Novel Devaki-Paul Algorithm for Multiple Pattern Matching

Devaki Pendlimarri
Swarnandhra Institute of Engineering and Technology
Narsapur – 534 280, India

Paul Bharath Bhushan Petlu
Swarnandhra College of Engineering and Technology
Narsapur – 534 280, India

Dr. Ramesh Babu Satrasala
Swarnandhra College of Engineering and Technology
Narsapur – 534 280, India

ABSTRACT
Pattern matching is one of the major issues in the area of network security and also in many other areas. The increase in network speed and traffic may cause the existing algorithms to become a performance bottleneck. Therefore, it is very necessary to develop more efficient pattern matching algorithm, in order to overcome troubles on performance. There are several algorithms in use, in which, DP algorithm (Devaki – Paul algorithm) is yielding good results in many cases. However, this algorithm was proposed only for the single pattern matching. But, now-a-days it is very necessary to find the multiple patterns in NIDS etc. In this paper, we are giving a new proposal to the DP algorithm (Devaki – Paul algorithm) to make it efficient and works also for the multiple pattern matching. The algorithm was tested and validated and the results have proved that the performance of DP algorithm is better than BM algorithm (Boyer – Moore algorithm) and the Quick Search algorithm. In case of tests with repeated character, its performance is greater than 1%~50% with BM and Quick Search algorithms. In case of tests with the English Text and Random Pattern, it's greater than 33%~91% with BM and 37%~85% with Quick Search algorithms. In case of tests with the English Text and Random Pattern of an unsuccessful search, its performance is greater than BM and Quick Search algorithms with 100%, if either the first and/or the last character of the pattern in the given text were not present.

General Terms
Exact String matching algorithms, pattern matching, Network Security Systems, Algorithms et. al.

Keywords
Devaki-Paul algorithm, pattern matching, string matching algorithm, Boyer-Moore algorithm and network security.

1. INTRODUCTION
Pattern matching is one of the basic and most important issues, which have been studied, in the research areas of computer science. In a standard problem, we are required to find all occurrences of the pattern in the given input text, known as single pattern matching [6]. Suppose, if more than one pattern are matched against the given input text simultaneously, then it is known as, multiple pattern matching. The multiple patterns matching technique can be used in many applications. It is used in data filtering or data mining (to find selected patterns from a stream of new feed), network security applications (to detect certain suspicious keywords), glimpse to support Boolean queries by searching for all terms at the same time and then intersecting the results, DNA searching, patterns that can have several forms such as dates, names etc and also in many areas.

String matching algorithms are used also in Network Intrusion Detection Systems (NIDS)[3] which is widely recognized as a powerful tool for identifying deterring and deflecting malicious attacks over the network. In network security realm, the pattern is a string indicating a network intrusion, attack, virus, snort, spam or dirty network information etc [9]. Besides, there are many other applications which can be found in [4], [5], [6], and [7]. Here, we are proposing an enhancement to the preprocessing phase of the DP algorithm (Devaki – Paul algorithm). With this enhanced preprocessing phase the DP algorithm is able to find all the occurrences of multiple patterns in a given input text.

Since the evolution of the Boyer – Moore (BM algorithm) [8] and the Knuth–Morris–Pratt (KMP algorithm) [2] and [12] algorithms, many more techniques were proposed by many researchers, to find the exact pattern matching, by improving the performance and efficiency. The BM algorithm is considered as one of the most efficient pattern matching algorithm in general applications and is also known as the best average-case performance algorithm of any algorithm [8], [9] and [10]. The algorithm requires a preprocessing of the given input text with respect to the given pattern to construct a table before starting the search, which takes $O(m+\sigma)$ time. If we want to search for more patterns in the same given input text, we need to repeat the preprocessing phase on the same given input text with respect to every new pattern. Hence, the BM algorithm is yielding a good result in case of single pattern matching, but, not for multiple patterns matching.

The DP algorithm [1] is yielding good results when compared with the above said algorithms in many cases for single pattern matching. The key features of DP algorithm for single pattern matching are:

- The preprocessing phase is of $O(m)$ time.
- The search phase time complexity is directly proportional to the number of occurrences of the first and the last characters of the pattern in the given input text.
- If either the first and/or the last character occurrences is zero in the preprocessing phase, then the time complexity of the search phase is of $O(1)$ time.
- If the search is for another pattern for which the first and the last character are same as the previous pattern, then the preprocessing phase is not required. The previously computed table of occurrence can be used.

