ASCII based Sequential Multiple Pattern Matching Algorithm for High Level Cloning

Manu Singh
School of ICT,
Gautam Buddha University
Greater Noida, Uttar Pradesh, India

Vidushi Sharma
School of ICT,
Gautam Buddha University
Greater Noida, Uttar Pradesh, India

***Abstract*—For high level of clones, the ongoing (present) research scenario for detecting clones is focusing on developing better algorithm. For this purpose, many algorithms have been proposed but still we require the methods that are more efficient and robust. Pattern matching is one of those favorable algorithms which is having that required 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 repetitive patterns in same file and clones at higher level of abstraction like file. In genetic area, there are a number of algorithms that are being used to identify DNA sequence. When compared with some of the existing algorithms the proposed algorithm for ASCII based sequential multiple pattern matching gives better performance. The present method increases overall performance and gradually decline the number of comparisons and character per comparison proportion by repudiating (avoid) unnecessary DNA comparisons.**

Keywords—Pattern matching; ASCII based; high level clone; file clone

I. 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 occur 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 investigates 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: Related work is explained in Section II. Proposed algorithm is explained in Section III. Then simulation results are presented in Section IV. Experimental results of proposed algorithm are discussed in Section V. Section VI explain graphically the effect of increasing pattern size on performance indices. Section VII discusses comparative analysis of proposed algorithm with another algorithm. Section VIII analyse the impact of cumulative pattern size increment on no. of comparison. Then final performance analysis of proposed algorithm is given in Section IX. Concluding remarks are given in Section X.

II. 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 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 An indexed based K-Partition Multiple Pattern Matching Algorithm (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 algorithm (IFBMPM) [9] patterns matching technique the characters in the given patterns are matched one by one in the forward and backward until a mismatch occurs or a whole pattern matches. In the Multiple Skip Multiple Pattern Matching Algorithm (MSMPMA) [10] technique the algorithm search the input text to find the all occurrences of the pattern based upon the skip technique. To get starting location of the matching Index is used; it compares the Text characters from the well-defined point with the pattern characters, and based on the match numbers decides the skip value (ranges 1 to m-1). In IBSPC [11] indexes have been used for the DNA sequence. Least occurring character index will be used to search for the pattern 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 in VB.NET and results are compared with already existing algorithms. Experimental results of applying the technique to DNA sequences show the effectiveness of the proposed technique.

III. PROPOSED ALGORITHM

The proposed approach has been used ASCII value of characters for comparison. The algorithm considered a DNA sequence string S of 1024 characters as input. First of all, extract substring from string S equal to the pattern length m. Calculate the ASCII sum of all substrings. Suppose the given pattern is P. Compare the ASCII sum of both the pattern and substring, If ASCII sum of both the pattern and substring match so start comparing the pattern and substring character by character. If characters are not matched then skip the rest comparison of characters of substring and aligned the pattern with the next substring of the string. This process Continue till substring is less than the pattern length. By above example we can conclude that comparing ASCII values reduces the number of comparisons as when ASCII sum is not match then there is no need to compare substring and pattern character by character.

A. 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.

```
Dim QueryASCIITable As String, patternASCIIValue As Int, _noOfComparison = 0
Step1: Dim count As Integer = 0, qStringarr As String (), patternarr As String (), tempstr As String
Dim queryIndex As Int32 = 0, blFound As Boolean = True
        If String.IsNullOrEmpty (String_S) Then
                qStringarr = String_SArray
'Array of substring
        If String.IsNullOrEmpty (Pattern_P) Then
                patternarr = m_Pattern_PArray
'Array of Pattern string
Step 2: ['Store the ASCII value of each size]
         For a As Integer = 0 To qStringarr.Length - 1
        If qStringarr.Length - a >= patternarr.Length Then
tempstr()=(ArraySelect(qStringarr,a,a+patternarr.Length)).ToArray()
QueryASCIITable.Add (a, GetASCIISum (tempstr), tempstr))
        End If
        Next
Step 3: [Store the ASCII value of pattern]
        patternASCIIValue = GetASCIISum(patternarr)
Step 4 : While (queryIndex < QueryASCIITable.Count - 1)
         If patternASCIIValue=QueryASCIITable(queryIndex).Key Then
 _noOfComparison += 1
 For patternIndex As Integer = 0 To patternarr.Length - 1
 _noOfComparison += 1
 If patternarr (patternIndex) = QueryASCIITable (queryIndex).Value (patternIndex) Then
 Continue For
 Else
 blFound = False
 Exit For
 End If
 Next
 If blFound Then
 indexarrfound.Add (queryIndex)
        End If
        End If
 queryIndex += 1
 End While
```

