String Matching and its Applications in Diversified Fields

Vidya SaiKrishna[1], Prof. Akhtar Rasool[2] and Dr. Nilay Khare[3]

[1]Department Of Computer Science And Engineering Maulana Azad National Institute Of Technology Bhopal-462051

[2]Asst.Professor,
[2]Department Of Computer Science And Engineering Maulana Azad National Institute Of Technology Bhopal-462051

[3]Associate Professor, HOD
[3]Department Of Computer Science And Engineering Maulana Azad National Institute Of Technology Bhopal-462051

Abstract

String searching algorithms, sometimes called string matching algorithms, are an important class of string algorithms that try to find a place where one or several strings (also called patterns) are found within a larger string or text.[11] String matching is a classical problem in computer science. In this paper we are trying to explore the various diversified fields where string matching has an eminent role to play and is found as a solution to many problems. Few of the fields exploited are intrusion detection in network, application in bioinformatics, detecting plagiarism, information security, pattern recognition, document matching and text mining. Here we discuss how string matching is found useful in finding solutions to above problems. String matching algorithms can be categorized either as exact string matching algorithms or approximate string matching algorithms . Also depending upon the kind of application, string matching algorithms are designed either to work on single pattern or multiple patterns. Followed by a brief introduction to string matching, the application areas are discussed.

Keywords: *String matching, intrusion detection, bioinformatics, pattern recognition, text mining, digital forensics.*

1. Introduction

String searching algorithms, sometimes called string matching algorithms, are an important class of string algorithms that try to find a place where one or several strings (also called patterns) are found within a larger string or text. Let Σ be an alphabet (finite set). Formally, both the pattern and searched text are vectors of elements of Σ. The Σ may be a usual human alphabet (for example, the letters A through Z in the Latin alphabet). Other applications may use binary alphabet ($\Sigma = \{0,1\}$) or *DNA alphabet* (Σ = A,C,G,T}) in bioinformatics.[11]
We assume that the text is an array T[1..n] of length n and that the pattern is an array of length[1..m] of length m and that m<=n. The character arrays T and P are often called strings of characters.
We say that pattern P occurs with shift s in text T (or equivalently that the pattern P occurs beginning at position s+1 in text T) if 0<=s<=n-m and T[s+1….s+m]=P[1..m]. If P occurs with shift s in T then we calls a valid shift otherwise we call s an invalid shift. The string matching algorithm is the problem of finding all valid shift with which a pattern P occurs in given text T.[1]

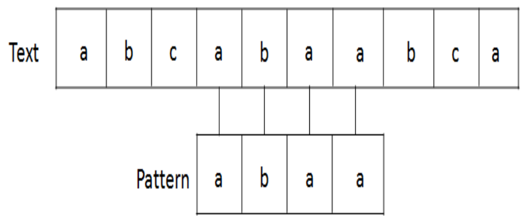


Fig 1

Large number of algorithms is known to exist to solve string matching problem. Based on the number of patterns searched for the algorithms can be classified as single pattern and multiple pattern algorithms. Applications may require exact or approximate string matching.

2. Exact String Matching Problem

We are given a text string pattern string we want to find all occurrences of *P* in *T*. In Exact string matching problem the pattern is exactly found inside the text.[12]
Consider the following example:

$$T = AGCCTAAGCTCCTAAGTC$$
$$P = CCTA$$

There are two occurrences of *P* in *T* as shown below:

$$AG\underline{CCTA}AGCT\,\underline{CCTA}AGTC$$

A brute force method for exact string matching algorithm:

```
T  =  ACCACTAGA
P  =  ACTA
        ACTA
          ACTA
            ACTA
```

If the brute force method is used, many characters which had been matched will be matched again because each time a mismatch occurs, the pattern is moved only one step.

There are many exact string matching algorithms. Nearly all of them are concerned with how to slide the pattern. Few of them are listed below.

2.1 Brute Force Algorithm[12, page 19-20]

- No preprocessing phase.
- Constant extra space needed.
- Always shifts the window by exactly 1 position to the right.
- Comparisons can be done in any order.
- Searching phase in O (m×n) time complexity.
- 2n expected text character comparisons.