But, the DP algorithm also requires a preprocessing phase every time for every new pattern which results to a poor performance.

Hence, in this paper, we propose a novel DP algorithm for multiple patterns matching, which undergoes preprocessing phase only once for all patterns before search phase. This algorithm

requires a preprocessing phase of $O(m+\sigma)$ time, where m is the size of the given input text, and σ is the fixed amount of time required to construct the table of ASCII character set.

2. METHODOLOGY

The DP algorithm (Devaki–Paul algorithm), for single pattern matching can be made efficient also for multiple pattern matching by replacing the preprocessing phase with this novel methodology.

Here, we present a novel multiple patterns matching algorithm, called, Devaki–Paul algorithm (DP algorithm). This algorithm requires a preprocessing phase, in which it prepares a table which constitutes a 256 member ASCII character set to store the occurrences of each character in the given input text. The given input text is scanned once and the occurrences of each character are stored in the respective 256 member ASCII character set table.

2.1 Preprocessing Phase

In this phase, we find the occurrences of each character of the 256 member ASCII character set table in the given input text.

Preprocessing*(char x[], int m)*

/ Constructing a table of occurrences for each character of the 256 member ASCII character set in the given input text*/*

Step 1: [initialization]

Construct a table constituting with 256 member ASCII character set

Step 2: [scan the given input text]

Find all the occurrences of each character represented in the 256 member ASCII character set in the given input text

Save these occurrences in the respective character entry in the 256 member ASCII character table.

Step 3: [Finish]

return

Then, it performs a search phase for the multiple pattern matching based on the pre-computed table with a set of rules.

2.2 Search Phase

In this phase, we find the probability of having an occurrence of a pattern in the given input text by using the table of occurrences of pre-processing phase.

Searching*(char x[], int m, char y[], int n, int a[], int alen, int b[], int blen)*

/ Search Phase: This algorithm will find the chances of getting pattern match */*

Step 1: [initialing the variables]

Initialize all index and other variables

Step 2: [get occurrences]

Read the first character occurrences of the pattern from the pre-computed 256 member ASCII character set occurrences table

Read the last character occurrences of the pattern from the pre-computed 256 member ASCII character set occurrences table

Step 3: [find the probability of occurrence of a pattern and do match]

Repeat step4 until the end of the table of last character occurrences

Step 4: [calculate the difference between the last and first character occurrence]

Case 1: difference > n – 1

Update the index of the table of first character occurrence

Go to step 3

Case 2: difference < n – 1

Update the index of the table of last character occurrence

Go to step 3

Case 3: difference = n – 1

Match()

Update the indices of both tables of first and last character occurrence

Go to step 3

Step 5: [finish]

return

In the above algorithm, once we find the probability of occurrence of a pattern in the given input text, we perform an exact pattern matching by comparing the remaining characters of the pattern sequentially with the characters of the given input text.

3. IMPLEMENTATION

The DP algorithm for multiple pattern matching requires a preprocessing of the given input text to prepare a table of the occurrences of the 256 member ASCII character set. This table is used to find the probability of having a match of the pattern in the given input text, which reduces the number of comparisons, improving the performance of the pattern matching algorithm. The probability of having a match of the pattern in the given text is mathematically proved.

3.1 Mathematical Proof

Here, we get the occurrences of the first and the last characters of the given pattern from the already pre-computed 256 member ASCII character set. If the difference between any two occurrences of the last and the first characters of the pattern in the pre-computed table is less than the size of the pattern by one, then, it is taken as one probability for occurrence of an exact pattern match.

Let, x, be the given text of size m and y, be the given pattern of size n, where $m \geq n$. Let us assume that, $A[a1, a2, \ldots, ai]$, is an array, in which, $a1, a2, \ldots, ai$ represents the occurrences of the first character of the given pattern, y, from the pre-computed 256 member ASCII character set, where, $0 \leq i \leq m$. Similarly, $B[b1, b2, \ldots, bj]$, is an array, in

which, $b1, b2, \ldots, bj$, represents the occurrences of the last character of the given pattern, *y,* from the pre-computed 256 member ASCII character set, where, $0 \leq j \leq m$.

