Efficient Parameterized Word Matching Using Bit-Parallelism and Partitioning the Text

Rajesh Prasad
Department of Computer Science
Yobe State University
Damaturu, Nigeria
rajesh_ucer@yahoo.com

***Abstract*— Word matching problem is to find all the exact occurrences of a pattern P[0…m-1] in the text T[0…n-1], where P neither contains any white space nor preceded and followed by space. In the parameterized word matching problem, a given word P[0…m-1] is said to match with a sub-word t of the text T[0…n-1], if there exists a one-to-one correspondence between the symbols of P and the symbols of t. Exact Word Matching (EWM) problem has been previously solved by partitioning the text into number of tables in the pre-processing phase and then applying either brute force approach or fast hashing during the searching process. This paper presents an extension of EWM problem for parameterized word matching. It first split the text into number of tables in the pre-processing phase and then applying prev-encoding and bit-parallelism technique, Parameterized Shift-Or (PSO) during the searching phase. Experimental results show that this technique performs better than PSO.**

***Index Terms*— Algorithm, string matching, prev-encoding, bit-parallelism, shift-or**

I. INTRODUCTION

Exact string matching problem is to find all the occurrences of a pattern P[0…m-1] in the text T[0…n-1], where m ≤ n and symbols of P and T are drawn from some finite alphabet Σ. Many algorithms for solving this problem exists [1, 2, 3, 4, 5]. Word matching problem is to find all the occurrences of a pattern P[0…m-1] in the text T[0…n-1], where P neither contains any white space nor preceded and followed by space. It is very important in information retrieval, as words are basic unit of information. It has been studied by various researchers on uncompressed and compressed data [14, 15, 16]. Recently, word searching problem in [9] has been solved by splitting the text into number of tables in the preprocessing phase and in the searching phase, the algorithm performs character to character comparisons with words of same length in the table by brute force manner. Later in [6], the same problem is solved by using fast hashing function (SDBM) [10, 11, 12] in the searching phase. The use of fast hashing improves the running time of the algorithm. In [7], word searching problem for multiple word has been solved by the use of bit-parallelism.

Another variation of exact string matching is the parameterized string matching. In the parameterized matching problem, two strings are said to be parameterized match, if one can be transformed into other by using some one-to-one mapping. It was first studied by Baker in [17] as a tool in software maintenance. Later on it was studied by several researchers in [18, 19, 20, 21]. In [22], Beal, and Adjeroh solved this problem by using LZW algorithm. Parameterized string matching problem has been previously solved in [13] by using the concept of bit-parallelism and prev-encoding both the pattern and the text.

This paper solves the parameterized word matching problem by first splitting the text into number of tables depending on the size of text in the pre-processing phase and then applying efficient Parameterized Shift-Or (PSO) [13] algorithm, with little bit modification, during searching phase. Here prev-encoding of whole text is not done in advance and is done during searching. The proposed algorithm is implemented on Pentium D, 3.00 GHz machine with 512 MB RAM, and running Fedora 8. Experimental results show that this is much faster than the previous existing PSO algorithm. This paper assume that pattern length (m) is less than or equal to word length (w) of computer used and text being processed is offline.

This paper is organized as follows: Sec. 2 presents related concepts and algorithms. Sec. 3 explains proposed algorithm. Sec 4 presents experimental results. Finally, Sec. 5 concludes the work.

II. RELATED CONCEPTS

Shift-or String Matching Algorithm

Terms Used:

- $b_{m-1}b_{m-2}\ldots b_1b_0$ denotes bits of computer word of length m.
- Exponentiation is used to denote bit repetition (e.g. $0^41=00001$)
- C-like syntax is used for operations on the bits of computer words: "|" is for bit-wise or, "&" is for bit-wise and, "~" complements of all the bits. The shift left operation, "<<r", moves all bits to the left by r and enters 'r' zeros in the right.

In the shift-or algorithm, an automaton is constructed as follows: The automaton has states 0, 1, 2…m. The state 0 is the initial state, state m is the final state and ∀ i = 0…m-1, there is a transition from state i to state i+1 for character P[i]. In addition, there is a transition for every c ∈ Σ from and to the initial state, which makes the automaton non-

IJECE

deterministic. For example, for the pattern P = ba, the shift-or automaton is shown in Fig. 1.

