Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 79 (2016) 161 – 169

Procedia Computer Science

7th International Conference on Communication, Computing and Virtualization 2016

Intelligent Predictive String Search Algorithm

Dipendra Gurung[a]*, Udit Kr. Chakraborty[b], Pratikshya Sharma[c]

[a,b,c]*Sikkim Manipal Institute of Technology, Majhitar, Sikkim, India*

Abstract

Text processing has moved beyond merely word processing and desktop publishing softwares, which currently form a miniscule part of its application. With the advent of the internet and the huge amount of text processing associated with information mining, string search algorithms have gained importance. In this paper a new string searching algorithm is presented that uses intelligent predictions based on text features to search for a string in a text. The proposed algorithm has been developed after analyzing the existing algorithms such as KMP, Boyer-Moore and Horspool. One unique feature of this algorithm is that unlike the existing algorithms, it does not require pre-processing the pattern to be searched. As a result it does not incur the overhead required in pre-processing the pattern. The algorithm searches through a given text to find the first occurrence of a pattern. It does not involve complex computations and uses simple rules during a match or mismatch of a pattern character. Based on the variety of applications coming up in areas of data and information mining, sentiment analysis, DNA pattern matching etc, this simple, elegant and intelligent algorithm will find its application.

Peer-review under responsibility of the Organizing Committee of ICCCV 2016

Keywords: String search; Predictive search

1. Introduction

The term text processing in computing refers to the discipline of automating the creation or manipulation of textual data. The processing may include use of computer algorithms to restructure or reformat the text, extract information and store the data as files on a computer[1]. Text processing carries a lot of importance as a lot of digital

* Corresponding author. Tel.: +91-9733181860.
E-mail address: gurungdipendra99@gmail.com

Peer-review under responsibility of the Organizing Committee of ICCCV 2016
doi:10.1016/j.procs.2016.03.116

information basically come down to a collection of text: configuration files, log files, etc. Moreover the internet provides information largely in the form of text.

One of the basic steps in text processing is word searching or word matching. A word search algorithm works by finding the first or all the occurrences of a word in a textual data. The word to be searched is generally called a pattern. Words alone provide valuable information for further processing and hence word search is an important component of text processing tasks like text editing, data retrieval and data manipulation[6]. Over the years word search has found immense application in text editors, web search and searching for patterns in biological databases. In the recent times sentiment analysis, online advertisements are some areas in which word search finds its use.

A word search algorithm takes a text T of length n and a pattern P of length m as the input. The text is then scanned using a window that has length equal to the size of the pattern. The leftmost ends of the pattern and window are aligned. The brute force method works by comparing each character of the pattern with that of the text and in case of a mismatch the pattern is shifted by one position to the right. Other existing algorithms generally work in two phases:- the pre-processing phase and the matching phase. The pre-processing phase is used to determine the number of positions by which the pattern needs to be shifted in case of a mismatch in the matching phase. The main goal of string matching algorithms is to increase efficiency by reducing the number of comparisons and increase the length of shifts in case of a mismatch. The issue of efficiency of string search algorithms has probably never been considered so seriously until the virtual text explosion caused by the internet and the task of mining valuable information from it. As a variety of tasks are presenting themselves different techniques, each efficient in its own specific area are being utilized.

The rest of the paper is organized as follows. Section 2 discusses about some previously existing algorithms. Section 3 presents the proposed algorithm. Section 4 presents an example for pattern searching using the proposed algorithm. Section 5 presents the experimental results of the proposed algorithm. Section 6 presents an analysis of the proposed algorithm and finally the paper is concluded in Section 7.

2. Survey of Existing Algorithms

2.1. Boyer Moore Algorithm

The Boyer Moore algorithm is one of the most extensively used pattern matching algorithms. All the algorithms prior to it attempted to find a pattern in a string by examining the leftmost character. Boyer and Moore believed that more information could be gained by beginning the comparison from the end of the pattern instead of the beginning[8]. This information often allows the pattern to proceed in large jumps through the text being searched[2]. The algorithm uses the bad character heuristic and the good suffix heuristic to determine the pattern shift in case of mismatch of a pattern character.

During the matching phase if there is a mismatch between the text character T[i] and the pattern character P[j] and if T[i] does not occur anywhere else in the pattern, then the pattern can be shifted completely by m positions towards the right. If T[i] is present in the pattern then the pattern is shifted until an occurrence of T[i] in the pattern gets aligned with T[i] of the text. This is the bad character heuristic.

