World Applied Sciences Journal 4 (5): 626-633, 2008
ISSN 1818-4952


# A New Approach for Solving String Matching Problem through Splitting the Unchangeable Text

[1]*Ibrahiem M.M. El Emary and* [2]*Mohammed S.M. Jaber*

[1]Faculty of Engineering, Al Ahliyya Amman University
[2]Faculty of Computer Science, Al Balqa Applied University, Al Salt Jordan

**Abstract:** This paper describes a new approach for solving string pattern matching problem with splitting unchangeable text in order to speed up the string pattern matching task The string matching problem consists of finding one or more generally all the exact or partial occurrences of a pattern P in a text T. So, this paper presents a new algorithm to solve the string matching problem. Application of the proposed algorithm assists in improving the search process of a specific pattern in a certain unchangeable text through decreasing the number of character comparisons. Operation concept of such an algorithm depends on reading the text and do two things: first split the text to n parts depending on the text size and in the same time construct n tables consisting of two columns; the first one is the words lengths exists in the text and the second one is the start positions of each word classified by the same length. After that, reading the pattern to obtain the pattern length and the pattern first character then the algorithm searches just in the words that consists of the same length of each table.

**Key words:** String matching T . Pattern matching P . n is the length of the word in the text . m is the length of the pattern

## INTRODUCTION

The purpose of this paper is to present a new algorithm used in solving the string matching problem to improve the search process depending on decreasing the number of character comparisons. The algorithm has two phases that works according to the following: first of all the text we are dealing with non-changeable text which means that the text is offline text. The first phase which is the preprocessing phase starts with reading and splitting the text to n equal parts depending on the size of the text and constructing n tables with two columns for each part of the text, the first one is the length of words and the second one is the start position for each word in the text. The start positions of the words will be classified by the same length. Once we have constructed the tables, we must sort them in ascending order using the length of the words as a key for sorting. This phase is done just only one time.

The second phase is searching for a specific pattern, so the algorithm calculates the number of the pattern characters and search for the same length in the tables starting from the first one, if the length is not exists in the first table then the algorithm search in the next one and so on. If the current table is the last table, then a message will appear denoting that the pattern is not exists. On the other hand if the length exists, then the algorithm will retrieve the words in the text using the stored start position in the table and begin to compare. If a match occurred and character comparisons equal the pattern length, then a full match is occurred since we are looking for an exact match, but if a mismatch happen at some character, then the algorithm will skip to the next start position and compare again. In this paper, we run some experiments using the Boyer-Moore algorithm because this algorithm and Boyer Moore Galil algorithm are considered the more fast algorithms theoretically and practically [1]. As a result of comparing our algorithm with others to full character comparisons at the same test paragraph, Boyer-Moore Algorithm has made 110 character comparisons to finish the text constructed from 91 words whereas the proposed algorithm have made 13 character comparisons on the same text.

To achieve the target of solving the matching problem using our new approach, we have presented this paper which is organized from various sections. Section two describes the problem formulation as well as the methodology of solution. Section three was devoted to the implementation phase of the proposed algorithm. Simulated results are shown in section four. In section five, we enclose the paper with conclusion and future works.

**Corresponding Author:** Dr. Ibrahiem M.M. El Emary, Faculty of Engineering, Al Ahliyya Amman University

## RELATED WORKS

A similar approach to our proposed one depending on making a table and a tree where the algorithm depending on visiting the nodes in the tree and the sub-trees is presented in this section. As known, using this kind of algorithms need recursion and more memory because the stack technique is used. Table 1 shows such a similar one algorithm that is described in [2]. The complexity of this algorithm is too expensive since the complexity of recursion is n(n-1)! [2].

Another algorithm dealing with non-changeable text is to make a table having the following information:

- Pattern
- Book number
- Page number
- Line number
- Word number

Illustration for this algorithm can be shown in Table 2.

This method save each word mentioned in a text or book as illustrated in the above table, when looking for a specific pattern, the algorithm returns the pattern positions based on the records stored. This method need O(n) to read the text and O(n) to search in the Table 12.

