A Composite Boyer-Moore Algorithm for the String Matching Problem

Zhengda Xiong
College of Computer Science and Technology
Huazhong University of Science and Technology, Wuhan,430074, China
xzda72@sina.com

***Abstract*—The string matching problem has found wide application in computer science, molecular biology, genetic engineering, semantics and many other fields. In this paper, we give analysis to several classical algorithms, KMP, BM and their improvements. Then, by compositing the main method of the BM algorithm, we propose a new algorithm — the Composite BM algorithm (CBM). Differing from the BM algorithm that only uses current matching information, the CBM algorithm tries to take full advantage of the historical matching information. In this way, CBM accelerates the forward speed of the pattern during the matching, and hence the whole string matching process is speeded up effectively. Finally, for binary matching, we made random test to BM and CBM, the result shown the efficiency of CBM is higher than of BM.**

Keywords: string matching; Pattern matching; Boyer-Moore Algorithm; CBM algorithm

I. INTRODUCTION

String matching is the most basic problem in the area of pattern matching. Not only is it widely used in the application of data compression, text editing, information retrieval etc, but it also plays an important role in the area of network security. Therefore, string matching algorithm is a basic component in many operating systems. Furthermore, string matching techniques also have some relations with other string operation problems. Including its application in the field of computer science, string matching problem also finds important applications in the field of molecular biology, genetic engineering [5,7], semantics and etc. Designing efficient and effective string matching algorithm will not only be very useful for pattern matching but also promote relevant application's development.

II. PROBLEM DESCRIPTION

String matching is a problem of finding occurrence(s) of a pattern string within another string or body of text. The problem is also known as exact string matching, string searching, and text searching. It can be formalized as follows:

Input: pattern P and text body T, whose characters were from alphabet Σ, a finite non-empty set. Let $P = p_1p_2\cdots\cdots p_m$, $T = t_1t_2\cdots\cdots t_n$, where $p_i \in \Sigma$ ($i \in \{1, 2, \ldots, m\}$), $t_i \in \Sigma$ ($j \in \{1, 2, \ldots, n\}$) and $m \ll n$.

Output: The indices in T where a copy of P begins.

Here Σ is a finite non-empty set, which elements are not limited to the ordinary characters. For example, for the problem of finding occurrence(s) of a string in an English text, Σ is the set of English alphabet; for the problem of finding occurrence(s) of a nucleotide in a DNA sequence in the area of genetic engineering, Σ = {a, c, t, g}; for the problem of finding occurrence(s) of an amino acid in a protein sequence, Σ = {A, R, N, D, C, Q, E, G, H, I, L, K, M, F, P, S, T, W, Y, V}[7]; when Σ = {0, 1}, the problem is transformed to the binary matching, which is commonly used in the area of computer security.

III. RELATED WORK

Algorithms for string matching mainly include BF algorithm, KMP algorithm, BM algorithm. Many algorithms proposed by later researchers are based on them. So we discuss the three algorithms in this section.

A. BF Algorithm

BF algorithm is known as the Brute Force matching algorithm, whose basic idea is that to compare the characters of the pattern and the characters of the text from the left to the right; if the matching failed during the comparison, then the pattern moves forward (right) one character, and the algorithm starts a new iteration of comparison. This is a naive string matching method, which is easily to be understood, at the same time it has a low efficiency: as every character in text T generally needs to be compared for several times, there are a large number of repeated comparisons. The complexity of BF algorithm is O(mn).

B. KMP Algorithm

Every character in text T needs to be compared for several times in the BF algorithm. In order to overcome this drawback, Knuth, Morris and Pratt proposed a KMP algorithm [4] in 1977. The basic idea of the KMP algorithm is trying to use the historical information generated during the matching process, such that every character in text T is compared only once. Its specific implementation is as follows: at the beginning, it constructs a finite automaton according to the pattern P, whose states correspond to the prefixes of P; then, the characters in text T are read one by one, and the state of automaton changed accordingly; in the way, all matches could be found.

KMP algorithm is a very beautiful algorithm, because every character in text T is scanned only once and the total comparisons is at mostly 2n, twice the length of text T. KMP algorithm has a pre-processing procedure to generate the automata, whose complexity is $O(m^2)$. Generally $m \ll n$, so the consuming time on the pre-processing procedure can be ignored. However, it increases the complexity of the algorithm's implementation.