The preprocessing algorithm builds a table B having one bit mask entry for each $c \in \Sigma$. For $0 \le i \le m-1$, the mask B[c] has i^{th} bit set to 0 iff P[i] = c otherwise it is 1. If the i^{th} bit in B[c] is zero, then in automaton, there is a transition from the state i to i+1 with character c.

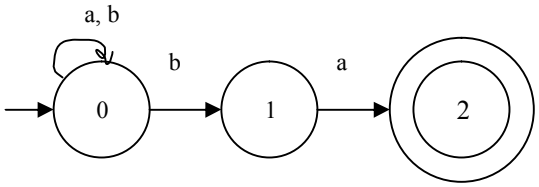


Fig. 1: A Shift-Or Automaton for P = ba

For searching, algorithm needs a bit mask D so that i^{th} bit of this mask is set to 1, if and only if state i in NFA is active. For each text symbol c the state vector D is updated by $D \leftarrow (D << 1) \mid B[c]$, where $D << 1$, makes the state active and then looks for transition on that state. If after the i^{th} step (after processing the i^{th} symbol of the text), the $(m-1)^{th}$ bit of D is zero, then there is an occurrence of P with shift i-m. If $m \le w$, then the running time of the algorithm is O(n). In general, running time of the algorithm is $O(n\lceil m/w \rceil)$.

Exact Word Searching Algorithm

In this section, we explain the recent optimal word searching algorithm (WSA) given by Ibrahiem M.M. El Emary et al. [6]. In this algorithm, the given text is assumed to be offline. This algorithm has two phases that works as follows: The first phase which is preprocessing phase starts with reading and splitting the text into k number of equal parts depending on size of the text and constructing k tables with two columns for each part of the text. The first column contains length of the word and the second contain the starting position of each word in the text. The start positions of the words will be classified by the same length. Once the table is constructed, it is sorted 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. The algorithm calculates the length of the pattern and search for the same length in the tables starting from the first table. If the length does not exist in the first table then the algorithm searches for the word in the next table and so on. If the current table is the last table, then a message will appear denoting that the pattern does not exist. On the other hand if the length exists in the table, then the algorithm will retrieve the words in the text using the stored start position in the table and begin to compare. If a full match occurred then occurrence of the pattern is reported. But if a mismatch happen at some character, then the algorithm will skip to the next start position and compare again. Ibrahiem M.M. El Emary et al. compare their algorithm with Boyer Moore in terms of number of character comparison and show that their algorithm takes very less number of character comparisons than Boyer Moore algorithm.

Parameterized String Matching Problem

This section presents small introduction to parameterized matching [3, 8] problem. Here we assume that all the symbols of P[0…m-1] and T[0…n-1] are taken from $\Sigma \cup \Pi$, where Σ is fixed symbol alphabet of size σ and Π is parameter symbol alphabet of size π. A pattern P matches the text substring T[j…j+m-1], for $0 \le j \le n-m$, if and only if $\forall\, i \in \{0, 1, 2, \ldots m-1\}$, $f_j(P[i]=T[j+i])$, where $f_j(.)$ is a bijective mapping on $\Sigma \cup \Pi$. It must be identity on Σ but need not be identity on Π. For example, let P = XYABX on Σ = {A, B} and Π = {X, Y, Z, W}. Pattern P matches the text substring ZWABZ with bijective mapping $X \rightarrow Z$ and $Y \rightarrow W$. This mapping can be simplified by *prev*-encoding [3]. For any string S, *prev*(S) maps its each parameter symbols s to a non-negative integer p, where p is the number of symbols since the last occurrences of s in S. The first occurrence of any parameter symbol in *prev*-encoding is encoded as 0 and if $s \in \Sigma$ it is mapped to itself (i.e. to s). For example, *prev*(P) = 00AB4 and *prev*(ZWABZ) = 00AB4. With this scheme of *prev*- encoding, the problem of the parameterized string matching can be transformed to the exact string matching problem, where *prev*(P) is matched against *prev*(T[j…j+m-1]), for $0 \le j \le n-m$. The *prev(*P) and the *prev*(T[j…j+m-1]) can be recursively updated as j increases with the help of following lemma in [3].

Lemma 1: Let S′ = prev(S). Then for S″ = prev(S [j…j+m-1]) for all i such that S [i] $\in \Pi$ it holds that S″ [i] = S′ [i] iff S″ [i] < m, otherwise S″ [i] = 0.