We know that,

y[1] = the position of the first character in the pattern

y[n] = the position of the last character in the pattern

then, the offset between the last character and the first character in the pattern is:

$$offset = n - 1 \qquad (1)$$

Now, from the pre-computed table,

If *A[i]* = a_i = one of the positions of the first character of the pattern in the given input text then the position of the last character of the pattern in the given input text in case of exact pattern match must be:

$$lastpos = a[i] + offset \qquad (2)$$

Let, if *b[j]* = one of the positions of the last character of the pattern in the given input text then,

$$lastpos = b[j] \qquad (3)$$

From equations (2) & (3), we get

$$b[j] = a[i] + offset$$
$$\Rightarrow b[j] - a[i] = offset$$

Substituting equation (1), we get

$$b[j] - a[i] = n - 1 \qquad (4)$$

Hence, we proved that, any condition which satisfies with the equation (4), is a probability of occurrence of a pattern.

3.2 Description

Let us assume that the example text and patterns are as in Fig 1. The table of occurrences of the first and last characters of the pattern in the given input text will be obtained from the pre-computed 256 member ASCII character set. After the pre-processing phase a search phase begins, where we use *Search* algorithm as given in the methodology, to find the probability of a pattern match. Here, we get three possibilities:

Possibility 1: *B[j] – A[i] > one less than the size of the pattern, i.e., n-1, where j ≥ i at all times*

From the table, we have: *B[j] = 16 and A[i] = 12.* Hence, the condition given in possibility 1 is satisfied. This specifies that there is no possibility of having an occurrence of the pattern in the given input text at the current location. Hence, the index, *i,* is incremented to search for the next possibility.

Possibility 2: *B[j] – A[i] < one less than the size of the pattern, i.e., n-1, where j ≥ i at all times*

From the table, we have: *B[j] = 16 and A[i] = 15* as the index variables, *i,* was incremented two times in the possibility 1. Hence, the condition given in possibility 2 is satisfied. This specifies that there is no possibility of having an occurrence of the pattern in the given input text at the current location. Hence, the index, *j,* is incremented to search for the next possibility.

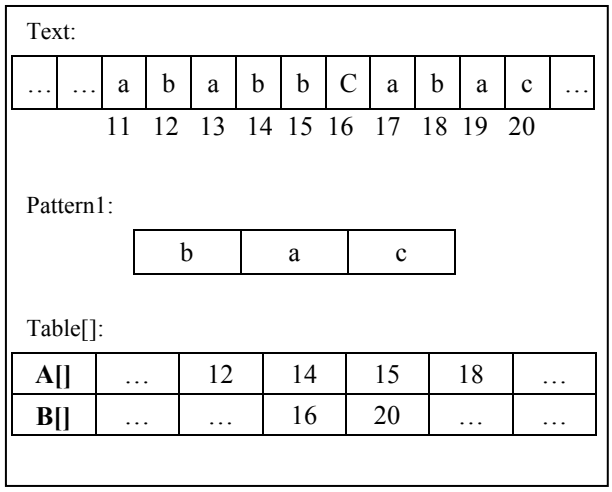

Fig. 1 The example text and pattern with the table of occurrence of the first and the last characters of the pattern in the given input text.

Possibility 3: *B[j] – A[i] = one less than the size of the pattern, i.e., n-1, where j ≥ i at all times*

From the table, we have: *B[j] = 20 and A[i] = 18* as the index variable, *i and j,* were incremented in the possibilities 1 and 2. Hence, the condition given in possibility 3 is satisfied. This will be taken as a probability of occurrence of a pattern in the given input text at the current location and then execute the algorithm, *Match(),* which finds whether the pattern exists in that place or not by comparing the remaining character of the given pattern with the character in the given input text sequentially. In this example, here, we find the pattern. Index variables, *i and j,* are incremented to search for the next possibility.

4. RESULTS AND ANALYSIS

We have implemented and tested the novel DP algorithm for multiple patterns matching using object oriented programming with Java and the results are as below.