2.2 Searching with automation [12, page 25-30]

- Builds the minimal Deterministic Finite Automaton recognizing the language ∑*x.
- Extra space in O(m×σ) if the automaton is stored in a direct access table.
- Preprocessing phase in O(m×σ) time complexity.
- Searching phase in O(n) time complexity.

2.3 Rabin Karp Algorithm[12, page 31-35]

- Uses an hashing function.
- Preprocessing phase in O(m) time complexity and constant space.
- Searching phase in O(m× n) time complexity.
- O(m+n) expected running time.

2.4 Shift OR Algorithm[12, page 37-40]

- Uses bitwise techniques.
- Efficient if the pattern length is no longer than the memory word size of the machine.
- Preprocessing phase in O(m+σ) time and space complexity.
- Searching phase in O(n) time complexity(independent from the alphabet size and the pattern length).
- Adapts easily to approximate string matching.

2.5 Morris Pratt Algorithm[12, page 41-44]

- Performs the comparisons from left to right.
- Preprocessing phase in O(m) space and time complexity.
- Searching phase in O(m+n) time complexity independent from the alphabet size.
- Performs at most 2n -1 text character comparisons during the searching phase.
- Delay bounded by m.

2.6 Knuth- Morris Pratt Algorithm[[12, page 47-50]

- Performs the comparisons from left to right.
- Preprocessing phase in O(m) space and time complexity.
- Searching phase in O(m+n) time complexity independent from the alphabet size.
- Performs at most 2n -1 text character comparisons during the searching phase.
- Delay bounded by $\log_{\Phi}(m)$ where Φ is the golden ratio(1+√5)/2.

2.7 Colussi Algorithm[12, page 61-67]

- Refinement of the Knuth Morris Pratt algorithm.
- Partitions the set of pattern positions into two disjoint subsets. The positions in the first set are scanned from left to right and when no mismatch occurs the positions of the second subset are scanned from right to left.
- Preprocessing phase in O(m) time and space complexity.
- Searching phase in O(n) time complexity.
- Performs 3/2n text character comparisons in the worst case.

2.8 Forward DAWG Matching algorithm[12, page 87-90]

- Uses the suffix automaton of x.
- O(n) worst case time complexity.

- Performs exactly n text character inspections.

2.9 Boyer Moore Algorithm[12, page 91-96]

- Performs the comparisons from right to left.
- Preprocessing phase in O(m+σ) time and space complexity.
- Searching phase in O(m×n) time complexity.
- n text character comparisons in the worst case when searching for non periodic pattern.
- O(n/m) best performance.

2.3.0 Quick Search algorithm[12, page 121-123]

- Simplification of the Boyer Moore algorithm.
- Uses only the bad character shift.
- Easy to implement.
- Preprocessing phase in O(m+σ) time and O(σ) space complexity.
- Searching phase in O(m×n) time Complexity.
- Very fast in practice for short patterns and large alphabets.

3. Approximate String Matching Problem

Approximate string matching is a recurrent problem in computer Science which is applied in text searching, computational biology, pattern recognition and signal processing applications[13]. The problem can be stated as follows:

For a length n and pattern of length m , we are supposed to find all the occurrences of pattern in the text whose edit distance to the pattern is at most K. The edit distance between two strings is defined as minimum number of character insertion, deletion and replacements needed to make them equal.

An Example of approximate string matching is shown below[14]:

T ="appropriate_meaning"

P="approximate_matching"

```
     1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
   T a p p r o p r i a t  e   _  m  ε  e  a  n  i  n  g
      | | | | |         | | |   | |                |  | |
   p: a p p r o x I m a t e   _  m  a  t  c  h  i  n  g
```

δ(t, p) = 7

Dynamic Programming is a method to solve approximate string matching problem. For a Text of length n and pattern of length m ,the dynamic programming method returns a time complexity of O(nm). Bit Parallelism results in faster approximate string matching algorithm like the the fastest non-filtering algorithms in practice are the O(kn[m/w]) algorithm of Wu & Manber, the O([km/w]n) algorithm of Baeza-Yates & Navarro, and the O([m/w]n) algorithm of Myers, where m is the pattern length, n is the text length, k is the error threshold and w is the computer word size.