The second type of shift is guided by a successful match of the last k>0 characters of the pattern, P[j...m] and corresponding characters, T[i...(i+k)] of the text. P[j…m] is referred to as the suffix of size k of the pattern and is denoted as suff(k). If there is no occurrence of suff(k) in the pattern then it is shifted by its entire length. However if there exists a prefix (beginning part of the pattern) of size l<k that match suffix of the same size l then the pattern is shifted by a distance equal to the distance between the prefix and the suffix. On the other hand if there is another occurrence of suff(k) not preceded by the same character that caused the mismatch then the pattern is shifted by a

distance equal to suff(k) and its rightmost occurrence[9]. This is the good suffix heuristic. The shift distance is taken to be the maximum of the distances obtained by the bad character heuristic and the good suffix heuristic.

The Boyer Moore algorithm is considered to be an efficient algorithm for pattern searching. It has the property that the longer the pattern is, the faster it performs. However the algorithm suffers from the phenomenon that it tends to work inefficiently on small alphabets like DNA. The skip distance tends to stop growing with the pattern length because substrings re-occur frequently[14]. Also, the pre-processing for the good suffix heuristic is difficult to understand and implement[10]. Furthermore, it suffers from the need for very large tables or state machines and thus requires extra space[14]. It also requires extra time for processing the pattern.

2.2. Horspool Algorithm

The Horspool algorithm also begins the comparison from the end of the pattern but unlike the Boyer Moore algorithm it only uses the bad character heuristic. Since the good suffix heuristic is complicated and difficult to implement, Horspool suggested that using only the bad character heuristic would also give performance similar to that of the Boyer-Moore algorithm. The Boyer Moore algorithm used the bad character of the text that caused a mismatch to determine the pattern shift distance. On the contrary Horspool's bad character heuristic uses the rightmost character of the current text window. During the matching phase, if T[i] and P[j] do not match and T[l] is the rightmost character of the current text window then the pattern is inspected to find the rightmost occurrence of T[l] in it. If no occurrence of T[l] exists in P, the pattern is shifted completely by its length m, otherwise the pattern is shifted until T[l] gets aligned to its rightmost occurrence in P.

The Horspool algorithm is a refinement of the Boyer Moore algorithm. Since it uses only the bad character heuristic, it requires less space but has a poorer worst case performance[11]. Like the Boyer Moore algorithm, the Horspool algorithm gets faster for longer patterns. However for shorter patterns the naïve algorithm is considered to be better[12].

2.3. KMP Algorithm

The Knuth Morris Pratt or the KMP algorithm begins the comparison from the leftmost character of the pattern. The following example explains the algorithm.

0	1	2	3	4	5	6	7	8
A	B	C	A	B	C	A	B	D
A	B	C	A	B	D			
			A	B	C	A	B	D

Fig. 1 KMP algorithm example[10]

At the first attempt the characters through position 0-4 or the prefix ABCAB of the pattern have matched. Comparison C-D at position 5 yields a mismatch. In order to determine the shift of the pattern let us define the term border. A border of a string is a substring that is both proper prefix and proper suffix of the string. In the above example the border of the matching prefix ABCAB is AB. The width of the prefix and its border is 5 and 2 respectively. The shift distance is determined by the difference between the width of the matching prefix and its border, which is 3[10]. The pattern is shifted by three positions towards the right. This shift aligns the pattern with its occurrence in the text.

The KMP algorithm makes use of the information gained by previous character comparisons unlike the naïve algorithm. Hence it never needs to move backwards in the text, this makes the algorithm useful for processing large files[13]. However the performance of the KMP algorithm degrades for longer patterns as the possibility of character mismatch increases.

The algorithms discussed above and their variants have been in use in computer systems. However, these algorithms have their disadvantages either in terms of time and space requirements or the size of the pattern. As a result the algorithms fail to achieve the desired performance in certain applications or situations as mentioned previously. The proposed algorithm attempts to overcome these disadvantages and find its application in areas in which the discussed algorithms fail to perform.

3. The Proposed Algorithm

The proposed algorithm finds the first occurrence of a pattern in a text that consists of words separated by a blank space. It does not require pre-processing the pattern to be searched and aims to search for a pattern by using features of the text. It begins the matching by aligning the leftmost ends of the text and the pattern. The leftmost characters are compared for a match. If there is a match the rightmost character of the pattern is compared with the rightmost character of the current window. If it matches the order of comparison of the remaining characters is from right to left. In case of a mismatch the algorithm uses two rules to make a shift namely alphabet-blank mismatch and alphabet-alphabet mismatch.

