opportunity of answers for the thesis. It is observed pretty meticulously the area of intrusion detection systems where our findings are placed.

2 Related Works
In [1], string matching algorithms in softwares like virus preventive programs or intrusion detection systems is emphasised for refining security of information above the internet. The methods are implemented for sequence scrutiny, gene discovery. Other grounds like music expertise, Artificial Intelligence is found to use string matching algorithm as the essential portion for theoretic and handson tools. Many difficulties are there in string matching occurred as an outcome of such constant, extensive use, were quickly cracked with the help of computer experts. Many useful results to the real world complications for multiple patterns could be resolved by string matching algorithms. String
Matching Algorithms namely Aho-Corasick, Bit parallel, Commentz-Walter, Wu-Manber, Rabin-Karp etc. are to be concentrated in our work. The selected multiple pattern string matching algorithms are established on time complexity, the kind of search, strategic notions and approach constraints.

In [2], the swift growth in Internet world speed and the continuous evolution with respect to quantity of attacks marks network security a serious test. Intrusion detection systems show a critical part in learning malicious actions and similarly in avoiding their destructive influence. Prevailing intrusion detection system would have substantial overheads with respect to implementation time and space consumption in memory mostly because of the pattern matching process. Here, the research plans to rush the pattern matching process by concurrenting a matching algorithm on a multi-core Central Processing Unit. It is noted that the performance progress drops the identification time and be able to offer a useful influence to sight the variations in the pace of wire and the aggregate quantity of attack signatures.

In [3], it is noted that string matching challenge is to discover the each occurrences of a particular string pattern within a text that is enormous. This contest is ultimate in Computer Science, one of the elementary requirement of various applications. Uniquely, the most common multi pattern string matching algorithm is Aho-Corasick, is developed on the principle of Deterministic Finite Automata amid pattern texts and a precise matching algorithm. The effort of Aho-Corasick algorithm are analysed with respect to its values, demerits and many real time systems like intrusion detection, bioinformatics, identifying plagiarism, text mining and digital forensic etc. are analysed.

The experimental effects display Aho-Corasick algorithm is applicable for multiple pattern matching and be able to be consumed in various application domains. On the other hand, if the magnitude of automata rises severely the performance of algorithm worsens with respect to the complexity of time and space both[4].

2.1 Aho-Corasick Algorithm
The Aho–Corasick algorithm probes a string effectively. Being a type of dictionary-matching process that finds elements of a fixed number of strings in a given text. The procedure equals all the strings at an instance by equating all the strings concurrently in the set and identify these in the text which means the algorithm can match all the strings quickly there by dipping the time taken to perform the search process [5]. The complication is direct or linear for the algorithm with respect to the dimension of the strings along with the length of the text which is searched and the amount of output matches.

Aho-Corasick algorithms applies suffix trees which controls the performance of the execution. Suffix trees can be coded in different ways and based on the way of implementation of the suffix tree, the running time of the algorithm differs [6].

The algorithm builds a trie with suffix tree like structure for the whole patterns which are to be coordinated. The suffix tree has a set of links from every node expressing a pattern to the node that has the extensive proper suffix. The suffix tree is utilised through the run time for matching commitments. So, the novel algorithm implements a procedure so-called output for verifying this besides then gives the keyword identical at the tolerant state. The Aho-Corasick algorithm’s automata is such that a shift into an tolerant state shows a identical of one or other keywords [7].

2.1.1 Preprocessing
In this phase, build an automaton of all pattern words that has mainly three functions.

**Goto:** Store the next state for current state and character.

**Failure:** Store next state for current state when current character doesn’t matches. **Output:** Store indices of all words that end at the current state.

2.1.2 Matching
In the matching phase, traverse the given text over built automaton to find all matching words. Figure 1 shows Working of Aho-Corasick Algorithm
2.1.3 Pseudo Code For Aho-Corasick Algorithm

1: procedure AC(x, m, q₀):

/* Input: x ← m bytes in an array as the input parameters m ← length of the string as integer q₀ ← preliminary state */

2: currstate ← q₀
3: for a = 1 to m
   { 
4:     while h(currstate, x[a]) == failure
       { 
           // while h(currstate, x[a]) is not defined
5:         currstate ← f(currstate)
6:       }
7:     currstate ← h(currstate, x[a])
8:     if o(currstate, x[a]) = null then
9:         print a
10: }

Figure 1. Working of Aho-Corasick Algorithm

Shown (Figure 2) here is a sample for suffix tree. The diagram displays a suffix tree for the string ‘Bananas’. The tree (or trie) covers the whole suffixes of “Bananas” comprising the word itself BANANAS, ANANAS, NANAS so on and ending with S.

