International Journal of
Emerging Research in Management &Technology
ISSN: 2278-9359 (Volume-4, Issue-5)

Research Article | **May 2015**

Efficient Parameterized String Matching Algorithm

Richa Sharma*
M.tech (CSE)
VITS Ghaziabad
Uttar Pradesh, India

Vibhor Gupta
Department of CSE
VITS Ghaziabad
Uttar Pradesh, India

Vikas Kumar
Department of CSE
ISM Dhanbad
Jharkhand, India

***Abstract*—**

String matching problem has attracted a lot of interest throughout the history of computer science and is necessary for the computing industry. It refers to finding all occurrences of a pattern P[1....m] within the text T[1....n], where m≤n and character of both the strings (P and T) are drawn form same alphabet ∑. In case of multiple string matching, the problem is to find all occurrences of the pattern set {P0, P1, P2.....Pk}, k≥0 in the text T[1....n], simultaneously. Parameterized string matching problem refers to the problem of finding occurrences of a pattern string P within the substrings t of the text T, where symbols of P and T are consistently renamed with some bijective mapping. In this paper, we propose an algorithm for multi-pattern parameterized matching.

Keywords*— *String matching, exact string matching, parameterized string matching, predecessor string, bad character, good suffix

I. INTRODUCTION

String matching is a very important area in the domain of text processing and algorithms. String matching algorithms basically tries to find a place where a given string is found within a large body of text. It can be both exact and parameterized. The exact string matching problem is the most fundamental string matching problem. In this problem, the task is to find all exact occurrences of the pattern in the text. In the parameterized string matching problem, the pattern matches a substring of the text if the characters of the text substring can be renamed in such a way that the pattern matches the renamed substring exactly. Multiple string matching problems aims to find all simultaneous occurrences of each patterns in a pattern set $\{P_0, P_1, P_2,.....,P_{r-1}\}$, $r \geq 1$, in the body of text T. By searching multiple patterns in one go reduces the overall searching time. Multiple string matching technique have various applications in different areas [1] like, bibliographic search, bioinformatics, data filtering, security applications, DNA searching, and information retrieval etc. There are many existing, multiple string matching algorithms namely, Aho-Corasick [2], based on finite automata, Wu-Manber [3] algorithm based on Boyer-Moore algorithm and Quick Multiple Matching algorithm [4] which is an extension of Wu-Manber algorithm. It uses the concept of quick search algorithm.

In the parameterized string matching problem, patterns and text consists of two symbols: fixed symbol from alphabet Σ and parameter symbol from alphabet Π. During matching, fixed symbols are used as it is, while parameterized symbols can be renamed via some bijective mapping. Two strings of equal-length are said to be parameterized match if one string can be transformed into the other by applying a one-to-one correspondence that renames the parameter symbols. Baker in 2009 introduced the concept of predecessor string to simplify this bijective mapping. Any two parameterized strings are parameterized match with each other if their predecessor strings matches exactly. Predecessor string for a given string is calculated as, if characters at position i have already occurred at position j then predecessor string contain i-j at position i. Otherwise predecessor string contain 0. In this paper we will focus on parameterized matching.

Parameterized string matching problem was first considered by Baker [5] in 1993 in his paper a theory of parameterized pattern matching with an application to software maintenance.

In this paper, we extend an existing exact string matching algorithm for parameterized string matching.

Further this paper is divided into several sections. Section II presents the related concepts and algorithms for exact and parameterized string matching. Section III gives the proposed algorithm for multiple parameterized string matching. Section IV presents the implementation results and finally section V concludes the paper.

II. RELATED ALGORITHMS

A. Parameterized String Matching Problem

Your Parameterized matching problem refers to finding p-match occurrences of all substring of text with the symbols of the pattern. In the parameterized string matching problem, patterns and text consists of two symbols: fixed symbol from alphabet Σ and parameter symbol from alphabet Π. During matching, fixed symbols are used as it is, while parameterized symbols can be renamed via some bijective mapping. For example, consider the fixed alphabet symbol Σ = {a, b} and parameterized alphabet symbol Π = {x, y}. Parameterized string (p-string) over these alphabets is: axybyx.

Two strings of equal-length s and s′ are said to be parameterized match (p-match), if one string can be transformed into the other by applying a one-to-one correspondence that renames the parameter symbols. For example: Let Π = {x, y}, Σ = {a, b}. Two strings s = xbbxa and s' = ybbya are parameterized matches using the bijections: x->y.

