Advances in Computer Science and Information Technology (ACSIT)
p-ISSN: 2393-9907; e-ISSN: 2393-9915; Volume 4, Issue 3; April-June, 2017, pp. 155-160

http://www.krishisanskriti.org/Publication.html

ASCII Based Algorithm for High Level Cloning

Manu Singh[1] and Vidushi Sharma[2]

[1,2]School of ICT Gautam Buddha University, Greater Noida, Uttar Pradesh, India
E-mail: [1]manu_akumar@yahoo.co.in, [2]vidushi@gbu.ac.in

Abstract—*The current research in clone detection is now focusing on developing better algorithm for high level clones. Many algorithms have been proposed but more efficient and robust methods are needed. Pattern matching is one of the promising areas which show potential in research of computer science. The Structural clones of high level clones comprised lower level smaller clones with similar code fragments. In this repetitive occurrence of simple clones in a file may prompt higher file level clones. The proposed algorithm detects clones at higher level of abstraction like file and also detects repetitive patterns in same file. To identify DNA sequences there are various pattern matching algorithms used in genetic area. In the proposed work an ASCII based sequential multiple pattern matching algorithm gives better performance when compared with some of the existing algorithms. The present method avoids superfluous DNA comparisons as a consequence the number of comparisons and character par comparison proportion gradually decline and overall performance increments.*

Keywords*: Pattern matching, ASCII based, high level clone, file clone.*

1. INTRODUCTION

A software system is constantly changing, and consistent maintenance is required to help it adapt to the new changes. Designs, software upgrades, compilers, hardware upgrades and so forth all influence the working of software. Because of standard adjustments in code, redundancies happen in code and programming will be more mind boggling and troublesome in keeping up. Now and then this excess is known as cloning. Cloning may occurs at various abstraction levels and have unusual source [1]. Literature study portrays half cloning in the source codes [2]. In literature several techniques used to identify simple clone fragments [3] but detection clones at higher levels remains a promising area till now. One of the promising area in clone detection is pattern matching. pattern matching is the act of checking the occurrences of a particular pattern of characters in a large file. This paper investigate the applicability of a new technique of pattern matching approach called ASCII based Pattern Matching algorithm, for detection of high level clone in source code. High Level Clones are classified [4] in structural clone, concept clone, behavioural clone [5] and domain model clone. This classification depicts that structural clones are formed by similar fragments of code at low level. This approach avoids lengthy comparisons in string sequence and reduces the effort for each character comparison at each attempt. The proposed algorithm gives better results as compared to other algorithms.

The rest of the paper is organized as follows. Proposed algorithm is explained in section II. Experimental results are presented in section III. Concluding remarks are given in section IV.

2. RELATED WORK

There are various string matching techniques which mainly deal with problem of identifying occurrences of a substring in a given string or locate the occurrences of specific pattern in a sequence. In this section we explore these different types of string matching techniques. Some techniques are based on algorithms of exact matching in string, such as Naïve string search, Brute-force algorithm, Bayer-Moore algorithm, Knuth-Morris-Pratt algorithms [6],[7] and some are based on approximate string matching algorithms, dynamic programming is mostly used approach. In IBKPMPM [8] choose the value of k and divide both the string and pattern into number of substring of length k, each substring is called as a partition. We compare all the first characters of all the partitions, if all the characters are matching while we are searching then we go for the second character match and the process continues till the mismatch occurs or total pattern is matched with the sequence. In Index based forward backward multiple pattern matching technique [9] the elements in the given patterns are matched one by one in the forward and backward until a mismatch occurs or a complete pattern matches. In the MSMPMA [10] technique the algorithm scans the input file to find the all occurrences of the pattern based upon the skip technique. Index is used as the starting point of matching; it compares the file contents from the defined point with the pattern contents, and finds the skip value depending upon the match numbers (ranges 1 to m-1). In IBSPC [11] indexes has been used for the DNA sequence. After creating the index the algorithm will search for the pattern in the string using the index of least occurring character in the string. In Index Based Algorithm [12], on the basis of frequently occur character index table is created and then align pattern with string and matched occurrence of patterns with multiple times one by one from left to right in the file. This paper proposed the most efficient approach for finding similarity between

multiple pattern, till date. To further increase the performance of pattern matching an ASCII based multiple pattern matching algorithm using ASCII value comparison between pattern and substring is proposed. It is a simple approach for finding multiple occurrences of patterns from a given file. This algorithm gives better results when compare it with existing algorithms. This approach provides best results with the DNA sequence dataset. Proposed algorithm is implemented and compared with existing algorithms. Experimental results of applying the technique to DNA sequences show the effectiveness of the proposed technique.