Figure 2. Example of Aho-Corasick Algorithm
2.2 Rabin-Karp Algorithm

Rabin and Karp suggested a pattern matching algorithm which was applicable to a two level pattern matching. The algorithm works on the principle done by Karp-Rabin Algorithm, and specially, applies the Rabin fingerprinting method as a number-theoretic perception [8-15].

The algorithm possess distinctive mathematical characteristics occasionally denoted as rolling hash. For inputs that are larger, the impression is a small identifier resembling a hash value. Also, there lies a minimal opportunity of couple of diverse objects with same fingerprint. This characteristic is known as a small probability of collision [16-19].

**Algorithm**

- The initial step is to calculate the value of hash for each patterns.
- Then it pattern’s hash value is equated with the hash value of the present substring in the text.
- If the hash value is similar, every single character is compared.

So, the algorithm has to compute the hash values for the i) pattern itself and ii) every substrings of the string whose length is m.

As the hash value for every substring with a size m from the entire string has to be computed resourcefully, the hash function should possess the following properties.

- It should be efficient in computation.
- It should be extremely selective for strings.

Hash at the subsequent shift is proficiently calculative beginning at the present hash value and upcoming letter in text. Now, rewrite \( h(\text{text}[t+1 \ldots t+m]) \) should be proficiently calculable from \( h(\text{text}[t \ldots t+m-1]) \) and \( \text{text}[t+m] \) i.e., \( h(\text{text}[t+1 \ldots s+m]) = rh(\text{text}[t+m], h(\text{text}[t \ldots s+m-1])) \) and rh rehash should be of O(1) procedure.

The pseudo code and the working of the algorithm in Figure 3 follows

**Pseudo Code For Rabin Karp Algorithm**

1: Procedure RKSet(string s[1..L], group of string subs, length m);
2: set hashsubs := EMPTY
3: for all substring in the group
4:      create hashfunc(sub[1..m]) in hashsubs
5: hashvalue := hashfunc(s[1..m])
6: for count : 1 to n - m + 1
7:    if hashvalue belongs hashsubs and s[i..i+m-1] belongs subs
8:      return count
9:    hashvalue := hashfunc(t[count+1..count+m])
10: return void

**Figure 3. Working of Rabin-Karp Algorithm**
Example
Input:
text[] = "AABAACAADAABAABA"
pattern[] = "AABA"
Output: Pattern seen at index 0
         Pattern seen at index 9
         Pattern seen at index 12

The below Figure 4 shows the same.

Figure 4. Example of Rabin-Karp Algorithm

2.3 Bit-Parallel Algorithm
Bit-parallelism is one procedure which takes the merits of fundamental parallelism of the actions on bit
within a computer word, letting for minimising the quantity of operations for executing an algorithm
over an aspect up to n, where n is the total bits possessed by the computer word. This algorithm is
specifically appropriate for the proficient imitation of non-deterministic automata [20-22].

Bit parallel Algorithms like shift-Or has an edge over that it gets implemented to perform the
string matching complications at the level of bit that outcomes will be acquired quicker when associated
with further approaches of string matching. As per the algorithm, patterns should be not lengthier than
the computer word size.

The bit-parallel algorithms are much effective when applied for patterns of small and larger size
respectively in comparison with other procedures. The execution time drops as the size of the pattern
surges and they effect similar running times for all the scenarios expect for alphabets which are in binary
[23-25].

Shift OR algorithm is an estimated algorithm where it searches the quantity of pattern at an
instance, along with the chances of occurrence of errors. The direction of search activity commences at
left and ends at right. The strategy for patterns that are large in size having same length and not greater
than the word size of computer is worth.

2.3.1 Algorithm
There are two phases involved Shift OR process: 1. Pre-Processing Phase 2. Searching Phase

2.3.1.1 Pre-Processing Phase
Here, the algorithm finds the bit vector of each character of alphabet. If the character is at a
position ‘i’, then that bit is placed with zero. If not so, 1 is placed in that position and the order writing
is done in reverse.
2.3.1.2 Searching Phase
Here, the automation shifts from state ‘s’ to state ‘s+1’ on text t, conditional to ‘sth’ bit in \( t^\text{th} \) index of B is 0. In state vector V where ‘sth’ bit is 0 conditional if state ‘s’ in the automation is lively. If 0 is occurs at MSB, it is understood that the pattern is found in the corresponding position.