## PROBLEM FORMULATION & METHODOLOGY OF SOLUTION

The search process used in this area by the various algorithms start to search for the pattern despite of the existence of the pattern or not in the text. Also, the search process does not exclude any word from the text when start searching for the pattern. If we would like to search for another different pattern in the text, all algorithms except the presented one will start search again from the beginning for the new pattern. Also, most of other algorithms need to make a preprocessing on the text or on the pattern or on both each time for a new pattern.

The motivation to present this paper is to propose a new different algorithm in solving string matching problem to improve the search process through decreasing the number of character comparisons, which takes advantages from building an equal size tables for only one time consists of two columns for each, the first one is the words lengths exists in the text and the second one is the start positions of each word classified by the same length. The advantage from building more than one table is that if we find the needed pattern in the first table and the case was to find just the first occurrence, then we will not looking for the pattern in the other tables.

In view point of the tested data, there are a lot of cases that we may deal with in string matching process and pattern matching areas. These cases are summarized as following:

- The length of the text and the length of the pattern; each of them may be long or short.
- The text itself may be for educational purposes or normal text.
- The searching algorithms may search for a copy of the pattern or for a partial matching.
- Searching for a specific word may be needed just for the first occurrence or for accurate number of occurrences or for all occurrences.
- The texts could be changeable or non changeable.
- The text has capital letters and small letters

The case that we want to work with for the text is: normal English, non changeable text and normal pattern. A normal text likes one presented in this paper or a text from the internet for example. This algorithm is searching for a copy of the pattern not for a partial match in small letter. The pattern is a normal word in small letter also.

In view point of test environment the experiments was run on Pentium 3 of 450 Mhz clock with 128 Mb RAM and a 20 GB local hard disk. The operating system is Windows XP during all experiments; the data structures used in the testing were all in physical memory during the experiments. Finally, the algorithm has been implemented in Visual Basic programming language.

For the comparison of the string matching algorithms with the proposed algorithm, we have used the number of character comparisons. The counting of the number of character comparisons is the same as that used by Smith, which is based on computing the number of actually compared characters to the number of passed characters in the text. Since all algorithms are designed to find all occurrences of a pattern in the text in the experiments, the number of passed characters is always n-m+1. The operation steps of the proposed algorithm to search for a specific pattern in a certain text are shown in Fig. 1.

## IMPLEMENTATION PHASE OF THE PROPOSED ALGORITHM

The idea of searching about a certain pattern in details is given below which starts when the user writes a pattern to be searched in the text. Implementation of the algorithm requires various phases given by:-

**Preprocessing phase:** In this phase, the algorithm splits the text to n equal parts, let's says four parts and

Table 1: Other Similar approach of the proposed algorithm

| Antecedent | POS | Label | Count | Description |
|---|---|---|---|---|
| NP | NP | * | 18,334 | NP trace (e.g., Sam was seen*) |
| | NP | * | 9,812 | NP PRO (e.g., *to sleep is nice) |
| WHNP | NP | *T* | 8,620 | WH trace (e.g., the woman who you saw *T*) |
| | | *U* | 7,478 | Empty units (e.g., $ 25 *U*) |
| | | 0 | 5,635 | Empty complementizers (e.g., Sam said 0 Sasha snores) |
| S | S | *T* | 4,063 | Moved clauses (e.g., Sam had to go, Sasha explained *T*) |
| WHADVP | ADVP | *T* | 2,492 | WH-trace (e.g., Sam explained how to leave *T*) |
| | SBAR | | 2,033 | Empty clauses (e.g., Sam had to go, Sasha explained (SBAR)) |
| | WHNP | 0 | 1,759 | Empty relative pronouns (e.g., the woman 0 we saw) |
| | WHADVP | 0 | 575 | Empty relative pronouns (e.g., no reason 0 to leave) |

Table 2: Similar approach of the proposed algorithm

| Pattern | Book number | Page number | Line number | Word number |
|---|---|---|---|---|
| Recursion | 1 | 101 | 4 | 7 |
| | 2 | 220 | 7 | 3 |
| | 2 | 5 | 2 | 2 |
| | 4 | 114 | 5 | 5 |
| Recursive | | | | |
| Recurred | | | | |
| Recur | | | | |

constructs four tables each one has two columns, the first one is the length of words starting from one character and ending with the most long word length of characters in the text. The second column is the start position of each word in the text classified by the same length. Then the algorithm stores the lengths and the start positions for each word in the text in each table for each part of the text. These tables are constructed only one time since the text is non changeable.