3. PROPOSED ALGORITHM

3.1 ASCII Based Multiple Pattern Matching Algorithm –

Input: String S of n characters and Length of pattern P of m characters

Output: The number of occurrences of Pattern in String, its location and the number of characters compared.

Step 1: [Initialization of variables & Index array]

Integer i:=0 and ctr:=0, subStr[m], start:=0

Step 2: [Extract Pattern P from String S]

For i=0 to pattenLength

P[i]=S[i]

If (stringLength<patternLength)

Exit

Endif

Step 3: [Calculate Ascii value of Pattern]

Step 4: LOOP till End of File

Step 5: End = start + patternLength – 1;

Step 6: Extract substring subStr from String S

For i=start to End

subStr[i]=S[i]

Step 7: [Calculate Ascii value of substring]

Step 8: [compare the ascii sum of string S and ascii

sum of pattern P]

If substr_asc == patternasc

Step 9: [If they are equal compare individual characters of string and pattern.]

For i=start to End

num=num+1

If (substr(i) == P(i))

found=1;

continue;

else

break;

endif

End

Step 10: if found==1

[Increment ctr with 1]

print (found at location, start)

endif

Step 11: [Increment start with 1]

End

Step 12: print (Total Pattern Found, ctr)

Step 13: END

3.2 Working Example –

Assume there is a string to understand the proposed algorithm

S= R G B G R W B B B G R W B R R G B G R W W B G R W B G B W of 29 characters

Divide the string into 5 characters in each substring. And calculate the ascii value of all substrings. Suppose the given pattern is P.

P = R G B G R of 5 characters

Compare the ascii value of both the pattern and substring, if match then align the pattern and substring and compare character by character

S= R G B G R W B B B G R W B R R G B G R W W B G R W B G B W of 29 characters

P =R G B G R

The first character matches then it compares the second character of the pattern.

S= R G B G R W B B B G R W B R R G B G R W W B G R W B G B W of 29 characters

P =R G B G R

The second character also matches then it compares the third character.

S= R G B G R W B B B G R W B R R G B G R W W B G R W B G B W of 29 characters

P =R G B G R

The third character also matches then it compares the fourth character.

S= R G B G R W B B B G R W B R R G B G R W W B G R W B G B W of 29 characters

P =R G B G R

The fouth character also matches then it compares the fifth character.

S= R G B G R W B B B G R W B R R G B G R W W B G R W B G B W of 29 characters

P =R G B G R

All the characters are matched, so the pattern is found at position 1.

Next the algorithm aligns the pattern with next character in string.

S= R G B G R W B B B G R W B R R G B G R W W B G R W B G B W

P= R G B G R

Continue till ascii sum of pattern and ascii sum of substring does not match.

S= R G B G R W B B B G R W B R R G B G R W W B G R W B G B W

P= R G B G R

Ascii value of both the pattern and substring match so start comparing the pattern and substring character by character

S= R G B G R W B B B G R W B R R G B G R W W B G R W B G B W

P= R G B G R

The first character matches then it compares the second character of the pattern.