Parameterized Shift-or Algorithm

This section presents Parameterized Shift-Or (PSO) string matching algorithm [6]. The algorithm is generalization of the algorithm explained in Section 2. It has been generalized in the following way:

i. The pattern P is encoded by *prev*-encoding and stored as *prev*(P). In order to compute *prev*(P), an array *prev*[c] is formed, which for each symbol $c \in \Sigma$, stores the position of its last occurrence in P. For example, for the pattern P = XAYBX with Σ = {A, B} and Π = {X, Y}, *prev* (P) = 0A0B4.
ii. For all j = 0, 1, 2...n-m, *prev* (T[j…j+m-1]) can be efficiently *prev*-encoded by Lemma 1.
iii. The table B is built such that all the parameterized pattern prefixes can be searched in parallel. To simplify indexing into array B, it is assumed that Σ = {0, 1…σ-1}, and *prev*-encoded parameter offsets are mapped into the range {σ…σ+m-1}. For this purpose, an array *Arr*[0…σ+m-1] is formed, in which the positions 0…σ-1 are occupied by element of Σ and the rest positions are occupied by *prev*-encoded offsets.

For example, for the pattern P = XAYBX on Σ = {A, B} and $\prod$ = {X, Y}, with σ = 2, m = 5, the *prev*-encoded pattern P is *prev*(P) = 0A0B4. The array *Arr* looks like as:

IRJECE

A	B	0	1	2	3	4
0	1	2	3	4	5	6

Searching for P′= *prev*(P) in T′ = *prev*(T) can′t be done directly as explained below. Let the pattern P = XAXX and the text T = ZZAZZAZZ. P′ = 0A21 and T′ = 01A21A21. Obviously, P has two overlapping parameterized occurrences in T (one with shift = 1 and another with shift = 4) but P′ does not have any occurrences in T′. The problem occurs because of when searching for all the m prefixes of the text in parallel, then some non-zero encoded offset p in T′ should be interpreted as zero in some case. For example, when searching for P′ in T′[1…4] = 1A21, 1(from left) should be zero.

The solution to this problem is that, the lemma 1 should be applied in parallel to all m-length sub-strings of T′. This can be achieved in the following way. The bit vector B[*Arr*[σ+i]] is the match vector for *Arr*[i], where $0 \le i \le \sigma+m-1$. If the j^{th} bit of this vector is zero, it means that P′[j] = *Arr*[i]. If any of the i^{th} least significant bit of B[*Arr*[σ]] is zero then corresponding bit of B[*Arr*[σ+i]] is also cleared. This can be achieved as: B[*Arr*[σ+i]] ← B[*Arr*[σ+i]] & (B[*Arr*[σ]] | (~ 0 << i)) which signifies that for $\sigma \le i \le \sigma+m-1$, *Arr*[i] is treated as *Arr*[i] for prefixes whose length is greater than *Arr*[i] and as zero for shorter prefixes thus satisfies Lemma 1.

III. PROPOSED ALGORITHM (PWSA)

In this section, we present the proposed Parameterized Word Searching Algorithm (PWSA). In our proposed algorithm: the whole text is splitted into k equal parts according to size of text T. For each part, a table having two columns is created. In the preprocessing phase: each part of the text is read. While reading, length of word is computed and according to length of the word, starting positions of the words is stored in table. Starting position of all words of same length are stored in same row corresponding to length of the words in their table. Pattern is preprocessed as in PSO. In the searching phase a modified parameterized Shift-or (PSO) algorithm is used. For a word (P) of length m (where m is the pattern length) in the table, extract the word W using its stored starting position. Take the prev-encoding of the word (W).

Take a bit-vector D, initialize D ← ~ 0. Now using PSO, check the occurrence of pattern (P) in W. To check this take the m-th bit of vector D, if this bit is 0 then the pattern exist.

For each table repeat until words of same length are checked i.e. all words of the length of pattern are checked. Algorithm 1 and Algorithm 2 given below present the pseudo code of algorithm.