As an illustration for the reading process we may take a case study to explain this idea as follows:-

*"the author name is mohammed sulieman mohammed jaber i am one of the alblaqa*
*1 5 12 17 20 29 38 47 53 54 57 61 64 68*
*applied university faculty i have born in kuwait and i have finished the bachelor's*
*76 84 95 103 104 109 114 117 123 127 128 133 142 146*
*degree from princess sumaiya university for technology i have the master degree from*
*157 164 169 178 186 197 201 212 214 219 223 230 237*
*albalqa applied university i have finished the master degree last year"*
*242 250 258 269 271 276 285 289 296 303 308*

Assuming that each line of this paragraph forming one part of the text, so we have four parts and each part will have one table, the tables will be as following:

Table 3: P art 1 of the text

| Length | Start position |
|---|---|
| 1 | 53,103 |
| 2 | 17,54,61 |
| 3 | 1,57,64 |
| 4 | 12 |
| 5 | 47 |
| 6 | 5 |
| 7 | 68 |
| 8 | 20,29,38 |

Table 4: Part 2 of the text

| Length | Start position |
|---|---|
| 1 | 103,127 |
| 2 | 114 |
| 3 | 123,142 |
| 4 | 104,109,128 |
| 6 | 117 |
| 7 | 76,95 |
| 8 | 133 |
| 10 | 84,146 |

Table 5: Part 3 of the text

| Length | Start position |
|---|---|
| 1 | 212 |
| 3 | 197,219 |
| 4 | 164,214,237 |
| 6 | 157,223,230 |
| 7 | 178 |
| 8 | 169 |
| 10 | 186,201 |

Table 6: Part 4 of the text

| Length | Start position |
|---|---|
| 1 | 269 |
| 3 | 285 |
| 4 | 271,303,308 |
| 6 | 289,296 |
| 7 | 242,250 |
| 8 | 276 |
| 10 | 258 |