Baker in [6] introduced the concept of predecessor string to simplify this bijective mapping. According to Baker, any two parameterized strings are p-matches with each other if their predecessor strings matches exactly. Predecessor string

for a given string S is denoted by prev(S) and calculated as: if a character at position i have already occurred at position j then predecessor string contain i-j at position i. Otherwise predecessor string contain 0. For example, for the string S = "axxbxyay", where x and y are parameterized symbol, predecessor string is prev (S) = a01b20a2. Also, prev (ayybyxax), the prev is a01b20a2. Hence string "axxbxyay" p-matches with "ayybyxax".

Parameterized matching is useful in software maintenance, color image and plagiarism detections, Computational Biology. Parameterized matching problem was first time introduced by Baker [6] as an application to software maintenance that was used to find duplication present in the large program. Two section of program are considered to be equivalent if variables and identifiers of one can be renamed in such a way that they can be transferred into the other. She developed a framework to efficiently solve the problem in 1D that includes parameterized suffix trees.

Parameterized matching has many important applications in real life. With the fast growth of text collections, chances of plagiarism, duplicity, cloning, and software maintenance etc., become high [7]. According to a clone detection survey [8], [9] parameterized matching has been very useful in detecting clones in software. In the field of molecular biology, it is said that two biological sequences tend to have similar properties, if they have similar 3-D structure. It is useful to find not only similar sequences in the string sense, but also structurally similar sequences from the database [10]. In image processing, two different images may have different colors, even if their basic structures are same. In the software maintenance [6], it is required to find the equivalency between two sections of codes. Here equivalency means that one section can be transformed into the other via some one-to-one correspondence. All these said problems can be efficiently solved with the help of parameterized matching.

B. Boyer-Moore Algorithm (BM)

It was developed by Robert S. Boyer and J Strother Moore [11]. Boyer –Moore algorithm was the first sub linear string matching algorithm. Boyer-Moore algorithm processes the window of length m in the text. The basic idea of this algorithm is that it processes the window of length m in the text and in each window the characters are read from right to left. During preprocessing algorithm creates two tables, Boyer-Moore bad character (bmBc) and Boyer-Moore good-suffix (bmGs) tables. A bad-character table stores the shift value based on the occurrence of the character in the pattern, while a good-suffix table stores the matching shift value for each character in the pattern. If the character of text that is being compared with the pattern does not occur in the pattern at all, then shift the pattern by m position ahead. Whenever a mismatch occurs, algorithm considers both bad character and good suffix table and take the larger one as a shift.

This algorithm forms the basis for several pattern-matching algorithms. Therefore, the algorithm compares the pattern P with the substring of sequence T within a sliding window in the right-to-left order. The bad character rule table and good suffix rule table are used to determine the movement of sliding window.

Bad character rule

This rule is applied if the character of text that causes mismatch present somewhere else in the pattern. In this case pattern is to be shifted so that the bad character is aligned with its occurrence in the pattern. For example, as shown in table 2.1, P= abbac

Table 2.1 Bad character shift table for boyer-moore algorithm

Char{c}	A	B	c	*
bC{c}	1	2	5	5

Let T= ababbacab
i = 0 1 2 3 4 5 6 7 8;
T= a b a b b a c a b
P= a b b a c
P= a b b a c

Comparison between b and c causes a mismatch. Text character b occurs in the pattern at position 1 and 2. Shift value of: BCS[b] = 2. The pattern is shifted by 2, so that the right most b in the pattern is aligned to text character b.

Good suffix rule

There are two cases:
Case 1: The matching suffix occurs somewhere else in the pattern.
Sometimes there may be a situation when the bad character produces a negative shift.
For example,
T= a b a a b a c a b
P= c a b a b
P= c a b a b

Here, an alignment of the rightmost occurrences of the pattern character "a" with the text symbol "b" would produce a negative shift of -1.However, suffix "ab" of text window which has matched with the pattern is also present as a substring in the pattern at position 1. So if we align this substring with the matched suffix of the text window, a shift of 2 is possible. Therefore, we take maximum of -1 and 2 to decide the shift.

Case 2: Only a part of the matching suffix occurs at the beginning of the pattern.

Sometime, some suffix of matched suffix is prefix of the pattern. In this case we take the shift as maximum of bad character and good suffix shift.

For example:

T= b a a b b b a b a c

P= b b a b b

P= b b a b b

Here, the suffix "abb" of the text window is matched with the pattern and any other occurrence of "abb" is not present in the pattern. But prefix of the pattern"bb" matches with the end of "abb". So we take maximum shift = max{2,1]=2.