The motivation for approximate string matching comes from low quality of text, heterogeneousness of databases, spelling errors in the pattern or text, searching for foreign names and searching with uncertainty [13].

4. Multiple String Matching Problem

In multiple string matching we are given a text T = t_1t_2 : : : t_n and want to search simultaneously for a set of strings P = {p^1, p^2, … p^r} Where p^i=p^i_1 p^i_2........... p^i_{mi} is a string of length mi, for i=1…..r[16]. There are many algorithms used for multipattern searching , which varies in speed measured in terms of time complexity. A few are described below .

4.1 Aho-Corasick string matching algorithm

In computer science, the Aho–Corasick string matching algorithm is a string searching algorithm invented by Alfred V. Aho and Margaret J. Corasick. It is a kind of dictionary-matching algorithm that locates elements of a finite set of strings (the "dictionary") within an input text. It matches all patterns simultaneously. The complexity of the algorithm is linear in the length of the patterns plus the length of the searched text plus the number of output matches.

Informally, the algorithm constructs a finite state machine that resembles a trie with additional links between the various internal nodes. These extra internal links allow fast transitions between failed pattern matches (e.g. a search for cat in a trie that does not contain cat, but contains cart, and thus would fail at the node prefixed by ca), to other branches of the trie that share a common prefix (e.g., in the previous case, a branch for attribute might be the best lateral transition). This allows the automaton to transition between pattern matches without the need for backtracking.[17]

4.2 Rabin Karp String matching 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. 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*).A practical application of Rabin–Karp is detecting plagiarism. Given source material, Rabin–Karp can rapidly search through a paper for instances of sentences from the source material, ignoring details such as case and punctuation. Because of the abundance of the sought strings, single-string searching algorithms are impractical [18].

4.3 Commentz-Walter algorithm

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 [19].

5. Applications of String Matching

5.1. Intrusion Detection

Security is a big issue for all networks in today's enterprise environment. Hackers and intruders have made many successful attempts to bring down high-profile company networks and web services. Many methods have been developed to secure the network infrastructure and communication over the Internet, among them the use of firewalls, encryption, and virtual private networks. Intrusion detection is a relatively new addition to such techniques. Intrusion detection methods started appearing in the last few years. Using intrusion detection methods, you can collect and use information from known types of attacks and find out if someone is trying to attack your network or particular hosts. The information collected this way can be used to harden your network security, as well as for legal purposes.[2]
Intrusion detection is a set of techniques and methods that are used to detect suspicious activity both at the network and host level. Intrusion detection systems fall into two basic categories: signature-based intrusion detection systems and anomaly detection systems. Intruders have signatures, like computer viruses, that can be detected using software. You try to find data packets that contain any known intrusion-related signatures or anomalies related to Internet protocols. Based upon a set of signatures and rules, the detection system is able to find and log suspicious activity and generate alerts. Anomaly-based intrusion detection usually depends on packet anomalies present in protocol header parts. In some cases these methods produce better results compared to signature-based IDS. Usually an intrusion detection system captures data from the network and applies its rules to that data or detects anomalies in it. Snort is an open source IDS available to the general public. NIDS are intrusion detection systems that capture data packets traveling on the network media (cables, wireless) and match them to a database of signatures. Depending upon whether a packet is matched with an intruder signature, an alert is generated or the packet is logged to a file or database. One major use of Snort is as a NIDS.

String matching in intrusion detection

The earlier intrusion detection systems makes use of the AC Algorithm (Aho- Corasick) .It is an automaton based multiple string matching algorithm which locates all the occurrences of keywords in a string or text. It first builds a finite state machine of all the keywords in a string and then uses the machine to process the payload in a single pass. The AC algorithm is having a deterministic performance which does not depends on specific input and therefore not vulnerable to various attacks, making it attractive to Network intrusion detection systems.[3]
Let us consider an example to understand how the algorithm works. P={at, cat,rat} is a set of keywords and the algorithm searches for the occurrences of keywords in the text =”cratcatar”. The AC automaton built on pattern P is shown in figure 2. The Dashed arrows represent failure transitions. The failure function is shown in figure 3 and output function in figure 4.