Algorithm 1 (PWSA, Parameterized Word Searching Algorithm)

```
MPWSA (T, P)
1 n ← length(T)
2 m ← length(P)
3 Create_Table(T, n)
4 for i ← 0 to σ-1
5         do B[i] ← ~0
6 for i ← 0 to m-1
7         do B[P[i]] ← B[P[i]] & ~(1<<i)
8 for each table
9  do found ← Search_mth_index (Table, m)
10     if(found)
11    then for each entry
12    do
13     w ← Extract_word()
14     D ← ~0
15     mm ← 1 << (m-1)
16     i ← 0
17    while (i<m)
18     do D ← (D<<1) | B[prev[T[i]]]
19       if(D && mm) ≠ mm
20         then report match at position i-m+1
23 end for
```

Algorithm 2 (Table creation using starting position of words)

```
Create_Table (T, n)
1 n ← length(T)
2 current_table ← table 1
3 k ← 0
4 for i ← 0 to n
5  do k ← k+1
6      w ← Extract_word()
7      if (k > x) then /*x is length of text per table*/
8          k ← 0
9          current_table ← current_table+1
10         l ← length(w)
11         Update (current_table, l, i )
12         i ← i+1
13 end for
```

The Example

The idea of searching a certain pattern in details is given. Implementation of the algorithm requires two phases: Suppose we divide our text into two equal parts. For each part we construct a table. The row index represent the length of the word in the text T. Starting position of a word is stored in the table. Suppose alphabet Π = {a} and rest symbols are taken in the set of Σ.

ram	*is*	*the*	*King*	*of*	*India*
1	*5*	*8*	*12*	*17*	*20*
	he	*has*	*two*	*Sons*	
	26	*29*	*33*	*37*	

In the above example, for each part first line is part of the text and second line shows the starting position of words in the text. Using this we create tables as follows:

IJIRECE

Table I. Part 1 Of the Text

Length	Starting Position
2	5, 17
3	1, 8
4	12
5	20

Table II. Part 2 of the Text

Length	Starting Position
2	26
3	29, 33
4	37

Consider the patterns P = ram. Prev-encoding of P is r0m. Now from Table 1, we extract a word from length three. W="ram" and W = "the" are extracted. We take prev-encoding of both to get r0m and the. After applying PSO, we get that the pattern occur at position 1.

IV. EXPERIMENTAL RESULTS

Proposed algorithm PWSA have been implemented in C++, compiled with GCC 4.2.4 compilers on the Pentium 4, 2.14 GHz processor (word length w = 32) with 512 MB RAM, running ubuntu 10.04. It is compared with existing algorithm PSO. The patterns and text are chosen from ASCII character set randomly.

In order to evaluate the efficiency of the proposed algorithm, several experiments are performed on three kinds of texts:

- Alphabet of size 4 characters: a random file of size 12 KB over alphabet {a, b, c, d} was created.
- Alphabet of size 2 characters: text is composed of two characters 0 and 1 that are randomly created. The size of file is equal to 12 KB.
- Alphabet of size 256 characters: Bible files of size 12 KB is taken from www.BibleTexts.com.

To evaluate algorithms thoroughly, these algorithms are executed with different pattern-set-length and different size of pattern set. In each run, the pattern set X is randomly created.

Table III shows the comparison between existing bit-parallel algorithm PSO and PWSA. Table IV shows the effect on time after increasing the alphabet size. Table V shows the comparison between PSO and PWSA on increasing the alphabet size.

TABLE III. COMPARISON TIME(S) BETWEEN PSO AND PWSA (ALPHABET SIZE = 4, IS FIXED)

Parameterized Set	Pattern Length	Time (s) in PSO	Time (s) in PWSA
{a}	10	5.097	3.024
	20	5.271	3.5
	30	5.823	4.00
{a, b}	10	5.271	3.004
	20	5.529	3.448
	30	5.932	3.998
{a, b, c}	10	5.467	3.972
	20	5.924	3.064
	30	5.883	3.960
{a, b, c, d}	10	5.394	3.004
	20	5.672	3.504
	30	5.982	3.184

TABLE IV. EXECUTION TIME(S) OF (PWSA) ALGORITHM ON DIFFERENT ALPHABET

Length of Pattern	Execution Time (alphabet size=2)	Execution Time (alphabet size=4)	Execution Time (alphabet size=256)
4	0.841026	0.841020	0.859341
6	0.716483	0.639560	0.694505
8	0.712821	0.639560	0.638095
10	0.659341	0.521245	0.401465
12	0.586081	0.366312	0.283150

TABLE V. EFFECT ON RUNNING TIME (S) AFTER INCREASING THE ALPHABET SIZE (PARAMETER {A, B} ARE FIXED, PATTERN LENGTH =10 IS FIXED)

Alphabet Size	Time (s) in PWSA	Time (s) in PSO
2	0.641026	0.659341
4	0.439560	0.494505
256	0.239560	0.438095

IRJECE

V. CONCLUSION

This paper, extended WSA algorithm of exact string matching for the parameterized string matching. It compares the proposed algorithm (PWSA) with Parameterized Shift-Or (PSO) algorithm for the case, when the pattern length is within the word size (w) and for varying alphabet size. From Table III, it is clear that our proposed algorithm PWSA is better than PSO. From Table IV, it is clear that on increasing the alphabet size, time decrease.