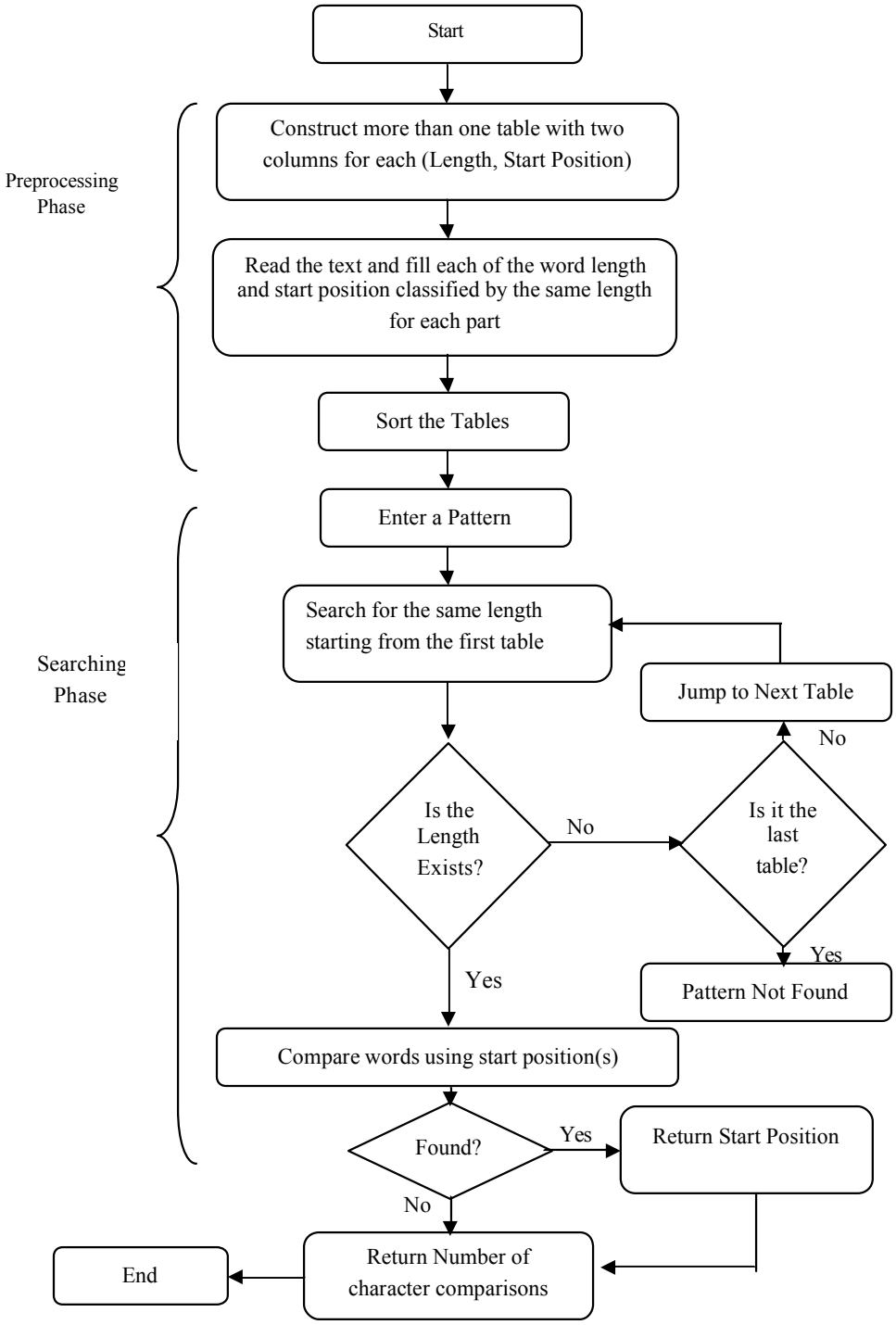


Fig. 1: A flowchart of the proposed algorithm

Once the algorithm constructs the above tables, then we have finished the preprocessing phase and ready to search for any word in the text. These tables are constructed just only one time since the text is unchangeable. We will deal with two cases here, the first one is if the pattern length is not exists in the text and the second one if exists.

**Case number one:** If the user of the text try to look for a word that have nine characters, then the algorithm will start with the first table using the length of the pattern as a key in the length column, but the algorithm will skip to the second table because there is no length equal nine, also the algorithm will skip to the third and the fourth table for the same reason, then the algorithm

will return a message denoting that the pattern is not exists in the text without searching the text at all.

**Case number two:** First of all, if the pattern length exists in any table then the algorithm will ignore the rest rows of other lengths in the table and focus just on the row of the same pattern length that we look for. The algorithm starts comparing the first character of the pattern with the first character of the first word by using the start position which stored in the table, since we store the first character and the length of the pattern from the beginning. If the first character is not matched, then the algorithm ignores the current word and starts with the second one. If the first character is matched, then the algorithm continues to compare the rest of word characters. In case a mismatch occurred with some character after that, then the algorithm ignore the current word and skip to the next start position with the next word.

But in case of full matching occurred, the algorithm will return the start position of the word matched and continue to search for other occurrences until the end of the row for the same length in all the tables if the case was to find all occurrences in the text. For example, if the user look for the word "shift" which consists of five characters, as we can see in the above example, only table number one has the length five so the algorithm will start comparing with the start position number 47 and the result will be just 1 comparison between the character "j" and the character "s" then the algorithm will skip to the next table and search for the length equal five but it is not exists and the third and the fourth table are the same then the algorithm will return a message denoting that no match exists. If we want to look for the word "jaber", so the whole comparisons done is just five characters which is the best case.