C. BM Algorithm

Before BM algorithm was proposed, the direction of character comparison was consistent to the moving direction of the pattern, i.e. both are from the left to the right. As a result, every character in text T needs to be compared and no character can be overleaped. In 1977, Boyer and Moore proposed the BM algorithm [2] during which the direction of character comparison is different from the moving direction of the pattern. The core idea of algorithm BM is as follows: when the pattern is positioned over the text at a place, it compares the characters of the pattern with the corresponding characters in the text one by one from the right of the pattern to the left side; if the matching is failed, then based on the comparison information of current iteration, it right shifts the pattern for one to several characters.

For example, suppose P = 101101 and the matching failed at the 4th character of P during the character comparison, so T has a substring "a01"($a \neq 1$) at the corresponding places; therefore, P should be right shifted 5 characters.

For the shifting distance has no relation with text T, the right shifting number of pattern P can be calculated in advance. According to the BM algorithm, two tables related to P needs to be made beforehand:

1) Right[t]: for $t \in \Sigma$, Right[t] is the rightmost index of P where character t appears, if there does not exist any t in pattern P, Right[t] is set to 0;
2) Jump[i]: Jump[i] is the right shift number of pattern P if it failed matching at P[i] during the right-to-left comparison ($1 \leq i \leq m$).

According the two tables, if algorithm BM fails matching in the i character of P, and the corresponding character in text T is t, then pattern P needs to be right shifted for $\max\{i - \text{Right}[t], \text{Jump}[i]\}$ characters.
The most important feature of BM algorithm is that the characters comparison is from the right of P to the left. Its advantage is that many characters in text T may be ignored during the comparison, in this way the efficiency of algorithm could be substantially improved. The two tables of Right and Jump for algorithm BM are only concerned with pattern P, and so they can be calculated in advance. The computation complexity of algorithm BM is $O(m)$. However, the table of Right has no use for binary string matching [1]. In most cases of the practical application, the efficiency of BM algorithm is higher than of KMP algorithm.

After BM algorithm was proposed, there were some algorithms to improve it. In 1980, Horspool simplified BM algorithm and proposed BMH algorithm [3]. Although it only used the information of the table Right, BMH algorithm acquired no bad efficiency. In 1990, Sunday proposed BMHS algorithm[6] that improved the BMH algorithm. In 2004, Qian qi and Hou Yibin made a combination of the previous algorithm's advantage, and enhanced it, proposed BMH2C[8] algorithm. The above algorithms both used the information of the table Right to improve BM.

IV. IMPROVEMENT ON THE BM ALGORITHM

According to the BM algorithm, the shifting distance of pattern P is completely determined by Table Jump and Table Right when the character comparison fails in matching. For pattern P='101101' in Table 1, if the mismatch appears at the right side of P, P should be right shifted for Jump[6] = 1 character, for the reason that Table Jump only utilizes the comparative information at current iteration. If we can utilize the history comparison information at previous iteration, pattern P may get a larger shifting distance.

For the pattern P='101101', suppose the mismatch appears at the right side of P at previous iteration, as Table 1 shows, so the character in P that corresponds a is not "1". Suppose the mismatch also appears at the right side of P at current iteration, so the character in P that correspond b is not "1" either. At this time, according to algorithm BM, P should right shift for 1 character. But, in fact, the results of the comparison of the two iterations show that none of the characters in P that correspond to a and b are "1", as Table 1 shows. For P does not consist continuous two characters that are not "1", P can be right moved out of the previous position, that is, it can right shift for 6 characters. So, if the mismatch appears at the right side of P at both previous iteration and current iteration, P can right shift for 6 characters.

Table 1. Improvement on the BM algorithm

T	*	*	*	*	*	*	*a*	*b*	*	*	*	*	*	*	*
Location for P at previous iteration		1	0	1	1	0	1								
Location for P at current iteration			1	0	1	1	0	1							
Shift location for P according to BM				1	0	1	1	0	1						
Improved shift location for P									1	0	1	1	0	1	

Above example shows that if we could utilize the history comparative information at previous iteration when the mismatch appears at current, iteration, the shift distance of pattern P could be increased, and then the efficiency of the comparison could be improved.