S= R G B G R W B B B G R W B R R G B G R W W B G R W B G B W

P= R G B G R

The second character not matches with the second character of the pattern. Now continue

S= R G B G R W B B B G R W B R R G B G R W W B G R W B G B W

P= R G B G R

Ascii value of both the pattern and substring match so start comparing the pattern and substring character by character

S= R G B G R W B B B G R W B R R G B G R W W B G R W B G B W

P= R G B G R

The second character not matches with the second character of the pattern. Now continue

S= R G B G R W B B B G R W B R R G B G R W W B G R W B G B W

P= R G B G R

Ascii value of both the pattern and substring match so start comparing the pattern and substring character by character

S= R G B G R W B B B G R W B R R G B G R W W B G R W B G B W

P= R G B G R

S= R G B G R W B B B G R W B R R G B G R W W B G R W B G B W

P= R G B G R

S= R G B G R W B B B G R W B R R G B G R W W B G R W B G B W

P= R G B G R

All the characters are matched, so the pattern is found at position 15. Next the algorithm aligns the pattern with next character in string.

S= R G B G R W B B B G R W B R R G B G R W W B G R W B G B W

P= R G B G R

Ascii value of both the pattern and substring match so start comparing the pattern and substring character by character

S= R G B G R W B B B G R W B R R G B G R W W B G R W B G B W

P= R G B G R

The second character not matches with the second character of the pattern. Now continue

S= R G B G R W B B B G R W B R R G B G R W W B G R W B G B W

P= R G B G R

Ascii value of both the pattern and substring match so start comparing the pattern and substring character by character

S= R G B G R W B B B G R W B R R G B G R W W B G R W B G B W

P= R G B G R

The second character not matches with the second character of the pattern. Now continue

S= R G B G R W B B B G R W B R R G B G R W W B G R W B G B W

P= R G B
G R

The ASCII values doesn't match and the pattern is aligned with the next substring of the string

Now the pattern is moved to the next index and the comparison does not occur and algorithm terminates because the length of substring is less than the pattern length.

Now at the end of the algorithm the total 2 Patterns occurred in the String. By above example we can conclude that comparing ascii values reduces the number of comparisons.

4. EXPERIMENT AND RESULT ANALYSIS

The DNA Sequence data has been taken from the Multiple Skip Multiple Pattern Matching algorithm MSMPMA[10] for testing the proposed algorithm. After implementation of the proposed ASCII based multiple pattern matching algorithm for the 1024 characters and finding the no of occurrences and no. of comparisons, It has been concluded that the number of comparisons reduces as the pattern size of DNA increases and are shown below in the Table 1.

Table 1: Experimental Results of Proposed Algorithm

Pattern	No. of Occurrence	No. of Comparison
A	259	516
AG	53	278
CAT	10	131
GACA	6	127

By using the proposed algorithm different patterns are analyzed and the graph is plotted by using these results as shown in Fig. 1.

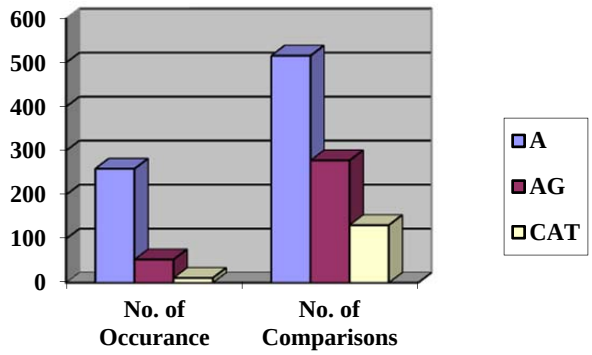

Fig. 1: Analysis based on increased pattern size

4.1 Comparison of different algorithms using DNA Sequence

As we have collected the data for various existing algorithm and drawn the comparative analysis with respect to the various existing algorithm. The results of proposed ASCII based algorithm for DNA Sequence with different pattern size were compared with Brute Force, MSMPMA[10], Bayer Moore[14], IFBMPM[9], and index based Algorithms[12] are shown in the Table 2.

Table 2: Comparison of different algorithms using Different pattern size

PATTERN	Brute Force	MSMPMA	Bayer Moore	IFBMPM	Index Based	ASCII Based
A	1024	1024	1024	518	774	516
AG	1284	1230	734	624	414	278
CAT	1318	1298	607	567	201	131
GACA	1376	1359	504	614	272	127

When compared the proposed algorithm with some of the other algorithms the following experiments has been observed. Fig.2 shows the number of comparisons made for different algorithms to the single pattern of length 1. For a single pattern "A" the proposed algorithm takes 516 comparisons whereas all the other algorithms like Brute force, MSMPMA, Bayer Moore[14] take 1024 comparisons, index based taken 774 comparisons and IFBMPM takes 518 comparisons.