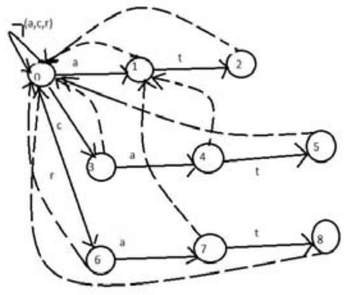
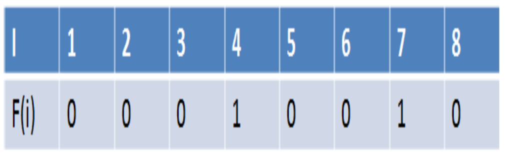
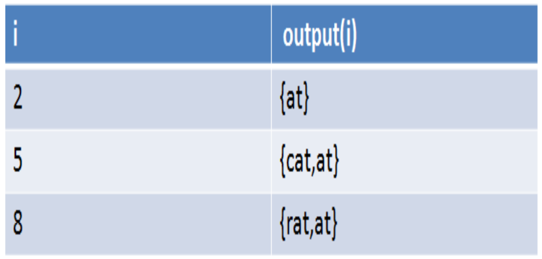


Fig 2: AC Automaton



I	1	2	3	4	5	6	7	8
F(i)	0	0	0	1	0	0	1	0

Fig 3: failure Function



i	output(i)
2	{at}
5	{cat,at}
8	{rat,at}

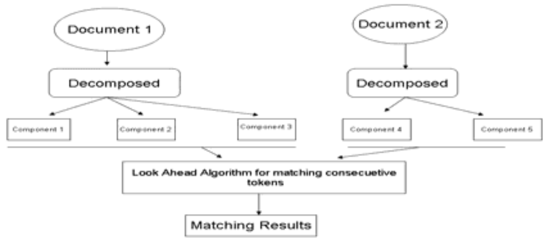
Fig 4: output function

Original AC automaton representations are unsuitable for high performance requirements, which require too much memory, and too much time in the matching process. Recently many works have been presented with the goal of memory reduction for DFAs, by exploiting the intrinsic redundancy in regular expression sets. NIDS not only focus on the header fields but also have to check signatures in the data payload portion of a packet. The most widely used NIDS like Snort and Bro use DFA to describe attack signatures in their rule sets. Because of the large traffic volume and complexity of the process, signature matching can easily become the performance bottleneck in deep packet inspection.[4]

5.2. String matching in detecting plagiarism

Management of large collection of replicated data in centralized or distributed environments is important for many systems that provide data mining, mirroring, storage, and content distribution. In its simplest form, the documents are generated, duplicated and updated by emails and web pages. Although redundancy may increase the reliability at a level, uncontrolled redundancy aggravates the retrieval performance and might be useless if the returned documents are obsolete. Document similarity matching algorithms do not provide the information on the differences of documents, and file synchronization algorithms are usually inefficient and ignore the structural and syntactic organization of documents. For this purpose the *S2S* matching approach is used The *S2S* matching is composed of structural and syntactic phases to compare documents [5]. Firstly, in the structural phase, documents are decomposed into components by its syntax and compared at the coarse level. The structural mapping processes the decomposed documents based on its syntax without actually mapping at the word level. The structural mapping can be applied in a hierarchical way based on the structural organization of a document. Secondly, the syntactic matching algorithm uses a heuristic look-ahead algorithm for matching consecutive tokens with a verification patch. The two-phase *S2S* matching approach provides faster results than currently available string matching algorithms.[4,5]