4.1 Tests with Repeated Characters

The input text and the patterns are either taken with the same character or with the repeated set of characters. It provides the worst case situation for the pattern matching algorithm. A text of size 1024 bytes was taken as given below and tested with the multiple patterns as shown in the table 1.

“agaacgcagagacaaggttctcattgtgtctcgcaatagtgttaccaactcgggtgcctattg
gcctccaaaaaaggctgttcaacgctccaagctcgtgacctcgtcactacgacggcgagtaa
gaacgccgagaaggtaagggaactaatgacgcgtggtgaatcctatgggttaggatcgtgtct
accccaaattcttaataaaaaacctaggaccccccttcgacctagactatcgtattatggacaagc
tttaactgtcgtactgtggaggcttcaaaacggagggaccaaaaaatttgcttctagcgtcaatg
aaaagaagtcgggtgtatgccccaattccttgctgcccggacggccaggcttatgtacaatcc
acgcggtactacatcttgtctcttatgtagggttcagttcttcgcgcaatcatagcggtacttcata
atgggacacaacgaatcgcggccggatatcacatctgctcctgtgatggaattgctgaatgcg
caggtgtgaatactgcggctccattcgttttgccgtgttgatcgggaatgcacctcggggactgt
tcgatacgacctgggatttggctatactccattcctcgcgagttttcgattgctcattaggctttgc
ggtaagtaagttctggccacccacttcgagaagtgaatggctggctcctgagcgcgtcctccg
tacaatgaagaccggtctcgcgctaaatttcccccagcttgtacaatagtccagtttattatcaaa

gatgcgacaaataaattgatcagcataatcgaagattgcggagcataagtttggaaaactggg aggttgccagaaaactccgcgcctactttcgtcaggatgattaagagtatcgaggccccgccg tcaataccgatgttcttcgagcgaataagtactgctattttgcagacccttgccaggccttgtcta aaggtatgttacttaatattgacaatacatgcgtatggccttttccggttaactccctg"

TABLE 1: Patterns with their sizes

S. No.	Pattern	Size (bytes)
1	A	1
2	AG	2
3	CAT	3
4	AACG	4
5	AAGAA	5
6	AAAAAACG	8
7	TTCTTAATAAAA	12
8	GGCTGTTCAACGCTCC	16

The results were compared with the BM algorithm and the Quick Search algorithm and are shown in the table 2 and also plotted in the graph as shown in the Fig. 2. The performance of DP algorithm has been improved with respect to the BM algorithm with 2.8%~74.71% and the Quick Search algorithm with 3.86%~74.71%.

TABLE 2: Comparison with BM and QS algorithms

Patterns	Occurr-ences (respect-ively)	Character Comparisons			Least
		BM*	QS*	DP	
A	259	1024	1024	259	259 DP
A, AG	259, 52	1758	1676	711	711 DP
A, AG, CAT	259, 52, 11	2399	2283	1228	1228 DP
A, AG, CAT, AACG	259, 52, 11, 4	2909	2787	1754	1754 DP
A, AG, CAT, AACG, AAGAG	259, 52, 11, 4, 2	3282	3153	2281	2281 DP
A, AG, CAT, AACG, AAGAG, AAAAAACG	259, 52, 11, 4, 2, 0	3554	3529	2831	2831 DP
A, AG, CAT, AACG, AAGAG, AAAAAACG, TTCTTAATAAAA	259, 52, 11, 4, 2, 0, 1	3731	3729	3398	3398 DP
A, AG, CAT, AACG, AAGAG, AAAAAACG, TTCTTAATAAAA, GGCTGTTCAACGCTCC	259, 52, 11, 4, 2, 0, 1, 0	4041	4086	3928	3928 DP

*preprocessing is required for every pattern before search with respect to that pattern in using BM and QS algorithms, where as pre-processing phase is required only once in case of DP algorithm.