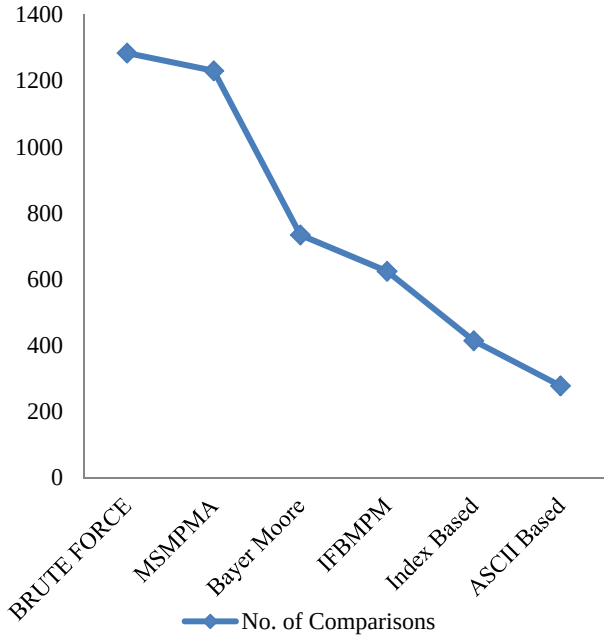

Fig. 2: Experimental results of different algorithms for a single character pattern size

Fig. 3 shows the number of comparisons made for different algorithms to the pattern of length 2. The pattern "AG", in the proposed algorithm takes 278 comparisons where as all the other algorithms like Brute-force, MSMPMA, Bayer Moore, IFBMPM and Index based algorithms takes 1284, 1230,734, 624 and 414 comparisons respectively. We are reducing the comparisons by using the ASCII based technique.

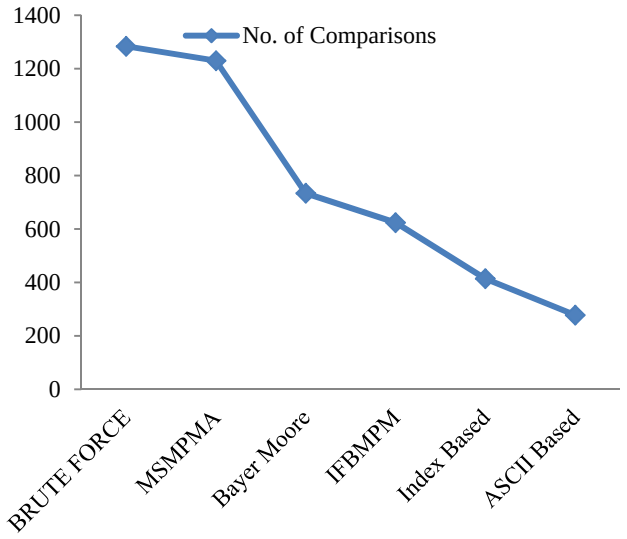

Fig. 3: Experimental results of different algorithms for a pattern size "2"

Fig. 4 shows the number of comparisons made for different algorithms to the pattern of length 3. The pattern "CAT", in the proposed algorithm takes 131 comparisons where as all the other algorithms like Brute-force, MSMPMA, Bayer Moore, IFBMPM and Index based algorithms takes 1318, 1298, 607, 567 and 201 comparisons respectively.