C. Wu-Manber Algorithm (WM)

Wu-Manber (WM) algorithm[12] was proposed by Sun Wu and Udi Manber in 1994. It uses the jump idea of BM and hash function. WM algorithm is one of the most quickly multiple pattern matching algorithms. In addition, WM algorithm is not sensitive to the character set. WM algorithm has two core mechanisms, the filter mechanism based on hash technology and the block character shift mechanism based on the bad character shift technology from BM algorithm. By calculating the hash value of the suffix block character in patterns, the patterns with same suffix are linked in a list. In hash table all entries in the list are stored.

When searching a text, we calculate the hash value of the block character inside current match window, and lookup the hash table to get the entry of the patterns that contain the same block character as their suffix. Obviously, these patterns are possible matching strings. Symbol "B" is used to represent the size of the block character. It usually is 2 or 3. When a matching is finished, the match window should right shift. The size of match window is determined by the length of the shortest pattern. Here we suppose it is "m". The bad character shift technology is used for the match window shift.

During the preprocessing, the size of the match window should be determined first and three important tables should be established, a SHIFT table, a HASH table and a PREFIX table. The SHIFT table stores the shift distance of the block character occurred in text. The HASH table stores the entry of the link list which links all the patterns with the same suffix inside the match window. The PREFIX table stores the entry of the link list which links all the patterns with the same prefix inside the match window.

Given text T = abcaababbac and pattern set P = {abc, bbac, bcaa }. During preprocessing stage, first we compute minlength of a pattern and considered only first m character of each pattern. Here Minlength (m) = 3. Minlength substrings are:{"abc", "bba", "bca"}. Now, we construct shift table, it contains all possible string of size B. Here B = 2, extract all distinct 2 byte sequence from minlength substrings: {"ab", "bc", "bb", "ba", "ca"}. If during scanning phase, input pair is not one of these then we shift the pointer by minlength–1 bytes.Shift table is constructed by using the following function: Shift[xy] = minlength- qxy, Where qxy is the rightmost ending position of xy in any pattern substring. SHIFT values are shown in Table 2.2.

Table 2.2 Bad character shift table for wu-manber algorithm

ab	ba	bb	ba	ca
1	0	1	0	0

When shift[xy] = 0, we match some patterns. During Scanning stage, T = abcaababbac. Shift[ab] = 1 shift by 1. Now pointer points to b at position 2 in the text. Shift[bc] = 0. Since we get shift value = 0 it means there is a match. So we check in which pattern B is a substring. We found that B is a substring of p3. Compare p3 against text, test succeeds for p3 at position 4. Shift the pointer by 1. Now pointer points to c in the text, shift[ca] = 0 . We found that B is a substring of p3. Compare p3 against text, no match occurs, shift by 1. Now pointer points to a in the text, shift[aa]= 2 , since B does not appear as substring in any string of patterns, we can safely shift minlength-1 character to right so shift the pointer by 2.

D. Quick Multiple Matching Algorithm (QMM)

Liuling Dai[13] has developed QMM algorithm in 2008. QMM is the algorithm for multiple string matching problems using the concept of quick search algorithm. It is an extension of Wu-Manber algorithm that maximizes the shift distance. The QS algorithm improves the BM algorithm by examining the character next to the scan window after each test to enlarge the average shift distance. In practice, the QS algorithm is very simple and fast.

It uses 3 tables: HASH, PREFIX and SHIFT. HASH and PREFIX table are used exactly in the same manner as in Wu-Manber algorithm. The only difference is in SHIFT table. Shift table is constructed in an aggressive manner to get larger shift distance. Shift distance is independent of whether the match occurs or not. It is only decided by SBC. SBC (subsequent-block-character) is the string that consists of last character of the scan window and one character success it. To utilize the character after the current scan window, after each test, we compute the shift distance based on the SBC of the scan window.

E. Parameterized Boyer-Moore Horspool Algorithm

This algorithm [14] has been developed by L.Salmela in 2008. L.Salmela extended the Horspool variant of Boyer-Moore algorithm into a parameterized string matching problem. This algorithm is for single pattern parameterized matching and based on the concept of Boyer-Moore-Horspool algorithm. Boyer-Moore-Horspool algorithm first reads

the last character of scan window of length m. If it does not matches with the last character of pattern then shift the scan window based on the last character aligned with the pattern.