The entire searching is processed commencing at left and ending at right one after the other, if 0 occurs at MSB it indicates presence of a pattern. Pre-processing and searching is easy and simple because of the usage of logical operations of Shift and AND in bitwise. The time taken is slow to progress a character confined by a value which is dependent only on the length of the pattern. Here, the size or length of all pattern must be equal. Figure 5 shows Working of Bit-Parallel Algorithm.

2.3.3 Pseudo Code For Bit Parallel Algorithm

1: Pre-Processing
   \[ \text{[string[index]]} < - 1m, \text{shift}=1 \]
   \[ \text{for } x = 0 \ldots z \quad \text{for } x = 0 \ldots c-1 \]
   \[ \text{Bit[pre[x][index]]} < - B[pre[x][index]] \& \sim (\text{shift}<<1) \]
   \[ \text{end for } \]
   \[ \text{end for } \]

2: Searching
   \[ \text{While } \text{pos} < \text{length} \]
   \[ S = 1m \]
   \[ S = S << 1 \mid B[\text{string[pos+j]}] \]
   \[ \text{if } S > 1m-1 \]
   \[ \text{pos} < - \text{pos}++ \quad \text{else} \]
   \[ \text{cnt} < - \text{cnt}++ \quad \text{posn} \]
   \[ \text{posn} \leftarrow \text{posn-m+1} \quad S \leftarrow 1m \quad \text{posn} \leftarrow \quad \text{posn}++ \]

\begin{figure}
\centering
\includegraphics[width=\textwidth]{image5}
\caption{Working of Bit-Parallel Algorithm}
\end{figure}

Example
Here an example is illustrated to know the working of the algorithm. Presume strings FAST, MTYC and BERG are patterns of length 4. Let STRINGFASTMATCH stand with a length of 15. Bit Vector: \( BV[F] = 1110, BV[A] = 1101, BV[S] = 1011, BV[T] = 0011, BV[M] = 1110, BV[C] = 0011, BV[B] = 1110 \) and 1111 for the remaining. Figure 6 shows Various steps of algorithm Shift OR
At the beginning, the value of is assigned to 1111 and is updated when a character is encountered from the text as below in equation (1)

\[ S = (S << 1) | BV[c]. \]  

3. Proposed Algorithm

The anticipated system is to upturn the competence of Pattern Matching Methods by dipping the use of memory and increasing the speed of intrusion detection.

- To diminish the utilization of memory – **Linked list**.
- To improve the speed of pattern search – **Dynamic Pointer**.
- To practise the pattern with dissimilar lengths efficiently.

Our proposal is unique emphasising to develop a list evolution arrangement along with dynamic pointer search method for multi pattern matching. This facilitates handling a huge sum of patterns which would be joined with any existing multiple pattern matching application. We focus on developing a Linked List transition arrangement and dynamic Pointer search method for multi-pattern matching. Our proposed work is to focus the less memory utilization as we use only one pointer table compared to 3 tables used in Wu Manber. Figure 7 shows the workflow of our proposed algorithm.

3.1 Methodology

In pre-processing stage, a linked list transition structure is created representing pattern set P. The linked list L has m levels, m being the maximum size of patterns in P. At each level, we retain the different characters in every patterns. This effectively in our approach, reduces the storage space since the different characters are stored at each level. So, the memory requirement would also be equivalent to or not as much of the storage capacity of DFA alike and state machine and goto function of the algorithms.
we discussed in related works. Secondarily, as the proposed algorithm doesn’t practice the SHIFT and HASH table because it would minimize the development of time and storage table spaces as shown in Figure 8.

To demonstrate the procedure of the algorithm, let us study the given case: Patterns set \( P = \{ \text{"search", "ear", "arch", "chart"} \} \) \( T= \text{"strcmatecadnsearchof"} \). In the searching phase, we implement a pointers list for examining to minimise the consumption of storage capacity. The maximum amount of item in the pointer will be equivalent to the number of patterns. So, the pointer value is initialized and assigned to the length of each pattern. To scrutinize the search method, the given string is implemented as \( T = \text{"strcmatecadnsearchof"} \).