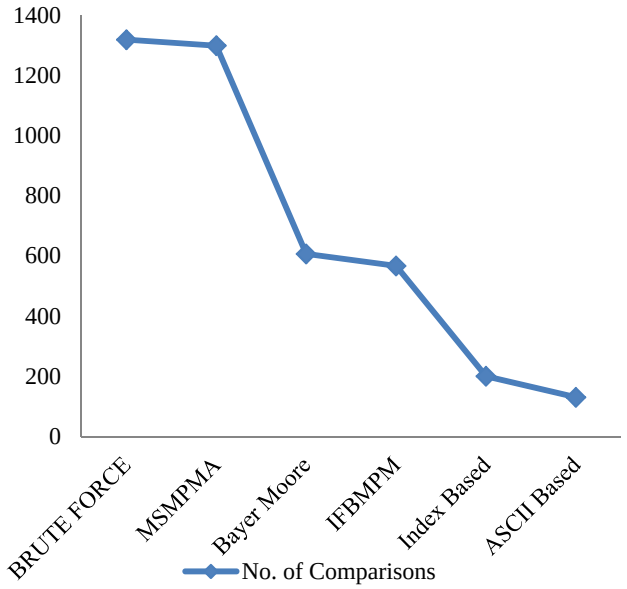

Fig. 4: Experimental results of different algorithms for a pattern size "3"

Among them proposed ASCII based algorithm gives very good performance in reducing the number of character comparisons compared with other popular methods and existing algorithms. Towards X-axis we have taken algorithm names and towards Y-axis show the total number of comparisons. It can be concluded through experiments that proposed algorithm:

- Decreases number of comparisons as pattern size increases.
- Take less time as lesser number of comparisons.
- Take less memory space as less memory storage required for comparison.
- Appropriate for large size input file.

5. CONCLUSION

We suggested a new algorithm which can be use for pattern matching in DNA sequences. This approach is suitable for high level cloning detection as it works with large size of input sequence. Dependent upon the experimental work our methodology gives beneficial execution related to DNA sequence dataset. Our suggested algorithm minimizes the total number of comparison when compared with the some of the best known popular algorithm.

REFERENCES

[1] Basit, H. A., Jarzabek, S., "A Case for Structural Clones", International Workshop on Software Clones, 2009.
[2] Baker, B. S., "On Finding Duplication and Near duplication in Large Software System", Proceedings of 2nd IEEE Conference of Reverse Engineering, 1995.
[3] William S. Evans, Christopher W. Fraser and Fei Ma, "Clone Detection via Structural Abstraction", Software quality journal Vol. 17, No. 4, 2009.
[4] Singh, M., Sharma, V., "High Level Clones Classification" International Journal of Engineering and Advanced Technology (IJEAT) ISSN : 2249 – 8958, Vol. 2, Issue - 6, August 2013.
[5] Singh, M., Sharma, V., "Detection of Behavioral Clone International Journal of Computer Applications (0975 – 8887) Vol. 102 – No.14, 2014.
[6] Bayer R. S., Moore, J. S., "A Fast String Searching Algorithm", Communications of the ACM, pp. 762-772 ,1977.
[7] Knuth D., Morris.J, Pratt.V.R., "Fast Pattern Matching in Strings", SIAM Journal on Computing Vol. 6 (1), 1977.
[8] Bhukya, R., Somayajulu, D.,"An Index Based KPartition Multiple Pattern Matching Algorithm", Proc. Of International Conference on Advances in Computer Science 2010 pp 83-87.
[9] Bhukya, R., Somayajulu, D.,"An Index Based Forward Backward Multiple Pattern Matching Algorithm", World Academy of Science and Technology. June 2010, pp347-355

[10] Ziad A.A Alqadi, Musbah Aqel & Ibrahiem M.M.EI Emary, Multiple Skip Multiple Pattern Matching algorithms. IAENG International Journal of Computer Science 34:2.

[11] Bhukya, R., Somayajulu, D." Index Multiple Pattern Matching Algorithm using DNA Sequence and Pattern Count", International Journal of Information Technology and Knowledge Management July-December 2011, Volume 4, No. 2, pp. 431-441

[12] Singh, M., Sharma, V., "Index based detection of file level clone for high level cloning", International Journal of Computer Science Engineering and Information Technology Research (IJCSEITR) ISSN(P): 2249-6831; ISSN(E): 2249-7943 Vol. 5, Issue 4, Aug 2015, 63-70