Parameterized matching algorithm uses the concept of last q-gram since last or even two last characters may not indicate that, window cannot match. Limitation of this algorithm is that it is applicable only for single pattern. This algorithm uses the concept of q-gram and uses only parameterized alphabet. As in parameterized matching, two strings p-match, if their prev-encoded strings match exactly. Thus algorithm constructs a shift table (used in Horspool) with the prev-encoded strings.

Two strings are p-match if their predecessor strings are exactly match. So, we wish to index the shift table with the predecessor strings.

Using Hashing Technique

The algorithm is known as Parameterized Boyer-Moore-Horspool with Hash (PBMH-Hash). The minimum shifts stored in the shift table for those q-grams that have same hash value will be smaller than the shift values without hashing. We perform hashing by transforming q-gram first into the predecessor strings and then adding up all the positions of the predecessor string.

For example the q-gram “acgac” is transformed into an index using this hashing scheme. First “acgac” is transformed into the predecessor string “00033” after that we add up all the characters of the predecessor string which yield to the index value 6.

Example: Consider the pattern P = “abaaba" and text T = “bbabbababa". Here, all the symbols are taken as parameterized symbols. Let q = 3. The preview encoding of the q-grams of the pattern are: aba = 002, baa = 001, aab = 010, aba = 002. By reserving enough bits for each characters we get prev(P) = “2142". The bad character shift table is constructed by Horspool algorithm [15]. Table 2.3 shows the bad character shift table.

Table 2.3 Bad character shift table for horspool algorithm
(*meant to represent any other character)

Char (c)	2’	1’	4’	*
hpBC[c]	3	2	1	4

III. PROPOSED ALGORITHM

In this paper we propose an algorithm for handling multiple strings as well as parameterized string matching. This proposed algorithm is an extension of Quick Multiple Matching Algorithm for exact string matching. In this we use the concept of parameterized and hashed version of Boyer-Moore-Horspool algorithm for handling the parameterized patterns. Our algorithm consists of two parts one is pre-processing part and the other is scanning part.

During pre-processing transformation into predecessor string takes place and also PREFIX table is formed. Minimum length is calculated and then all min length substrings are extracted and are transformed into predecessor string. These predecessor strings are used for the construction of SHIFT table. The algorithm for pre-processing part is as follows:

```
1. for i←0 to n-1
2. do preEncode_Original pattern = preEncode (pattern[i], 0)
3.     preEncode_pattern = preEncode (pattern[i], minLength)
4.     for i ← 0 to n-1
5.     do for j ← 0 to (minLength - qGram)
6.     do preEncode_qGram[i][j] = preEncode(pattern[i][j], qGram)
7.     for i←0 to n-1
8.     do if (pattern[i][0] == pattern[i][1])
9.     prefix_table[i] = 1
10. else
11. prefix_table[i] = 0
```

During scanning part length of the scan window is set to m. First 2-length prefix of the scan window is converted into a predecessor string and then the hash value is computed. Then it is checked that whether this hash value is equal to any of the prefix value in the prefix table and if it matches with any of the prefix then the corresponding pattern is matched against the text.

IV. IMPLEMENTATION AND RESULT

The proposed algorithm is implemented in C on windows 7 having 1.73GHZ processor and 4GB RAM. The algorithm is executed on different files of alphabet size 2 i.e binary file, alphabet size 4 i.e DNA file and alphabet size 256 i.e bible file. Table 3 shows the execution time of the algorithm for different alphabet size of fixed pattern length. Here, we have assumed pattern length as 5. The table 4.1 and fig. 4.1 shows that on increasing the alphabet size the execution decrease. Here, it is, minimum for alphabet size 256 and maximum for alphabet size 2.

TABLE 4.1 EXECUTION TIME (MS) OF PROPOSED ALGORITHM (FOR MULTIPLE PATTERN OF FIXED LENGTH (5) ON DIFFERENT ALPHABET SIZE)

Number of Pattern	Execution Time (s)		
	Alphabet size 2	Alphabet size 4	Alphabet size 256
1	0.824176	0.769231	0.384615
2	1.153846	0.989011	0.714286
3	1.538462	1.208791	0.769231
4	1.648352	1.428571	1.098901
5	1.813187	1.648352	1.153846
6	2.307692	1.758242	1.318681
7	2.527473	1.978022	1.648352