B. Performance Indices

Pattern matching algorithm efficiency can be judged by using certain performances indices. To make the comparisons we have used following performance indices:

1) No. of Occurrences: If we are given an array of text T (1……n) of length n and the pattern is an array P (1…m) of length m such that m<=n then the number of occurrences of pattern will be (n-m+1).

2) No. of Comparisons: Objective of pattern matching algorithm is to reduce the number of character comparison in worst and average case analysis.

3) Best Case: The best case of this algorithm will be, when the pattern matches in the first shift. Therefore, best case is, $T(n)=\Omega(m)$.

4) Worst Case: Let in worst case situation the pattern matches at every shift in the text then there will be 'm' comparisons in each shift, so the total number of comparisons will be 'm(n-m+1)'. So the worst time complexity of this algorithm will be , $T(n)=O(m(n-m+1))$.

5) Comparisons per Character (CPC): CPC is used as a measurement factor. Complexity is decreased when CPC decreased. CPC ratio can be calculated as CPC = (Number of comparisons/file size).

IV. SIMULATION RESULTS

The algorithm was implemented using VB.NET, and it was tested using different DNA sequence with different file sizes. However, the proposed algorithm is compared with other algorithms. They are MSMPMA, Brute-Force, Trie-matching and Index Based algorithm. These algorithms are selected due to common features with the proposed algorithm as follows:

1) Multiple string matching

2) No pre-processing operations: As the proposed algorithm compares the ASCII value of the character rather than the character itself. Thus it will not take any pre-processing time and hence no pre-processing operation is required before comparison due to which this algorithm will become more efficient as compared to other algorithms.

3) Maintaining different type of files: The implementation and comparison with other algorithms process is carried out When text file size = 1024 bytes, using different patterns and sizes in implementation process. The results are obtained and can be grouped in various sections.

V. Performance Analysis of ASCII based algorithm

The DNA Sequence data has been taken from the Multiple Skip Multiple Pattern Matching algorithm MSMPMA [9] for testing the proposed algorithm. After implementation of the proposed ASCII based multiple pattern matching algorithm for the 1024 character and finding the no of occurrences, no of comparisons and CPC ratio it has been concluded that the number of comparisons reduces as the pattern size of DNA increases and are shown below in Table 1.

TABLE I. Experimental Results of Proposed Algorithm

S. N	Pattern	No. of Character	No. of Occurrence	No. of Comparison	CPC Ratio
1	A	1	259	516	0.5
2	AG	2	53	278	0.27
3	CAT	3	11	131	0.128
4	GACA	4	6	127	0.124
5	AACGC	5	2	2	0.001

VI. Effect of Increasing Pattern Size on Different Performance Indices

After the implementation of the proposed algorithm, the following points could be concluded from the obtained results in table below.

The result indicates that to find pattern with one char length from 1024 DNA data sequence proposed algorithm required 516 no. of comparisons (almost half). It means proposed algorithm requires 0.50 comparisons/character to search one-character pattern.

Further table indicates that when pattern increased in size, the no. of comparisons to find the pattern is decreased. Therefore, it can be said that this is very beneficial for detection of high level clones because high level cloning is found at coarser level not at the fine level.

When the pattern length increased, Comparison per Character decreased, and it is a well-known fact that Complexity time is affected by Comparison per Character. It depicts that the complexity is also decreased when pattern size increased.

If we take in consideration the number of pattern occurrences, we can say that the complicity is less than O(n), since We need less number of comparisons for the second match and less for the third and so on.

The number of comparisons which affects the processing time rapidly decreased after the first match, and the total number of comparisons for all occurrences will be less than the text file size.

A. Analysis on the Basis of Occurrence

Fig. 1 depicts that number of occurrences decreased when pattern size increased. Generally when pattern size increase there is less probability to find pattern in file and at that time algorithm that can search large pattern in less number of comparisons is required.