V. THE CBM ALGORITHM

According to the discussion in section 3, and referring the composite technology in Computational Mathematics, we propose a new string matching algorithm — Composite BM (CBM) algorithm in this paper. The key issue of the CBM algorithm is how to utilize the history comparison information achieved at previous iteration. So we construct a two-dimensional table Jump[*m*][*m*]. Jump[*i*][*j*] denotes the shift distance of pattern P, when the mismatch at previous iteration appears at p[*i*], and the mismatch at current iteration appears at p[*j*]. This table is only related to pattern P. Once Jump[*m*][*m*] is constructed, it can be utilized for searching P in different texts.

The comparison principle of algorithm CBM is shown in Figure 1. Suppose P is at place P0 at previous iteration, and the mismatch appears at index *i* of P0; and suppose P is at place P1 at current iteration, the mismatch appears at index *j* of P1; then P2, P’s new position, must meet following conditions: its substring at B matches with P1’s substring at B; its character at *b* does not match P1’s character at *j*; its substring at A matches P0’s substring at A; and its character at *a* does not matches with P1’s character at *i*. Above four matching conditions make a large shift distance Jump[*i*][*j*] for pattern P.

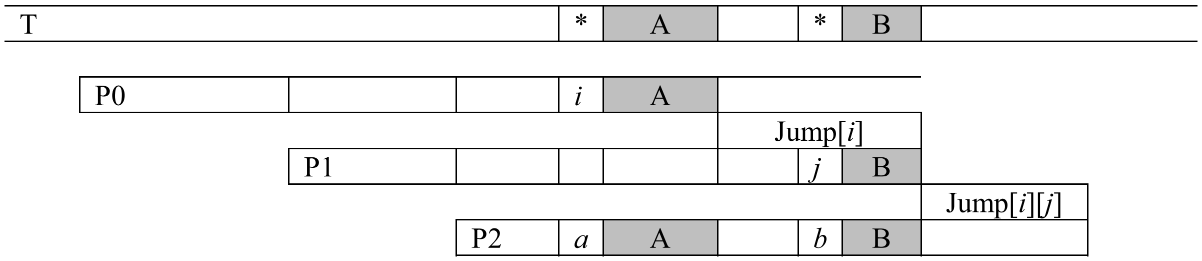

Figure 1. Working principle of CBM

The procedure of calculating Jump[*i*][*j*] is as follows:

```
int Jumps (int i, int j)
{
    bool IsMatch=true;
    CString p;
    int m=p.GetLength();

    if(SJump.empty())
        ComputerJumps();
    int jump=SJump[j];
    for ( ;jump<m; jump++)
    {
        IsMatch=true;
        for (int k=m-1; (k>j)&&(k>=jump); k--)
        {
            if (p[k]!=p[k-jump])
            {
                IsMatch=false;
                break;
            }
        }
        if(!IsMatch)        continue;
        if((j>=jump)&&(p[j]==p[j-jump]))    continue;
        IsMatch=true;
        int delta=jump+SJump[i];
        for (int k=m-1; (k>i)&&(k>=delta); k--)
        {
            if (p[k]!=p[k-delta])
            {
                IsMatch=false;
                break;
            }
        }
        if(!IsMatch)          continue;
        if((i>=delta)&&(p[i]==p[i-delta]))     continue;

        return jump;
    }
    return jump;
}
```

In the procedure, the initial values of Jump[*i*][*j*] is set to Jump[*j*] for every *i*. Then the values increased gradually by test, until it satisfies above four matching conditions.

In Table 2, if the mismatch appears at P[5] or P[6], the shift distance of Jump[*i*][*j*] is usually larger than that of Jump[*j*]. Let's see a simple probability analysis for this example, suppose text T is a random binary string, then the probability of mismatch at index 6 is 0.5; the probability of mismatch at index 5 is 0.25; So, the probability of having a larger shift distance is about 0.5 if we utilize Table Jump[*m*][*m*] instead of utilizing Table Jump[*m*].