The searching phase traverses the automata, in case of change and the change happens, if not the failure job is verified. The Commentz Walter algorithm performs 15 phases to discover the result= \{ear, arch, search\}, shown in Figure 9. While, the algorithm executes 9 phases to discover two patterns output= \{ear, search\}. Thus to search, we implement a series of pointers that minimise the memory consumption. The highest sum of elements denoted by pointer is equivalent to the total number of patterns. The value of pointer is reset to size of every pattern. The arrangement of the input string as pointer is seen in the Figure 9.

At every step, the pointer \( P_i \) is set where \( i \) relates to the index of the current character. Unknowingly when the present character is identical to the character of whichever pattern \( P_i \), then the decrement the pointer value by 1, or eliminate the value.

Suppose the present character is inequal in the graph \( G \), pointer \( P_i \) should be eliminated, we remain to sustain the process of the \( P_i \). The process of searching phase of the proposed algorithm is illustrated in Figure 9.

---

Figure 8. HASH table

Figure 9. Structure of the input string
It is observed from the above Figure 10 that the steps in algorithm is equivalent to the size of the input string $T$. When there is a text of length $n$, extreme size of the pattern is $L$, and $m$ is number of patterns, then the run time of the algorithm at a worst case is of $O(n\times m \times L)$, even still average case is frequently more improved by not consistently sustaining entire $m$ pointers $P_i$ at an instance.

3.2 Procedure

1: Procedure PatternMatch($T,N,P[],M$) Input:
   $T$ – Input character array
   $N$ – Length of the text in integer
   $P[]$ – Pattern array
   $M$ – Patterns
   Pointer – represents $P[k]$’s character

2: for $i=0$ -> $N$ do
3: for $k=0$ -> $M$ do
4: Pattern = $P[k]$;
5: Pattern.chained = false;
6: if ( $T[i] = $Pattern[Pointer] )
7: Pattern.Pointer++;
8: Pattern.chained = true;
9: endif
10: if ( Pattern.pointer = Pattern.length )
11: Output detected;
12: Pattern.Pointer = 0;
13: endif
14: if ( Pattern.chained not chained ) then
15: Pattern.Pointer = 0;
16: endif
17: endfor
18: endfor

Figure 10. Process of the Algorithm
End procedure

4 Results

Tests are considered to validate the performance of proposed algorithm, with respect to both time and memory consumption, and to compare it with the prevailing algorithms discussed in related works. Assessments are inferred in couple of aspects which balance one another: one where there are fixed quantity of patterns and dynamic pattern size; and the next set using pattern sizes that are static and the dynamic quantity of patterns. The patterns are produced without much concern with identical size, however the input is self-correlated, that shows a partial text produced casually but the remaining is produced according to that part. For example, if the text of size 10,000 is processed, 500 characters without much concern is produced, and the remaining text is produced in such a way that at each stint, many characters are chosen out of the initial 500 characters which is appended to the text till the size touches at 10,000; the site and the size of every choice is arbitrary. In order to mimic patterns and its movement in a network, a text is stemmed which should fulfill total number of matches in the process of identifying patterns that are arbitrary. This results effective in few matches. Although the characters are pointless, they afford a significant mention of the performance.

Figure 11 displays overall search interval of 50,000 characters size text and 500 number of patterns whose lengths are keep changing.

The time seen in Figure 12 is incremental time of 1,000 recurring times of lookup.

5 Conclusion

The proposed novel algorithm looks effective for multi pattern matching exactly than Wu-manber, Commentz-Walter, and their variants. The algorithm excludes the prefix table which is used in the Wu-manber. In its place of three shift table, we use only one pointer table. While in hash table we use list with dynamic pointer. With the incorporation of such modifications, the proposed algorithm shows better outcome when compared with the prevailing algorithms and other tools of pattern matching. Our trial results illustrates that proposed algorithm is much better in both aspect of time and space. In pre-processing stage if we combine all the patterns if possible and construct a trio we may get higher efficiency.
References

[1] Zeeshan Ahmed Khan1, R.K. Pateriya, International Journal of Scientific and Research Publications, Multiple Pattern String Matching Methodologies: A Comparative Analysis Volume 2, Issue 7, July 2012 1 ISSN 2250-3153

[2] Monther Aldwairi, Ansam M. Abu-Dalo and Moath Jarrah, EURASIP Journal on Information Security, Pattern matching of signature-based IDS using Myers algorithm under MapReduce framework. Article number:9, June 02, 2017

[3] Saima Hasib, Mahak Motwani, Amit Saxena, International Journal of Computer Science and Information Technologies Importance of Aho-Corasick String Matching Algorithm in Real World Applications, Volume 4, Issue 3, 2013 Page 467-469

[4] M Aldwairi, N Ekailan, Hybrid multithreaded pattern matching algorithm for intrusion detections systems. J. Inform. Assur. Secur. 6(6), 512–521(2011).