Figure 5: S2S matching



5.3 String matching in bioinformatics

Bioinformatics is the application of information technology and computer science to biological problems, in particular to issues involving genetic sequences. String algorithms are centrally important in bioinformatics for dealing with sequence information. Modern automated high throughput experimental procedures produce large amounts of data for which machine learning and data mining approaches hold great promise as interpretive means[6].
Approximate matching of a search pattern to a target (called the “text” in string algorithms) is a fundamental tool in molecular biology. The pattern is often called the “query” and the text is called a “sequence database”, but we will use “pattern” and “text” consistent with usage in computer science.
While exact string matching is more commonly used in computer science, it is often not useful in biology. One reason for this is that biological sequences are experimentally determined, and may include errors: a single error can render an exact match useless, where approximate matches are less susceptible to errors and other sequence differences. Another, perhaps more important, reason for the importance of approximate matching is that biological sequences change and evolve. Related genes in different organisms, or even similar genes within the same organism, most commonly have similar, but not identical sequences. Determining which sequences of known function are most similar to a new gene of unknown function is often the first step in finding out what the new gene does.
Another application for approximate string matching is predicting the results of hybridization experiments. Since strands may hybridize if they are similar to each other's reverse complements, prediction of which strands will bind to which other strands, and how stable the binding will be, requires approximate, rather than exact, string matching.[6]

Approximate string matching

```
character[] pattern, text;
integer i,j;
for (i=0;i<length text;i++){
for (j=0; j< length pattern; j++){
if (text[i+j] != pattern[j]) next i;
}
record_match(i);
}
```

Note: the command "next i" exits the current j loop without completing the call to record match.
Biological sequences can be represented as strings. Approximate matching algorithms that can tolerate insertions, deletions, and substitutions are extremely important for biological sequence comparison.
The Shift-AND method uses a bit manipulation approach to accelerate the process of approximate matching. The approach can be explained by comparing it to the naïve exact matching method, in which the pattern is compared character by character at each position along the text. This simple approach is inefficient (its time complexity is O(n*m)) because it contains two nested loops, where the inner loop is executed at each position along the text. Shift-AND uses bit-wise operations in entire registers to perform the inner loop operations on multiple positions in parallel. For patterns that can be contained within the length of a register, it has a time complexity proportionate to the length of the text being searched (O(n)). Shift-AND actually uses a set of four registers (called the "U" registers [Gusfield 1997]) to contain the pattern, with one bit in each register to represent each base of the pattern. Thus, a machine with 32 bit registers can easily represent 32 base patterns for use in shift-AND. Longer registers, such as the 128 bit registers of the PowerPC Alti Vec vector engine, can represent proportionately longer patterns. The algorithm can also be extended to use multiple registers to represent longer patterns, but (depending on the architecture), this would likely be at a cost of increased time complexity.
Of course, shift-AND is more sophisticated than a simple short-circuit of the inner loop in the naïve exact matching approach, in that it can be extended to allow for approximate matches 9
[Wu 1991].
The full algorithm allows for a given number of substitutions, insertions, or deletions in the pattern. This is achieved by maintaining a set of matrixes, which allow different numbers of errors. A bit is set if the current characters match and the prefixes of each string were within the error limits; if the prefixes had not reached the error limits, the bit is set even if there is an error at the current position.

5.4 String matching in Digital Forensics

Textual evidence is important to the vast majority of digital investigations. This is because a great deal of stored digital data is linguistic in nature (e.g. human languages, programming languages, and system and application logging conventions). Some examples of important text-based evidence include: email, Internet browsing history (both logs and the content itself), instant messaging, word processing documents, spreadsheets, presentations, address books, calendar appointments, network activity logs, and system logs.
Digital forensic1 text string searches are designed to search every byte of the digital evidence, at the physical level, to locate specific text strings of interest to the investigation. Given the nature of the data sets typically encountered, text string search results are extremely noisy, which results in inordinately high levels of information retrieval (IR) overhead and information overload (Beebe and Dietrich, 2007). Frequently, investigators are left to wade through hundreds of thousands of search hits for even reasonably small queries (i.e. 10 search strings) on reasonably small devices (i.e. 80 GB) – most of which (i.e. 80–90% of the search hits) are irrelevant to investigative objectives. The investigator becomes inundated with data and wastes valuable investigative time scanning through noisy search results and reviewing irrelevant search hits presumably by some automated means. This can prohibit There are fundamentally two classes of solutions to this problem: (1) decrease the number of irrelevant search hits returned, or (2) present the search hits in a manner which enables the investigator to find the relevant hits more quickly. The first solution class is disconcerting to many investigators and litigators, since it results in information reduction, the investigator from finding important evidence. The second solution class encompasses the basic approach that revolutionized web-based knowledge discovery: search hit ranking. This approach presents hits in priority order based on some determination of similarity and/or relevancy to the query. This approach is much more attractive to investigators and litigators, since it improves the ability to obtain important information, without sacrificing fidelity.
Current digital forensic text string search approaches fail to employ either solution class. They use simple matching and/or indexing algorithms that return all hits. They fail to successfully implement grouping and/or ranking algorithms in a manner that appreciably reduce IR overhead (time spent scanning/reviewing irrelevant search hits). Search hits are commonly grouped by search string and/or