An alphabet-blank mismatch during the comparison of the leftmost character indicates that the next position might be a probable beginning of the pattern. As a result the pattern is shifted by one position towards the right. In case of an alphabet-alphabet mismatch during the comparison of the leftmost characters, the pattern is shifted by two positions towards the right because the character at the next position might either be a blank or a character that is a part of the current word to which the pattern is aligned.

During the comparison of the rightmost character of the pattern an alphabet-blank mismatch indicates that the pattern is not present at the current position as clearly the current word aligned is shorter than the pattern. As a result the pattern is shifted by (i+(m-1)+1) positions towards the right. Here ‘i’ indicates the starting position of the current text window. The value of i starts at 0. In case of an alphabet-alphabet mismatch, the algorithm checks for the character at the next position. If it is a blank the pattern is shifted by (i+(m-1)+2) positions towards the right. If it is a character the pattern is shifted by two positions towards the right.

When a mismatch occurs at any position other than the leftmost and the rightmost positions, the pattern is completely shifted by m positions towards the right.

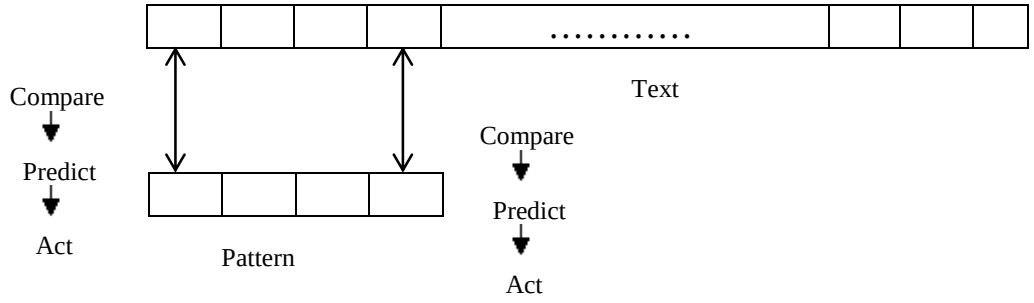

Fig. 2 Schematic diagram for the proposed algorithm

Figure 2 given above shows a schematic diagram for the proposed algorithm. The figure also depicts the steps that the algorithm takes at each step i.e. compare the characters of the text and the pattern, predict accordingly the next step to be taken in case of a mismatch and act as per it.

The following specifies the steps of the proposed algorithm. P is the pattern of length m to be searched in a text T of length n. The notation T[i] is used to denote the i^{th} character in T. The value of i begins with 0. The algorithm returns the position of the first occurrence of P in T, if P is present in T.

Algorithm_preditive_search(T,P)

```
1: Align T and P
2: Repeat steps 3 to 5 until a match is found or until the end of T is reached.
3: Compare P[0] and T[i]
4: if mismatch
        a: Alphabet-blank
                Re align P[0] to T[i+1]
        b: Alphabet-alphabet
                Re align P[0] to T[i+2]
5: if match compare P[m] to T[i+m]
        a: if match
                Compare remaining characters from right  to left
        b: if mismatch
                1: Alphabet-blank
                        Re align P[0] to T[i+(m-1)+1]
                2: Alphabet-alphabet
                        a: if T[i+(m+1)] is blank
                                Re align P[0] to T[i+(m-1)+2]
                        b: else
                                Re align P[0] to T[i+2]
```

4. Working Example

The following example shows the steps during the search of the pattern JOLLY of length 5 in a text of length 18.

J	O	H	N		I	S		J	O	L	L	Y
J	O	L	L	Y								
					J	O	L	L	Y			
							J	O	L	L	Y	
								J	O	L	L	Y

Fig. 3 Pattern search using the proposed algorithm

The leftmost comparison J-J yields a match. Therefore the rightmost character of the pattern is tried for a match, as this leads to an alphabet-blank mismatch, the pattern is shifted by its length. In the next stage the comparison I-J

causes a mismatch. According to the rule the pattern is shifted by two positions. The next comparison causes an alphabet-blank mismatch. As a result the pattern is shifted by one position.

The number of comparisons made up to this stage is four. The next shift of the pattern aligns the character J of the pattern to J of the current window. As this leads to a match, the rightmost character of the pattern is inspected. This too leads to a match. The remaining characters of the pattern also match with that of the current window. The algorithm then terminates and returns the index where the occurrence of the pattern starts in the text. For the above example the index is 8. Since all the characters of the pattern are compared, the number of comparisons finally made is nine.