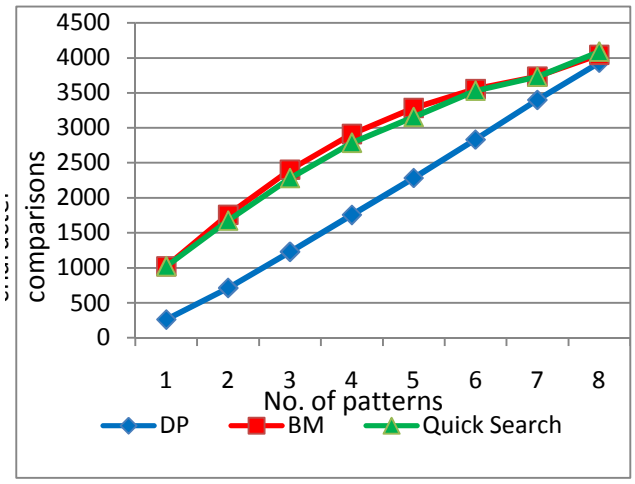

Fig. 2 The Performance of DP algorithm with Repeated characters

4.2 Tests with an English Text and Random Patterns:

A text of size 1024 bytes was taken as given below and tested with the multiple patterns as given in the table 3.

"patternmatchingisoneofthebasicandmostimportantissuesintherese archareasofcomputersciencethemeaningofthepatternmatchingisthat findingtheoccurrencesofagivenpatterninthegiventextpatternmatchi ngisoneofthemajorissuesintheareaofnetworksecurityandalsoinman yotherareastheincreaseinnetworkspeedandtrafficmaycausetheexisti ngalgorithmstobecomeaperformancebottleneckthereforeitisveryne cessarytodevelopmoreefficientpatternmatchingalgorithminorderto overcometroublesonperformancethereareseveralalgorithmsinusein whichdpalgorithmisyieldinggoodresultsinmanycaseshoweverthisal gorithmwasproposedonlyforthesinglepatternmatchingbutnowaday sitispatternmatchingisoneofthebasicandmostimportantissuesinther esearchareasofcomputersciencethemeaningofthepatternmatchingis thatfindingtheoccurrencesofagivenpatterninthegiventextpatternmat chingisoneofthemajorissuesintheareaofnetworksecurityandalsoinm anyotherareastheincreaseinnetworkspeedandtrafficmaycausetheexi stingalgorithmstobecomeaperformancebottleneckthereforeitisvery necessarytodevelopmoreefficientpatte",

TABLE 3: Patterns with their sizes

S. No.	Pattern	Size (bytes)
1	H	1
2	OF	2
3	AND	3
4	MOST	4
5	GIVEN	5
6	MATCHING	8
7	PATTERNMATCH	12
8	ONEOFTHEBASICAND	16

The results obtained by using DP algorithm for multiple patterns matching were compared with that of the BM algorithm and the Quick Search algorithm and are shown in the table 4 and also plotted in the graph as shown in the Fig. 2.

TABLE 4: Comparison with BM and QS algorithms

Patterns	Occurr-ences (respect-ively)	Character Comparisons			Least
		BM*	QS*	DP	
H	48	1024	1024	48	48 DP
H, OF	48, 12	1571	1440	133	133 DP
H, OF, AND	48, 12, 6	1963	1787	233	233 DP
H, OF, AND, MOST	48, 12, 6, 2	2295	2054	375	375 DP
H, OF, AND, MOST, GIVEN	48, 12, 6, 2, 4	2587	2292	495	495 DP
H, OF, AND, MOST, GIVEN, MATCHING	48, 12, 6, 2, 4, 8	2812	2506	595	595 DP
H, OF, AND, MOST, GIVEN, MATCHING, PATTERNMATCH	48, 12, 6, 2, 4, 8, 8	3028	2732	738	738 DP
H, OF, AND, MOST, GIVEN, MATCHING, PATTERNMATCH, ONEOFTHEBASICAND	48, 12, 6, 2, 4, 8, 8, 2	3173	2907	850	850 DP

*preprocessing is required for every pattern before search with respect to that pattern in using BM and QS algorithms, where as pre-processing phase is required only once in case of DP algorithm.