‘‘file item3’’ and/ or ordered by their physical location on the digital media.
Such grouping and ordering are inadequate; as neither substantially helps investigators get to the relevant hits first (or at least more quickly). New, better approaches in the second solution class are needed.
The Current researches in digital forensics aims at improving IR(information Retrieval) effectiveness of digital forensics text string searches.[7,8]

5.5 Text Mining Research

Text mining includes tasks designed to extract previously unknown information by analyzing large quantities of text, as well as tasks geared toward the retrieval of textual data from a large corpus of documents (Sebastian, 2002; Fan et al., 2006; Sullivan, 2001). Several information processing tasks fall under the umbrella of text mining: information extraction, topic tracking, content summarization, information visualization, question answering, concept linkage, text categorization/ classification, and text clustering (Fan et al., 2006).
These are defined as follows:
1. Information extraction: identifies conceptual relationships, using known syntactic patterns and rules within a language.
2. Topic tracking: facilitates automated information filtering, wherein user interest profiles are defined and fine-tuned based on what documents users read.
3. Content summarization: abstracts and condenses document content.
4. Information visualization: represents textual data graphically (e.g. hierarchical concept maps, social networks, timeline representations).
5. Question answering: automatically extracts key concepts from a submitted question, and
Subsequently extracts relevant text from its data store to answer the question(s).
6. Concept linkage: identifies conceptual relationships between documents based on Transitive relationships between words/concepts in the documents.
7. Text categorization/classification: automatically and probabilistically assigns text documents into predefined thematic categories, using only the textual content (i.e. no metadata).
8. Text clustering: automatically identifies thematic categories and then automatically assigns text documents to those categories, using only textual content (i.e. no metadata).

If applied to digital forensic text string searching, information extraction, content summarization, information visualization, and concept linkage would fit into the first solution class identified earlier (reduction of search results set size).These text mining approaches reduce the search result set size via data abstraction techniques, by and large.

5.6 String matching Based Video Retrieval

String matching can be effectively used to retrieve fast video as it uses the content based video retrieval in contrast with the traditional video retrieval which was slow and time consuming. String based video retrieval method first converts the unstructured video into a curve and marks the feature string of it. Approximate string matching is then used to retrieve video quickly. In this method the characteristic curve of the key frame sequence is first extracted followed by marking the feature string and then approximate string matching is used on the feature string to get fast video retrieval [10].

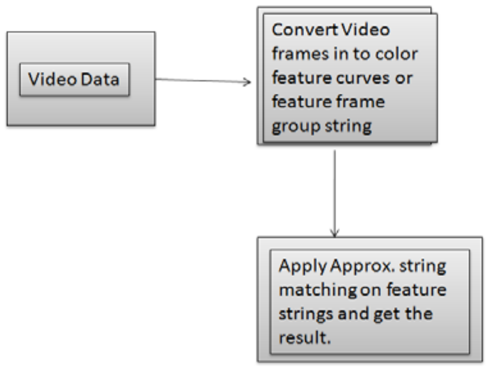

Video Data
Convert Video frames in to color feature curves or feature frame group string
Apply Approx. string matching on feature strings and get the result.

Fig 6: Video Retrieval Process

6. Conclusion

String matching has greatly influenced the field of computer science and will play an important role in future technology. The importance of memory and time efficient string matching algorithm will be increased in computer science. There are many more possible areas in which string matching can play a key role for excelling. Exact and approximate string matching algorithm makes various problems in the solvable state. Innovation and creativity in string matching can play a immense role for getting time efficient performance in various domains of computer science.