5. Experimental results

The proposed algorithm was tested on a number of test cases. The following shows some cases in which the pattern is located at the beginning, somewhere in the middle and at the end of a text. The number of comparisons that the algorithm makes, the number of attempts made is also shown. The following shows a test case in which the pattern occurs at the beginning of the text.

J	A	C	K		A	N	D		J	I	L	L
J	A	C	K									

Fig. 4 Comparisons made when pattern is at the beginning

The matching starts by comparing J of the pattern JACK with the first character of the current text window. This leads to a match. The last character of the pattern K is next examined. As this too leads to a match the remaining characters are compared. Every character of the pattern matches with that of the current word to which it is aligned. The number of comparisons made is 4.

The sequence of steps that the algorithm takes when the pattern is present at the end of the text is shown next.

C	A	T	Y		I	S		C	U	T	E
C	U	T	E								
					C	U	T	E			
							C	U	T	E	
								C	U	T	E

Fig. 5 Comparisons made when pattern is at the end

The first comparison C-C leads to a match. As per the order, the next comparison is Y-E. As this leads to an alphabet-alphabet mismatch and the next character is a blank space, as a result the pattern is shifted by its length plus one positions i.e. by 4 + 1 = 5, positions towards the right. The next comparison is I-C, this causes a mismatch. Thus the pattern is shifted by two positions. As the next comparison leads to an alphabet blank mismatch, the pattern is shifted by one position. This alignment causes a match of the pattern. The total number of comparisons made is 8.

The next two cases depict scenarios when the pattern is present somewhere within the text.

Considering the test case shown in figure 6 below. The first comparison I-H causes a mismatch. Thus the shift is made by two positions. The next comparison leads to an alphabet-blank mismatch, so the shift is made by one position. The comparison I-H causes a mismatch and thus a shift of two positions is made. The next comparison leads to an alphabet-blank mismatch and the resulting shift aligns the pattern HOT with its occurrence in the text. The number of comparisons made is 7.

I	T		I	S		H	O	T		A	N	D		D	R	Y
H	O	T														
		H	O	T												
			H	O	T											
					H	O	T									
						H	O	T								

Fig. 6 Comparisons made when pattern is in the middle

P	Y	T	H	O	N		I	N		E	I	G	H	T	I	E	S
I	N																
		I	N														
				I	N												
						I	N										
							I	N									

Fig. 7 Comparisons made when pattern is in the middle

For the test case shown in figure 7 above, the first comparison P-I causes an alphabet-alphabet mismatch, so the pattern is shifted by two positions. The next comparison T-I also causes an alphabet-alphabet mismatch and thus the shift is made accordingly. Same is the case for the comparison O-I. The pattern is finally aligned to its occurrence in the text as a result of the shift caused by the alphabet-blank mismatch in the second last attempt. The total number of comparison made is 6.

6. Analysis of the Algorithm

The algorithm finds for a match by making predictions as to what would occur at the next position in the text. The predictions are made on the basis of whether the character that caused a mismatch is a blank or an alphabet. At each mismatch the algorithm shifts the pattern by one or two positions depending on whether it is a blank or an alphabet mismatch. In the best case the algorithm shifts the pattern by its entire length when the rightmost character of the pattern coincides with a blank character in the text or the character present next to the last character of the current window is a blank. Unlike the traditional pattern matching algorithms, the proposed algorithm works in just a single phase i.e. the matching phase and hence does not require additional time for pre-processing. The following example shows the steps in finding the pattern using the Boyer-Moore algorithm for the example considered in figure 3.

J	O	H	N		I	S		J	O	L	L	Y
J	O	L	L	Y								
					J	O	L	L	Y			
								J	O	L	L	Y

Fig. 8 Pattern search using the Boyer Moore Algorithm

The Boyer-Moore Algorithm finds a match in the third attempt as seen in figure 4. It pre-processes the pattern JOLLY to determine the shifts in case of a mismatch. The comparison starts with the rightmost character of the pattern. As it can be seen from figure 4, this leads to a mismatch and since the pattern does not contain a blank space it is completely shifted. The next comparison O-Y also causes a mismatch but there is an occurrence of O in the pattern, hence the pattern is shifted such that O in the text in aligned with O in the pattern. The number of comparison made up to this stage if 2. The next comparison Y-Y leads to a match and so do the subsequent comparisons. The number of comparisons finally made is 7.

In Section 4, it was seen that for the same test case the number of comparisons made by the proposed algorithm was 9 and the pattern was found in the fourth attempt. This was achieved without the pattern being processed before the comparisons were made.