ACKNOWLEDGMENT

I wish to thank Prof. Yakubu Mukhtar, Vice Chancellor of Yobe State University, Damaturu, Yobe State, Nigeria for consistent financial and other supports. I also wish to thank Tertiary Education Trust Fund (TETFUND), Nigeria for providing fund to attend this conference.

REFERENCES

[1] R.A. Baeza-Yates and G.H. Gonnet, "A new approach to text searching," Communication of ACM, 35(10), pp. 74-82, 1992.

[2] Thomas H. Cormen, Charles E. Lieserson, Ronald L. Rivest and Clifford Stein, Introduction to Algorithms. MIT Press, Second Edition, 2001.

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

[4] A.V.Aho, and M.J. Corasick, "Efficient String Matching: An aid to bibliographic search," Communication of ACM 18(6), pp. 333-340, 1975.

[5] R. Prasad and S. Agarwal, "An Efficient String Matching by using Super Alphabet," proc. of the first International Conference on Emerging Trends in Engineering and Technology (available on IEEE Xplore), Nagpur, India., pp. 1181-1186, July 16-18, 2008.

[6] R. Prasad and S. Agarwal, "An Efficient Word Searching algorithm through splitting and hashing the offline text," Proc. of ARTCOM 09, India, pp. 387-389, 2009.

[7] R. Prasad and S. Agarwal, "An Efficient Bit-parallel Multi-patterns Word Searching algorithm through splitting the text," Proc. of ARTCOM 09, India, pp. 406-410, 2009.

[8] Kimmo Fredriksson, "Shift-or string matching with super alphabets," Information Processing Letters (IPL), 87(4), pp. 201-20*4, 2003*.

[9] Ibrahiem M. M. Emary and Mohammed S. M. Jaber, "A New Approach for Solving String Matching Problem through Splitting the Unchangeable Text," World Applied Sciences Journal 4 (5): 626-633, 2008

[10] R. J. Enbody and H. C. Du, "Dynamic Hashing Schemes," ACM Computing Surveys, vol. 20, No.. 2, 85-113, 1988.

[11] P. A. Larson, "Dynamic Hashing," BIT, vol. 18, 184-201, 1978.

[12] Donald E. Knuth, Sorting and Searching, volume 3 of "The Art of Computer programming," Addison Wesley, 1981.

[13] K. Fredriksson and M. Mozgovoy, "Efficient parameterized string matching," Information Processing Letters (IPL), 100(3), pp. 91-96, 2006.

[14] A. Gupta, and S. Agarwal, "A Scheme That Facilitates Searching And Partial Decompression of Textual Documents," *Intl. Journal of Advanced Computer Engineering,* 1(2), 2008.

[15] R. Garg, R. Prasad and S. Agarwal, "Fast Parameterized Word Matching on Compressed Text," 5th International Confeence on Computer and Communication Technology, September 26-28, India, pp. 317-321, 2014.

[16] R. Prasad and R. Garg, "Efficient Multi-word Parameterized Matching on Compressed Text," 6th IEEE ICAST, October 29-31, Ota, Nigeria, pp. 1-6, 2014.

[17] B. S. Baker, "A theory of parameterized pattern matching: Algorithms and applications," Proceedings of the 25th ACM Symposium on the Theory of Computation, ACM Press, New York, pp. 71–80, 1993.

[18] B. S. Baker, "Parameterized diff.," Proceedings of the 10th Annual ACM–SIAM Symposium on Discrete Algorithms, SIAM, Philadelphia, pp. 854–855, 1999

[19] A. Amir, M. Farach, S. Muthukrishnan, "Alphabet Dependence in Parameterized Matching," Information Processing Letter 49(3): 111-115, 1994.

[20] R. M. Idury, A. A. Schäffer, "Multiple Matching of Parameterized Patterns," CPM, pp. 226-239, 1994.

[21] Brenda S. Baker, "Parameterized Pattern Matching: Algorithms and Applications," J. Comput. Syst. Sci. 52(1): 28-42, 1996.

[22] R. Beal, and D. A. Adjeroh, "Compressed Parameterized Pattern Matching," 2013 IEEE Data Compression Conference, March 20-22, pp. 461-470, 2013.

IRJECE