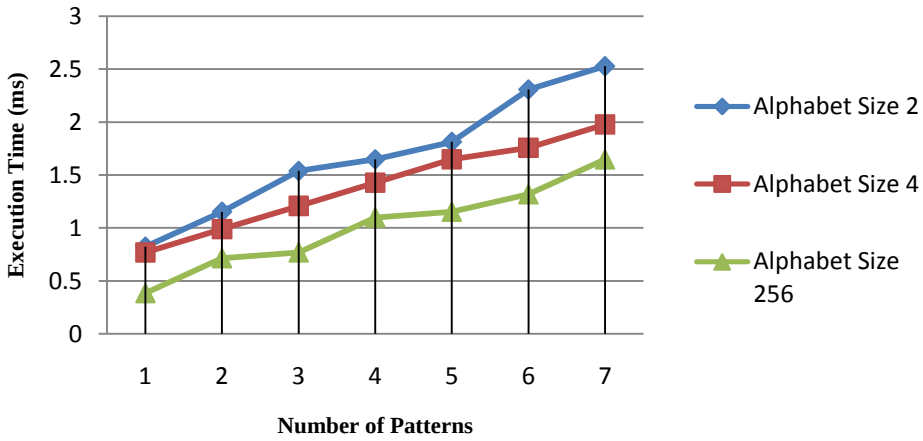

Fig. 4.1 Graph showing the variation in execution time

V. CONCLUSION

In this paper, a new algorithm has been proposed for multi-pattern parameterized string matching. The same algorithm has been implemented in C and analyzed over different files against various parameters like length of pattern, number of pattern, alphabet sized. The results obtained as in Table 4.1 and in Fig.4.1 shows that the execution time of the proposed algorithm is increasing with the increase in number of patterns and the same is decreasing with the increase in alphabet size. It is concluded that the execution time minimum for the alphabet size 256 i.e general text.

REFERENCES

[1] X. S. Sun, Q. Wang, Y. Guan and X. L. Wang, "An improved Wu-Manber multiple- pattern matching algorithm and its application,"Journal of Chinese Information Processing, Beijing, vol. 20, no. 2, pp. 47-52, 2006.

[2] A.V. Aho and M. J. Corasick, "Efficient string matching: an aid to bibliographic search," Communications of the ACM, vol. 18, pp. 333-340, 1975.

[3] S. Wu and U. Manber, A fast algorithm for multi-pattern searching, Technical Report TR-94-17, Department of Computer Science, University of Arizona, 1994.

[4] Liuling Dai,"An aggressive algorithm for multiple string matching", ELSEVIER, Information Processing Letters, January 2009.

[5] Salmela, Leena, and Jorma Tarhio. "Fast parameterized matching with q-grams." *Journal of Discrete Algorithms* 6.3 (2008): 408-419.

[6] Baker, B. S. Parameterized duplication in strings: algorithms and an application to software maintenance.SIAM Journal of Computing 26, 5 (1997), 1343-1362.

[7] Mccreight, E. A space-economical suffix tree construction algorithm. Journal of the ACM 23, 2 (1976), 262-272.

[8] Koschke, R. Survey of Research on Software Clones.Dagstuhl Seminar Proceedings 06301, Duplication, Redundancy, and Similarity in Software (2007), pp.1-4.

[9] Navarro, G., And Raffinot, M. A Bit-parallel approach to Suffix Automata:Fast Extended String Matching. In Proceedings of Combinatorial PatternMatching, Lecture Notes in Computer Science 1448 (1998), pp. 14-33.

[10] Amir, A., Aumann, Y., Cole, R, Lewenstein, M., And Porat, E. Function Matching: Algorithms, Applications, and a Lower Bound. J.C.M. Baeten etal. (Eds.): ICALP, Lecture Notes in Computer Sciece 2719 (2003), Springer-Verlag Berlin Heidelberg, pp. 929-942.

[11] Boyer, R. S., And Moore, J. S. A fast string-searching algorithm. Communication of ACM 20, 10 (1977), 762-772.

[12] R. Prasad, S. Agarwal, S. Misra, A. K. Sharma, A. Singh ," Maintaining software through bit-parallelism and hashing the parameterized q-grams", Tehnicki vjesnik, Vol.19 No.2 Lipanj 2012.

[13] Liuling Dai,"An aggressive algorithm for multiple string matching", ELSEVIER, Information Processing Letters, January 2009.

[14] Lecroq, T. A variation on the Boyer-Moore algorithm.Theoretical Computer Science 92, 1(1992), 119-144.

[15] Horspool, R. N. Practical fast searching in strings. Software-Practice & Experience 10, 6 (1980), 501-506.