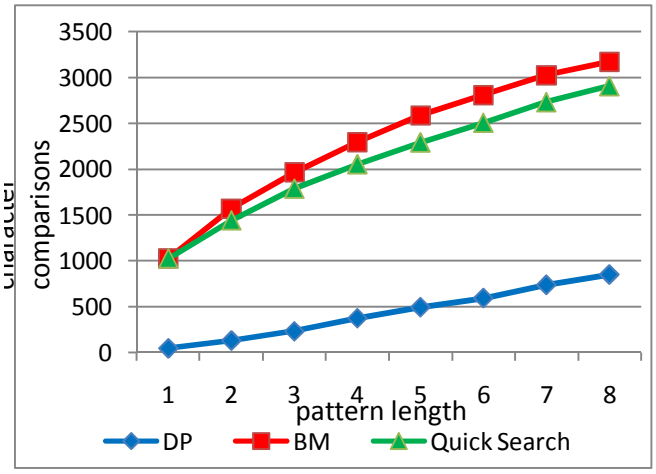

Fig. 3 The performance of DP algorithm with an English Text and Random Pattern

The performance of DP algorithm has been improved with respect to the BM algorithm with 73.21%~95.31% and the Quick Search algorithm with 70.76%~95.31%.

4.3 Tests with English Text and Random Patterns (Unsuccessful Search):

In this case, we have taken a set of patterns as an example, which leads to an unsuccessful search as given in the table 5 and a text as in the case of tests with English text and random patterns.

TABLE 5: Patterns with their sizes

S. No.	Pattern	Size (bytes)
1	Z	1
2	QE	2
3	ZOO	3
4	ZYLO	4
5	QUEUE	5
6	QUESTION	8
7	ZEROXANSWERS	12
8	QUESTIONONANSWER	16

The results obtained by using DP algorithm for multiple patterns matching were compared with that of the BM algorithm and the Quick Search algorithm and are shown in the table 6 and also plotted in the graph as shown in the figure Fig. 2.

Here, irrespective of the sizes of the patterns and the texts, the number of character comparisons is zero. For example, if either the first and/or the last characters of the patterns are not present in the given input text, then certainly, there is no possibility of having an occurrence of the pattern in the given input text. In this case, the performance of DP algorithm for multiple patterns matching is 100% better than the BM algorithm and the Quick Search algorithm. However, if both the first and the last characters of the pattern are present in the given input text, then the search case will be as in the second case, i.e., tests with English text and random patterns.

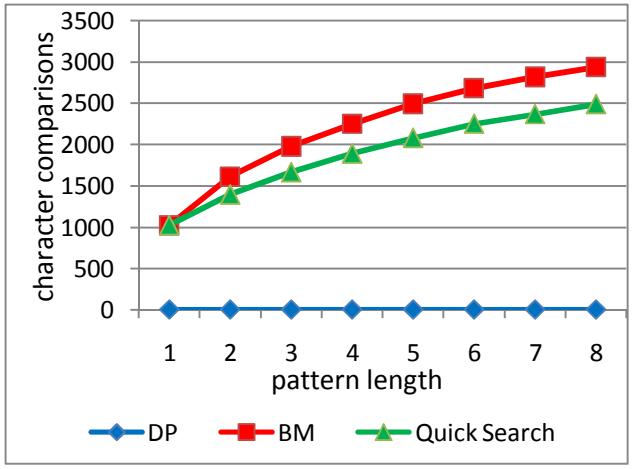

Fig. 4 The performance of DP algorithm with an English Text and Random Pattern (Unsuccessful Search)

TABLE 6: Comparison with BM and QS algorithms

Patterns	Occurr-ences (respect-ively)	Character Comparisons			Least
		BM*	QS*	DP	
Z	0	1024	1024	0	0 DP
Z, QE	0	1609	1395	0	0 DP

Z, QE, ZOO	0	1975	1664	0	0 DP
Z, QE, ZOO, ZYLO	0	2247	1889	0	0 DP
Z, QE, ZOO, ZYLO, QUEUE	0	2493	2079	0	0 DP
Z, QE, ZOO, ZYLO, QUEUE, QUESTION	0	2682	2250	0	0 DP
Z, QE, ZOO, ZYLO, QUEUE, QUESTION, ZEROXANSWERS	0	2816	2365	0	0 DP
Z, QE, ZOO, ZYLO, QUEUE, QUESTION, ZEROXANSWERS, QUESTIONONANSWER	0	2935	2486	0	0 DP

*preprocessing is required for every pattern before search with respect to that pattern in using BM and QS algorithms, where as pre-processing phase is required only once in case of DP algorithm.

The time complexity of DP algorithm is directly proportional to the total number of occurrences of the first and the last characters of the pattern in the given input text.