Table 2. Jump[6][6] for For P = '101101'

i \ *j*	1	2	3	4	5	6
1	3	3	3	5	2	4
2	3	3	3	5	2	4
3	3	3	3	5	2	4
4	3	3	3	5	2	1
5	3	3	3	5	5	4
6	3	3	3	5	5	6

After generating table Jump[m][m], the specific matching process is similar to the BM algorithm.

In the case of small alphabet and long pattern, values in Jump[m][m] that is close to the right column are usually larger than the corresponding values in Jump[m], and the matching efficiency are improved. Binary searching in Computer Science and DNA sequence tests in genetic engineering are such kind of applications.

VI. EXPERIMENT AND CONCLUSION

The efficiency of BM algorithm depends on the size of the alphabet, especially for binary matching, BM does not do quite as well [1]. In view of this situation, we had experimented and compared BM and CBM.

In the experiment, we set text T to a random binary string of length 5000, set pattern P to random binary string of length 11 to 20, and counted the numbers of character comparisons of the matching. The results are shown in Figure 2.

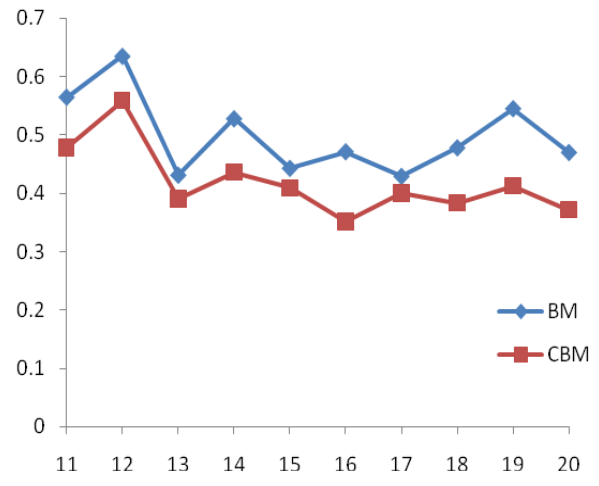

Figure 2. Comparisons of BM and CBM

In Figure 2, the vertical axis is ratios of the number of character comparisons and text's length. As can be seen from Figure 2, the numbers of character comparisons of CBM is about 84% of BM.

Generally, in the case of small alphabet and long pattern, values in Jump[m][m] that is close to the right column are usually larger than the corresponding values in Jump[m], and the matching efficiency are improved. Binary searching in Computer Science and DNA sequence tests in genetic engineering are such kind of applications.

ACKNOWLEDGEMENTS

This work was supported by National Natural Science Foundation of China (Grant No. 60773194) and NKBRPC (Grant No. G2004CB318000). Special thanks are to Dr. Kun He for many instructive suggestions.

REFERENCES

[1] Basse S., and A. V. Gelder, *Computer Algorithms,* Addison Wesley, 2000.

[2] R. S. Boyer, and J. S. Moore, “A Fast String Searching Algorithm”, Communications of the ACM, 1977, 20(10):762-772.

[3] R. N. Horspool, “Practical Fast Searching in Strings”, Software Practice and Experience, 1980, 10(6):501-506.

[4] D. E. Knuth, J. H. Morris, and V. R. Pratt, “Fast Pattern Matching in Strings”, SIAM Journal on Computing, 1977, 6(2):323-350.

[5] Y. Lu, S. Lu, and J. L. Ram, “Fast search in DNA sequence databases using punctuation and indexing”, Proceedings of the 2nd IASTED international conference on Advances in computer science and technology, p.351-356, January 23-25, 2006, Puerto Vallarta, Mexico.

[6] D. M. Sunday, “A very fast substring search algorithm”, Communications of the ACM, v.33 n.8, p.132-142, Aug. 1990.

[7] Renchao Jin, Enmin. Song, “A pre-processing algorithm for improving efficiency of the Boyer-Moore algorithm”, J. Huazhong Univ. of Sci. & Tech. (Nature Science Edition)(in Chinese), 2005, (33):265-267.

[8] Yi Qian and Yibin Hou, “A Fast String Matching Algorithm”, MINI-MICRO SYSTEMS ((in Chinese)), 2004, 25(3):410-413.