[5] M Kharbutli, A Mughrabi, M Aldwairi, Function and data parallelization of Wu-Manber pattern matching for intrusion detection systems. Network Protocol Algorithms J. 4(3), 46–61 (2012).

[6] X Su, Z Ji, A Lian, in Proceedings of the 2nd International Conference on Computer Science and Electronics Engineering. A parallel AC algorithm based on SPMD for intrusion detection system, (2013).

[7] D Xu, H Zhang, Y Fan, The GPU-based high-performance pattern-matching algorithm for intrusion detection. J. Computer. Information. System. 9(10) (2013).

[8] B. Watson. The performance of single-keyword and multiple-keyword pattern matching algorithms. Technical Report CS TR 94-19, Department of Computing Science, Eindhoven University of Technology, 1994.

[9] B. W. Watson and L. Cleophas. Sparse parts: a C++ toolkit for string pattern recognition. Software Practice and Experience, 34(7):697–710, 2004.

[10] B. W. Watson and G. Zwaan. A taxonomy of sublinear multiple keyword pattern matching algorithms. Science of Computer Programming, 27(2):85–118, 1996.

[11] Sun Wu, Udi Manber. A Fast Algorithm For Multi-Pattern Searching, Technical Report TR 94-17, University of Arizona at Tuscon, (May 1994).

[12] 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.

[13] 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.

[14] K. Lemstrom. String Matching Techniques for Music Retrieval. PhD thesis, University of Helsinki, Faculty of Science, Department of Computer Science, 2000.

[15] T. Lancaster. Effective and Efficient Plagiarism Detection. PhD thesis, School of Computing, Information Systems and Mathematics, South Bank University, 2003.

[16] M. Mozgovoy, V. Tusov, V. Klyuev, —The Use of Machine Semantic Analysis in Plagiarism Detection‖, Proceedings of the 9th International Conference on Humans and Computers, Japan,2006, p. 72–77.

[17] Prakash Duraisamy, XiaohuiYuan, ElSaba,A. and Sumithra Palanisamy, Contrast enhancement and assessment of OCT images, Proceedings of International Conference on Informatics, Electronics & Vision (ICIEV), 2012 Date: 18-19 May 2012 pp.91-95(Location :Dhaka,Print

[18] ISBN: 978-1-4673-1453-3,INSPEC Accession Number: 13058449,Digital Object Identifier :10.1109/ICIEV.2012.6317381

[19] Sumithra M. G., Thanuskhodi, K. and Helan Jenifer Archana .A New Speaker Recognition System with Combined Feature Extraction Techniques , Journal of Computer Science, Vol. 7, Issue 4, pp.459- 465, 2011. (With impact factor SNIP of 0.162 and SJR of0.034)Balasaraswathi,
M., Srinivasan, K., Udayakumar, L., Sivasakthiselvan, S. and Sumithra, M.G., 2020. Big data analytic of contexts and cascading tourism for smart city. Materials Today: Proceedings.

[20] Sivakumar, P., Boopathi, C.S., Sumithra, M.G., Singh, M., Malhotra, J. and Grover, A., 2020. Ultra-high capacity long-haul PDM-16-QAM-based WDM-FSO transmission system using coherent detection and digital signal processing. Optical and Quantum Electronics, 52(11), pp.1-18.

[21] S. Wu and U. Manber, A fast algorithm for multi-pattern searching, Technical Report TR94-17, Department of Computer Science, Chung-Cheng University, 1994.

[22] S. Wu and U. Manber, Agrep – A fast approximate pattern-matching tool, in Proc. USENIX Winter 1992 Technical Conference, 1992, pp. 153–162, San Francisco, CA.

[23] B. C. Walter, A string matching algorithm fast on the average, in Proc. the 6th Colloquium on Automata, Languages and Programming, pp. 118–132, London, UK, Springer-Verlag, 1979.

[24] B. C. Walter, A string matching algorithm fast on the average, Technical Report, IBM Heidelberg Scientific Center, 1979.

[25] A. V. Aho and M. J. Corasick, Efficient string matching: An aid to bibliographic search, Communications of the ACM, vol. 18, no. 6, pp. 333–340, 1975.