References

[1] Thomas H Corman, Charles E. Leiserson, Ronald L. Rivest & Clifford Stein “Introduction to Algorithms-String matching”, EEE Edition, 2nd Edition, Page no 906-907.

[2] Ali Peiravi, “Application of string matching in Internet Security and Reliability”, Marsland Press Journal of American Science 2010, 6(1): 25-33

[3] Peifeng Wang , Yue Hu, Li Li, “An Efficient Automaton Based String Matching Algorithm and its application in Intrusion Detection”, International Journal of Advancements in Computing Techology(IJACT), Vol 3, Number 9 , October 2011

[4] Pekka Kilpelainen, “Set Matching and Aho-Corasick Algorithm”, Biosequence Algorithms, Spring 2005, BSA Lecture 4

[5] Ramazan S. Aygün “structural-to-syntactic matching similar documents”, Journal Knowledge and Information Systems archive, Volume 16 Issue 3, August 2008.

[6] Robert M. Horton, Ph.D. “Bioinformatics Algorithm Demonstrations in Microsoft Excel” , 2004 - cybertory.org

[7] Nicole Lang Beebe, Jan Guynes Clark, “Digital forensic text string searching: Improving information retrieval effectiveness by thematically clustering search results”, d i g i t a l i n v e s t i g a t ion 4 S (2 0 0 7) S 4 9 – S 5 4

[8] Beebe NL, Dietrich G. “A new process model for text string searching”. In: Shenoi S, Craiger P, editors. Research advances in digital forensics III. Norwell: Springer; 2007. p. 73–85.

[9] Rafeeq Ur Rehman , “Intrusion Detection Systems with Snort Advanced IDS Techniques Using Snort Apache, MySQL, PHP, and ACID”

[10]Yin Jian, Yu Xiu ,Dong Meng, “ Application of Approximate String Matching in Video Retrieval”, 2010 3rd International Conference on Advanced Computer Theory and Engineering(ICACTE),vol 4, page 348-351.

[11]Wikipedia The free Encyclopedia *en.wikipedia.org/wiki/**String**_searching_algorithm*,

[12] Christian Charras,Thierry Lecroq , “Handbook of exact string matching algorithms”

[13] Ricardo Baeza-Yates, Gonzalo Novarro, “Fast Approximate string matching in a Dictionary”,Bulletin of the Technical Committee, 2000

[14] Yoan Pinzon, “Algorithm for approximate string Matching”,*dis.unal.edu.co/~fgonza/courses/2006.../ approx_**string_matching**.pdf* August 2006

[15] Heikki Hyyo, “Bit Parallel approximate string matching algorithm with transposition”String Processing and Information Retrieval, 2003 – Springer.

[16] D. Huson , “multiple string matching”, Comp. Sequence Analysis, Nov 17 , 2004

[17] Aho, Alfred V.; Margaret J. Corasick (June 1975). "Efficient string matching: An aid to bibliographic search". Communications of the ACM **18** (6): 333–340.

[18] http://en.wikipedia.org/wiki/Rabin_kar p_string_search_algorithm.

[19] http://en.wikipedia.org/wiki/Commentz_Walter_ algorithm

Vidya Saikrishna, B.E in Computer Science From University Institute of Technology, Bhopal in the year 2002, M.Tech in Computer Science From Maulana Azad National Institute of Technology. Worked as Asst. Prof in Truba Institute of Science and Technology, Bhopal, Asst. Prof. in Institute of Technology and Science , Ghaziabad. and Presently working as Asst. Prof. in SAM College of Engineering and Technology.

Akhtar Rasool, B.E in Computer Science from Rajiv Gandhi Technical University in the year 2003,M.Tech in Computer Science from Maulana Azad National Institute of Technology. Presently Working as Asst. Prof in Department of Computer Science in Maulana Azad National Institute Technology, Bhopal.

Dr. Nilay Khare, Associate Prof. and Head in Department of Computer Science in Maulana Azad National Institute of Technology,Bhopal. Reviewer of Journal Elsevier