**Complexity and analysis:** In the preprocessing phase, reading the text from beginning to the end will take a complexity of O(n). To sort the records in the tables, we may use the merge sort which have complexity of O(nlogn). So the overall complexity in the preprocessing phase is O(n*nlogn) which will be executed just only one time in the non changeable text, so complexity of the first part is not so important, because it is executed only once [3].

In the search phase, as a technique to search for the pattern in table I, suggest to use the Binary Search algorithm since the complexity for the binary search algorithm takes O(Logn). The number of comparisons done in each row is not fixed, we may compare with zero or one word or comparing with thirty words. Then the whole complexity for the searching phase is O (logn+Σ) wher Σ is the number of character comparison that is done in each row, the worst case.

## SIMULATED RESULTS

Now, we will make a comparison between the proposed algorithm one of the most famous and fastest algorithm in such area which is the BM algorithm to find out the improvement in the number of character comparisons that is done in each algorithm. As an example, we'll take the next paragraph to apply the algorithm with the patterns; "comparisons", "subproblem" and "algorithm". The pattern "comparisons" is not exists in the text but the others are exists.

*A recursive algorithm for solving a problem is an algorithm that works by dividing*
*1 3 13 23 27 35 37 45 48 51 61 66 72 75*

*the problem into several problems of a smaller size and by applying the same*
*84 88 96 101 109 118 121 123 131 136 140 143 152 156*

*algorithm to at least one of the smaller problems. Applying the same algorithm*
*161 171 174 177 183 187 190 194 202 212 221 225 230*

*to a subproblem of a problem is called a recursive step. Size of a*
*240 242 244 255 257 259 267 270 277 279 289 295 300 303*

*problem may mean different things for different problems, for instance the number*
*305 313 317 322 332 339 343 353 363 367 376 380*

*of elements to sort for a sorting problem, or the number whose factorial*
*387 389 398 401 406 410 412 420 429 432 436 443 449*

*we need to compute for a problem of computing a factorial".*
*459 462 467 470 478 482 484 492 495 505 507*

**Preprocessing phase:** First of all, the algorithm reads the whole text and then splitting it equally to many parts, if we assume that each two lines of the previous paragraph are one part, so the tables will be as following:

**Implementation phase:** For the first pattern "comparisons", we can see that this pattern constructed

Table 7: Part 1 of the text

| Length | Start position |
|---|---|
| 1 | 1,35,121 |
| 2 | 45,48,72,118,140 |
| 3 | 23,84,136,152 |
| 4 | 61,96,131,156 |
| 5 | 66 |
| 6 | - |
| 7 | 27,37,88,101,123 |
| 8 | 75,109,143 |
| 9 | 3,13,51 |
| 10 | - |

Table 8: Part2 of the text

| Length | Start position |
|---|---|
| 1 | 242,257,277,303 |
| 2 | 171,174,187,240,255,267,300 |
| 3 | 183,190,221 |
| 4 | 225,289,295 |
| 5 | 177 |
| 6 | 270 |
| 7 | 194,259 |
| 8 | 202,212 |
| 9 | 161,230,279 |
| 10 | 244 |

Table 9: Part3 of the text

| Length | Start position |
|---|---|
| 1 | 410 |
| 2 | 387,398,429 |
| 3 | 313,339,363,376,406,432 |
| 4 | 317,401 |
| 5 | 443 |
| 6 | 332,380,436 |
| 7 | 305,412,420 |
| 8 | 353,367,389 |
| 9 | 322,343,449 |
| 10 | - |

Table 10: Part4 of the text

| Length | Start position |
|---|---|
| 1 | 482,505 |
| 2 | 459,467,492 |
| 3 | 478 |
| 4 | 462 |
| 5 | - |
| 6 | - |
| 7 | 470,484 |
| 8 | - |
| 9 | 495,507 |
| 10 | - |

from eleven characters, when the algorithm consult the first table in the length column, the algorithm will find that there is no such length, then the algorithm will skip to the next tables for the same reason and finally the algorithm will return a message denoting that no such pattern exists in the text. Here, the number of character comparisons is done equals zero. For the pattern "subproblem", this pattern mentioned just only one time in the text, the algorithm will skip from Table 7 and 8 since there is no such length exists. In Table 8, there is one start position denoting that there is one word exists with the same length which is the same word. The total comparisons done equal to ten comparisons since there is no such length in Table 9 and 10.