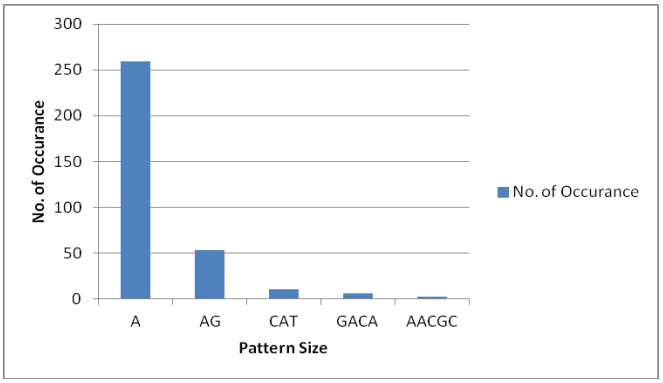

Fig. 1. Relation between pattern size and number of occurrences

B. Analysis on the Basis of Comparisons

The impact of increasing pattern size on number of comparisons have been displayed in Fig. 2. As the graph shows that when small pattern size is searched in file, number of comparisons is at its highest level but as the pattern size increased number comparisons decreased gradually.

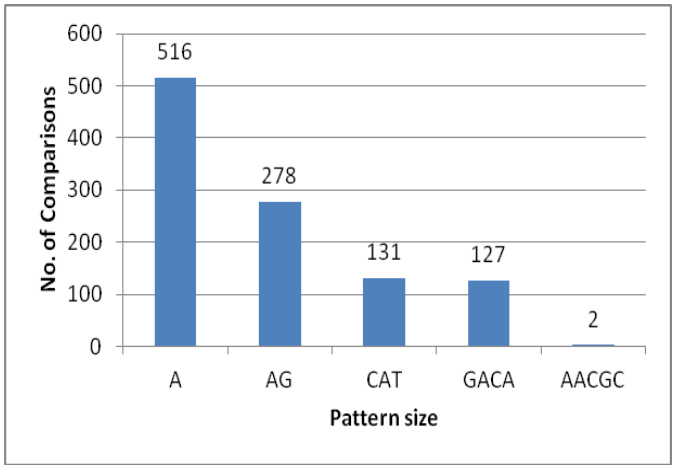

Fig. 2. Relation between pattern size and number of comparisons

C. Analysis on the Basis of Comparisons Per Character

The impact of increasing pattern size on comparison per character can be noticed in Fig. 3. When the pattern length increased, Comparison per Character decreased. Complexity time is affected by Comparison per Character. It depicts that the complexity is also decreased when pattern size increased.

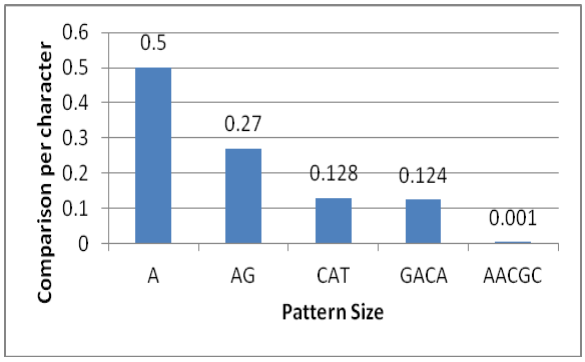

Fig. 3. Relation between pattern size and comparison Per character

VII. Comparative Analysis of ASCII based Algorithm with Other Existing Algorithms

As we have collected the data for various existing algorithm [10], [12] and drawn the comparative analysis in Tables 2.1, 2.2 and 2.3 with respect to the various existing algorithm.