5. CONCLUSION

We presented a Novel Devaki-Paul Multiple Pattern Matching Algorithm with a simple logic which is very easy to implement. We evaluated its performance with different texts and various set of patterns. The results were proved that the performance of the DP algorithm is greater than BM algorithm with 33%~91% and Quick Search algorithm with 37%~85%, in most of the cases. In case of unsuccessful search, the DP algorithm has zero character comparisons with irrespective of the size of the text and pattern, provided if either the first or the last character was not present in the given input text. In this case, the performance of the DP algorithm has been improved by 100%. The algorithm requires a pre-processing of the given input text only once before the search phase. It doesn't require further pre-processing phase for every pattern to search in the same given input text. The time complexity of the DP algorithm is directly proportional to the total number of occurrences of the first and the last characters of the patterns in the given input text.

6. ACKNOWLEDGMENTS

We thank Mr. R. S. Boyer and J. S. Moore for their research in single pattern matching. We thank also to the other experts/ researchers who have contributed their research towards the innovative ideas/algorithms in the area of pattern matching.

We thank our colleagues and others who contributed towards the development of this paper.

7. REFERENCES

[1] Devaki Pendlimarri and Paul Bharath Bhushan Petlu: "Novel Pattern Matching algorithm for Single Pattern Matching", (IJCSE) International Journal on Computer Science and Engineering, Vol. 02. No. 08. 2010, 2698-2704.

[2] Knuth D, J. Morris and V. Pratt: "Fast Pattern Matching in String", SIAM J. Computing 6(1977) 323-350.

[3] M. Fisk, and G. Varghese: "Fast content-based packet handling for intrusion detection", UCSD Technical Report CS2001-0670, May 2001

[4] U. Manber: "Finding Similar Files in a Large File System", USENIX Winter 1994 Technical Conference, San Fransisco (January 1994), pp 110.

[5] U. Manber and S. Wu: "GLIMPSE: A Tool to search through entire file systems", USENIX Winter 1994 Technical Conference, San Fransisco (January 1994).

[6] Wu S., and U. Manber: "Agrep – A Fast Approximate Pattern-Matching Tool", USENIX Winter 1992 Technical Conference, San Francisco (January 1992), pp 153 162.

[7] Wu S., and U. Manber: "Fast Text Searching Allowing Errors", Communications of the ACM 35 (October 1992), pp 83 91.

[8] R. S. Boyer and J. S. Moore: "A fast String Searching algorithm", Communications of the ACM, Vol 20, no. 10, pp. 762-772, 1977.

[9] C. Charras and T. Lecroq: "Exact string matching algorithms", http://www.igm.univ-mlv.fr/~lecroq/string/, 1997.

[10] M. Fisk and G. Varghese: "An analysis of fast string matching applied to content-based forwarding and intrusion detection", Technical Report CS2001-0670 (updated version), 2002.

[11] M. Crochemore, C. Hancart: "Pattern Matching in Algorithms and Theory of Computation Handbook", CRC Press Inc. Bocaaton, FL. 1999.

[12] Knuth, D. and J. Morris, V. Pratt, "Fast pattern matching in strings", SIAM J. Computing 6(1977) 323-350.

AUTHORS PROFILE

Pendlimarri Devaki, received MCA degree from S. V. University, Tirupati, India, in 2000. She is pursuing project of M. Tech. degree final semester in Computer Science and Engineering in Jawaharlal Nehru Technological University, Kakinada, India. Her main area of interests in research is intrusion detection, network security and data mining.

Paul Bharath Bhushan Petlu, received MCA degree from S. V. University, Tirupati, India, in 2000. He has been received the M. Tech. degree in Computer Science and Engineering from Sam Higginbottom Institute of Agriculture, Technology and Science – Deemed University, India in 2004. His main area of interests in research is network security and data mining.

Dr. Ramesh Babu Satrasala, received B.E. (Electrical) degree from Osmania University, Hyderabad, India in 1979. He also received M. Tech. (Plant Engineering and Management) degree from Jawaharlal Nehru Technological University (JNTU), Hyderabad in 1983. He received doctorate degree Ph. D. from Allahabad University, Allahabad, India in 2009. His main area of interests in research is intrusion detection and network security.