For the last pattern "recursive", when the algorithm consults the first table, the algorithm will focuses on the row with length equal nine since the word (recursive) is constructed from nine characters. Then, the algorithm will start comparing with the word at the start position 3. The first character is matched, then the algorithm will continue to compare until a full matching which is the best case. The algorithm continues to search for other occurrences starting from position 13, but the first characters is not matched and then skip to the start position 51 and so on in the other tables.

The total number of comparisons done equal to twenty seven comparisons, eighteen comparisons for the word (recursive) where this pattem exists twice in the text and nine comparisons between the letter (r) in recursive and the first character of the words for the same length in all tables. If the case is to find the first occurrence only, then the total comparisons equal to nine comparisons. If the case is to find all occurrences then the total comparisons equal to 27.

If we apply the Boyer-Moore algorithm on the same paragraph, then the results will be as following: For the first pattern "comparisons", which is not exists in the text at all, the algorithm will go on to compare until the end of the text, the total number of comparisons that is done equals 55. For the pattern "subproblem", the total number of comparisons done equals 110. For the last pattern "recursive", the total number of comparisons done equal 81.

**Comparative results:** As a comparative illustration between the proposed algorithm and other algorithms relative to the character comparisons, we show the output results as shown in the next Fig. 2 and 3 for the last word of the last example, the word “subproblem” which indicates that the algorithm is better than others in view point of number of comparisons done on each pattern need to be searched within a certain text.

As a comparative result for the word “subproblem” witch consists of 10 characters, the result shows that

Table 11:Number of character comparisons using BM and the proposed Algorithms for each of the patterns “Comparisons”, “subproblem” and "recursive"

| | Boyer-moore algorithm | | Proposed algorithm | |
|---|---|---|---|---|
| Word | First occurrence | Full occurrences | First occurrence | Full occurrences |
| Comparisons | 55 | - | 0 | 0 |
| Subproblem | 53 | 110 | 10 | 13 |
| Recursive | 10 | 81 | 9 | 27 |

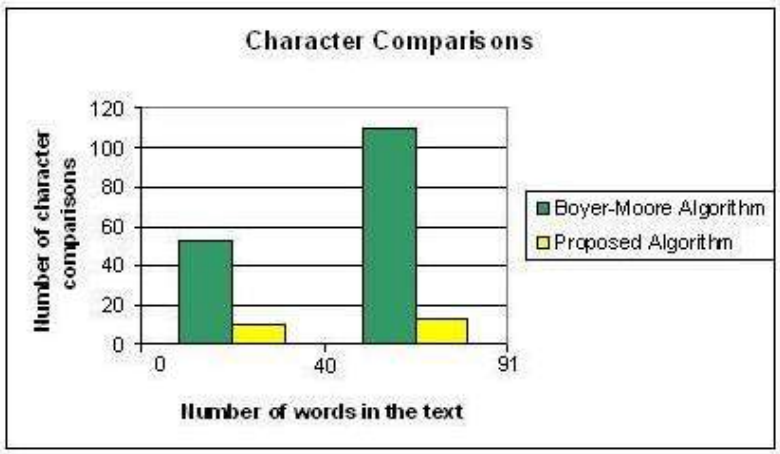


Fig. 2: Character comparison between the proposed algorithm and BM algorithm in view point of number of character comparisons(Y axis) against Number of words in the text (X-axis) for the word “subproblem”