TABLE II. (1) PATTERN=A (M=1)

Name of Algorithm	No. of Occurrences	No. of Comparisons	Comparisons per Character
MSMPMA	259	1024	1
Brute-Force	259	1024	1
Naïve String Search	259	1024	1
Trie-matching	259	1025	1.001
Index Based	259	774	0.75
ASCII Based	259	516	0.503

TABLE II. (2) PATTERN=AG (M=2)

Name of Algorithm	No. of Occurrences	No. of Comparisons	Comparisons per Character
MSMPMA	53	1230	1.201
Brute-Force	53	1282	1.252
Naïve String Search	53	1281	1.250
Trie-matching	53	1284	1.254
Index Based	53	414	0.404
ASCII Based	53	278	0.271

TABLE II. (3) PATTERN=CAT (M=3)

Name of Algorithm	No. of Occurrences	No. of Comparisons	Comparisons per Character
MSMPMA	11	1298	1.268
Brute-Force	11	1318	1.287
Naïve String Search	11	1321	1.290
Trie-matching	11	1310	1.279
Index Based	11	224	0.218
ASCII Based	11	131	0.128

Among them our algorithm which gives very good performance. It can be analysed that ASCII based algorithm gives improvements to other algorithms are following:

1) Decreases number of comparisons in average and best case analysis.
2) Appropriate for very large size input file.

VIII. Analysing the Impact of Cummulative Pattern Size Increment on Number of Comparison

Table 3 given below compare the total number of comparisons of different algorithms [13] with randomly selected different pattern sizes ranges from 1 to 8 in cumulative manner. As the size of pattern increasing in cumulative manner, the number of comparisons in proposed algorithm are lesser as compared to other pattern matching algorithms.

TABLE III. COMPARISON OF DIFFERENT ALGORITHMS USING DNA SEQUENCE FOR CUMULATIVE PATTERN [13]

	No. of Comparison							
Pattern	**Brute Force**	**MSM PMA**	**IFBM PM**	**IBMP M**	**Pair count**	**Boyer Moore**	**Index Based**	**ASCI I Base d**
A	**1024**	**1024**	**518**	**259**	**259**	**1024**	**774**	**516**
A+AG	**2308**	**2254**	**1142**	**777**	**506**	**1758**	**1188**	**794**
A+AG+CAT	**3626**	**3552**	**1709**	**1319**	**802**	**2365**	**1389**	**925**
A+AG+CAT+GACA	**5002**	**4911**	**2323**	**1933**	**1060**	**2869**	**1661**	**1052**
A+AG+CAT+GACA+AACGC	**6390**	**6286**	**2939**	**2540**	**1332**	**3235**	**1946**	**1054**
A+AG+CAT+GACA+AACGC+GACAAG	**7799**	**7680**	**3573**	**3163**	**1613**	**3611**	**2229**	**1058**
A+AG+CAT+GACA+AACGC+GACAAG+TCGGGTG	**9189**	**9070**	**4224**	**3797**	**1890**	**3811**	**2501**	**1060**
A+AG+CAT+GACA+AACGC+GACAAG+TCGGGTG+CCAAAAAA	**10538**	**10419**	**4822**	**4377**	**2163**	**4168**	**2759**	**1094**

The current technique gives good performance in reducing the number of character comparisons compared with other popular methods and existing algorithms. The results of proposed ASCII based multiple pattern matching algorithm and other existing algorithms for pattern size three also plotted in the graph as shown in Fig. 4.

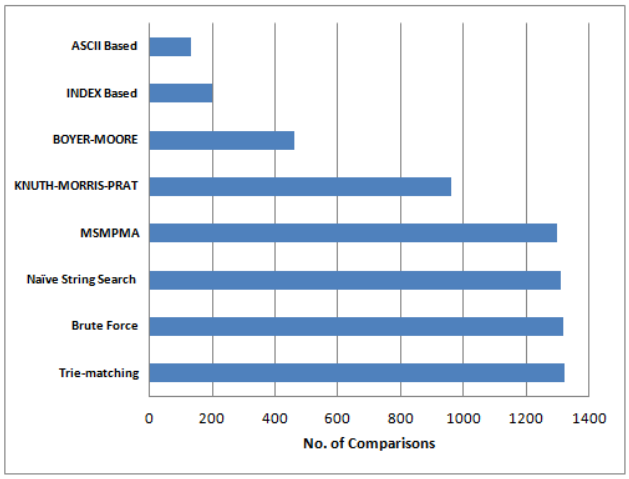

Fig. 4. Comparison of different algorithms for pattern [CAT]

This shows the reduction in number of comparison when pattern size is three-character long. Towards X-axis we have taken total number of comparisons whereas towards Y-axis it shows the names of all the algorithms.

IX. Performance Analysis of the Proposed Algorithm

Performance analysis of an algorithm is performed and explained by using the following measures.