Furthermore, for the test case shown in figure 7 in Section 5, the following observations were made when the Boyer Moore algorithm was used.

P	Y	T	H	O	N		I	N		E	I	G	H	T	I	E	S
I	N																
		I	N														
				I	N												
						I	N										
							I	N									

Fig. 9 Pattern search using the Boyer Moore Algorithm

The first comparison Y-N causes a mismatch and as Y does not occur anywhere else in the pattern, the pattern is shifted by its length. The next comparison H-N also causes a mismatch and since H is not present anywhere else in the pattern, it is shifted by its length. The number of comparisons made up to this stage is 2. The next comparison N-N leads to a match, hence the comparison O-I is made. This leads to a mismatch and so the pattern is shifted by its length. The number of comparison made up to this stage is 4. The comparison I-N causes a mismatch but I is present in the pattern. As a result the pattern is shifted such that I in the text that caused the mismatch is aligned with the I in the pattern. The number of comparisons made till now is 5. The next two comparisons N-N and I-I finds the occurrence of the pattern in the text. The total number of comparisons thus made is 7.

It was seen in Section 5 that for the same test case, the number of comparisons made by the proposed algorithm was 6.

Thus, from the above two observations we can say that for shorter patterns, the number of comparisons that the proposed algorithm makes to find a match is comparable to that of the Boyer-Moore Algorithm and in some cases its performance is even better. Moreover, the algorithm also does not incur an additional overhead to pre-process the pattern.

7. Conclusion

The proposed algorithm finds the first occurrence of a pattern in a text that consists of words separated by a blank space. One area in which it scores over other existing algorithms is that it does not require pre-processing but rather it uses intelligent predictions to find a match. The algorithm has been tested and it is noteworthy that for shorter patterns the results achieved are comparable and in some cases better than that of the Boyer Moore algorithm which is considered to be a benchmark algorithm for pattern matching. It is also important to note that, unlike the Boyer Moore algorithm, the proposed algorithm does not require complex computations, neither does it require extra space

and processing time like other algorithms discussed in this literature. As previously mentioned, the Boyer Moore and the Horspool algorithm fail to achieve the desired performance for shorter patterns. On the other hand although the KMP algorithm works well for shorter patterns, it is far more complex than the proposed algorithm. The proposed algorithm can possibly be implemented in applications that require exact pattern matching and involve search of short patterns.

References

1. Joseph B. Sidowski, On-line computer text processing: A tutorial in: Behavior Research Methods & Instrumentation, vol. 6, no.2, pp. 159-166, 1974.
2. Boyer, R. S.; Moore, J. S. A fast string searching algorithm. Commun. ACM 20, pp. 762-772, 1977.
3. Donald E. Knuth, James H. Morris, Vaughan R. Pratt, Fast Pattern Matching In Strings in: Siam, Vol. 6, No. 2, pp. 323-350, June 1977.
4. R. Nigel horspool, Practical fast searching in strings in: Software-Practice and Experience, vol. 10, pp. 501-506, 1980.
5. Timo Raita, Tuning The Boyer-Moore-Horspool String Searching Algorithm in: Software-Practice And Experience, Vol. 22(10), pp. 879-884, October 1992.
6. Nimisha Singla, Deepak Garg, String Matching Algorithms and their Applicability in various Applications in: International Journal of Soft Computing and Engineering (IJSCE) ISSN: 2231-2307, Volume-I, Issue-6, pp.156-161, January 2012.
7. Emma Haddi, Xiaohui Liu, Yong Shi, The Role of Text Pre-processing in Sentiment Analysis: International Conference on Information Technology and Quantitative Management, pp. 234-231, 2013.
8. Suranga Hettiariachchi, Wesley Kerr: Boyer-Moore String Matching algorithm, Technical paper downloaded from http://cs.eou.edu/CSMM/surangah/research/boyer/boy.pdf
9. Anany Levitin: Introduction to The Design and Analysis of Algorithms, 2nd edition, ISBN: 9780321358288, published by Pearson Education, Inc., 2007.
10. http://www.inf.fh-flensburg.de/lang/algorithmen/pattern/indexen.htm
11. http://www.boost.org/doc/libs/1_54_0/libs/algorithm/doc/html/the_boost_algorithm_library/Searching/BoyerMooreHorspool.html
12. http://www.personal.kent.edu/~rmuhamma/Algorithms/MyAlgorithms/StringMatch/boyerMoore.htm
13. http://cs.indstate.edu/~kmandumula/presentation.pdf
14. http://www.cs.utexas.edu/~moore/best-ideas/string-searching/index.html