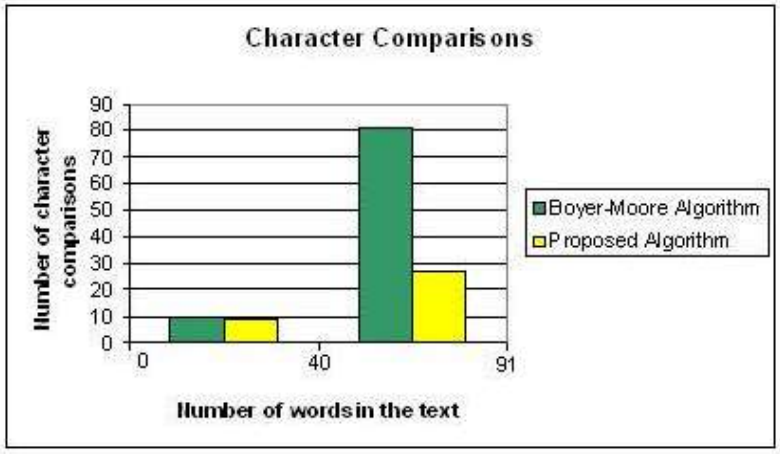


Fig. 3: Character comparison between the proposed algorithm and BM algorithm in view point of number of character comparisons (Y axis) against Number of words in the text (X-axis) for the word “recursive”

after 40 words, the number of character comparisons done for the proposed algorithm is 10 comparisons and 53 comparisons for Boyer Moore and 13 comparisons for the proposed algorithm and 110 comparisons for Boyer Moore after 91 words in the text.

For the pattern recursive which consists of 9 characters, the result shows that after 40 words, the number of character comparisons done for the proposed algorithm is 9 comparisons and 10 comparisons for Boyer Moore and 27 comparisons for the proposed algorithm and 81 comparisons for Boyer Moore after 91 words in the text.

## CONCLUSION

Decreasing the searching time to locate a specific word could give us a good improvement in pattern matching problem. The main advantages that we may

gain from this algorithm are: excluding the search for not needed text, Make no search at all if the pattern length does not exist and searching for different pattern without need to read the text again. We may look for the average case of the words length and then dealing with this case in different way to increase the efficiency of the search technique. This algorithm is done for the non-changeable text, because the time needed and space used to save the length and the start position. As seen, this algorithm construct tables having the lengths and start position for each word classified by the same length, if we have a huge text, then we will have a huge table which need more time to construct. As a future work, we may compress these tables to reduce the space used. Also, we may use this algorithm at the text editors. Constructing the tables as soon as finishing writing the text will reduce the time for constructing the tables and then the algorithm will be ready to search for any word in the text at any time.

## REFERENCES


1. Michailidis, P.D. and K.G. Margaritis, 2001. On-line String Matching Algorithms: Survey and Experimental Results. International Journal of Computer Mathematics, 76 (4): 411-434.
2. Mark Johnson, 2002. A simple pattern-matching algorithm for recovering empty nodes and their antecedents. Philadelphia.
3. Tomas Pet, 2005. Aho-Corasick string matching in C#.

00. Sara Baase and Allen Van Gelder, 2003. Computer Algorithms, Introduction to Design and Analysis. Third Edition.
00. Speeding Up String Pattern Matching by Text Compression, 2001. Masayuki Takeda, Yusuke Shibata, Tetsuya Matsumoto, Takuya Kida, Ayumi Shinohara, IPSJ Journal, Vol: 42 (3).
00. Ute Abel Proseminar, 2001. String matching Algorithms.
00. Robert Muth and Udi Manber1, Approximate Multiple String Search, Department of Computer Science. University of Arizona, Tucson, AZ 85721.
00. Tomas Pet, 2005. Aho-Corasick string matching in C#.
00. Fast Partial Evaluation of Pattern Matching in Strings, 2003. Mads Sig Ager Olivier Danvy, Henning Korsholm Rohde.
00. Improved Approximate Pattern Matching on Hypertext, Gonzalo Navarro, Dept. of Computer Science. Univ. of Chile Blanco Encalada 2120, Santiago, Chile.
00. Robert Muth and Udi Manber, 2003. Approximate multiple string search. Department of Computer Science. University of Arizona.