1) Number of Occurrences: The number of occurrences of pattern will be (n-m+1).As the proposed algorithm is based on the ASCII values of the characters, it does not required any pre-processing time and consumes less space in memory as compared to Brute-Force Algorithm because Brute-Force algorithm requires extra CPU registers to hold the intermediate value, but the proposed algorithm do not require any extra register because it directly compares the ASCII value for any comparison thus we can say that it is more efficient as compared to Brute-Force and all the other algorithms.

2) Number of Comparisons: The number of character comparison in worst and best case analysis are shown in Table 4 and discusses as follows:

a) *Best Case*: The best case of this algorithm will be, when the pattern matches in the first shift. Therefore, best case is, T(n)=O(m).

b) *Average Case*: The average case of this algorithm will be, T (n) =Ω (m).

c) Worst Case: Let in worst case situation the pattern matches at every shift in the text then there will be 'm' comparisons in each shift, so the total number of comparisons will be 'm(n-m+1)'. So the worst time complexity of this algorithm will be, T(n)=O(m(n-m+1)).

TABLE IV. Comparison of Different Algorithms [14]

Algorithm	Pre-processing Time Required	Running Time	Best Case	Worst Case
Brute-Force Algorithm	NO	O(n-m+1) m	O(m)	O(n-m+1) m)
Knuth-Morris Prat	YES	O(n+m)	O(n)	Θ (n.m)
Boyer-Moore	YES	O (n\m)	O(m)	O(n-m+1) m+ Σ
ASCII Based	NO	O (n-m+1)	O(m)	O(m(n-m+1)

Degenerating property, i.e., of pattern is used by the proposed algorithm (in the same pattern sub-patterns appearing more than one time) and improves the worst-case complexity. The fundamental thought behind proposed algorithm is: at whatever point we identify a mismatch (after some matches), we definitely know a portion of the characters in the text of next window. We take advantage about this majority of the data to evade matching those characters that we know will in any case match.

X. Conclusion

We proposed a new algorithm which can be used for pattern matching in DNA sequences. This approach is suitable for unlimited size of input sequence. It reduces the total number of comparison as well as the CPC ratio when compared with other popular algorithms. The proposed algorithm gives very good performance with the other algorithms. Based on the experimental work our approach provides good performance related to DNA sequence dataset. Our proposed algorithm reduces the total number of comparison as well as the CPC ratio when compared with the some of the best known popular algorithm. In future, the proposed algorithm detects repetitive patterns at higher level of abstraction like file.

References

[1] H. A. Basit, S. Jarzabek, " A Case for Structural Clones ", International Workshop on Software Clones , 2009.

[2] B. S Baker, "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] M. Singh, V. Sharma, "High Level Clones Classification" International Journal of Engineering and Advanced Technology (IJEAT) ISSN : 2249 – 8958, Vol. 2, Issue - 6, August 2013.

[5] M. Singh, V. Sharma, "Detection of Behavioral Clone International Journal of Computer Applications (0975 – 8887) Vol. 102 – No.14 , 2014.

[6] Bayer R. S., J. S. Moore, "A Fast String Searching Algorithm " , Communications of the ACM , pp. 762-772 ,1977 7.

[7] Knuth D., Morris.J ,Pratt.V.R., "Fast Pattern Matching in Strings ", SIAM Journal on Computing Vol. 6 (1), 1977.

[8] Raju Bhukya, DVLN Somayajulu,"An Index Based KPartition Multiple Pattern Matching Algorithm", Proc. Of International Conference on Advances in Computer Science 2010 pp 83-87.

[9] Raju Bhukya, DVLN Somayajulu,"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

[11] Raju Bhukya, DVLN Somayajulu," 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] M. Singh, V. Sharma, "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

[13] Raju Bhukya, DVLN Somayajulu, "Multiple Pattern Matching Algorithm using Pair-count", IJCSI International Journal of Computer Science Issues, July 2011, Vol. 8, Issue 4, No 2, pp. 453-465.

[14] Diwate. R.B., , Alaspurkar.S. J., " Study of Different Algorithms for Pattern Mining", International Journal of Advanced Research in Computer Science and Software Engineering 3(3), March - 2013, pp. 615-620