2019 IEEE 5th International Conference on Computer and Communications

Research of Pattern Matching Algorithm Based on KMP and BMHS2

Yansen Zhou
Department of Information Science and Technology
University of International Relations
Beijing, China
e-mail: zhouys@uir.edu.cn

Ruixuan Pang
Department of Information Science and Technology
University of International Relations
Beijing, China
e-mail: prx56789@163.com

***Abstract*—The improvement of the time performance of pattern matching algorithm mainly lies in reducing the number of character comparisons and increasing the distance of the matching window moving to the right when mismatch. In this paper, it adopts the second idea of improvement. The paper first analyzes KMP algorithm and its improved one, and then introduces BMHS2 algorithm. The distance of moving to the right of two improved algorithms is calculated when mismatch occurs respectively, and then proposes an improved algorithm based on the combination of improved KMP and BMHS2. The idea of this matching algorithm is that the overall matching is carried out from left to right, and every time the matching is carried out from left to right. When the text substring in matching window is mismatched with the pattern string, the larger jump distance of the I_KMP and BMHS2 is adopted to move the matching window to the right. Finally, two experiments to compare the time performance of the three algorithms above are carried out. The result shows that in the same experiment condition, compared to improved KMP and BMHS2, the time performance of improved pattern matching algorithm I_KMP_BMHS2 improved to some extent.**

Keywords-pattern matching, jumping distance, single module, time performance

I. INTRODUCTION

Pattern matching algorithm is widely used in search engine and text matching and has become a research hotspot. The room for performance improvement of a single pattern matching algorithm is limited. The current research focuses on combining multiple pattern matching algorithms and combining the advantages of each algorithm to further improve the time performance of pattern matching. KMP algorithm and BM algorithm [1] are two typical single-mode matching algorithms at present, with good time performance. The two algorithms have different matching sequences in each match, but they need to move the text string matching window to the right in case of mismatch. The improvement of performance of pattern matching algorithm [2] mainly lies in reducing the number of character comparisons and increasing the distance of the matching window moving to the right when matching. In this paper, the later one is adopted, which is increasing the distance of the matching window moving to the right when mismatch.

II. KMP ALGORITHM AND ITS IMPROVEMENT

A. KMP Algorithm

Compared to BF algorithm, KMP algorithm is an improved pattern matching algorithm. The algorithm was conceived in 1970 by Donald Knuth and Vaughan Pratt, and independently by James H. Morris, so it is called Knuth-Morris-Pratt operation. This algorithm [3] mainly eliminates the backtracking problem of the main string pointer in the matching process of BF algorithm, and makes use of the partial matching results to slide the pattern string to the right as far as possible and then continue to compare, so as to improve the efficiency of the algorithm to a certain extent. In the worst matching case, the time complexity of KMP algorithm is O(m+n), where the length of the pattern string is m and the length of the main string is n.

The array next[j] represents the position of the character in the pattern string that needs to be re-compared with the corresponding character in the main string when the character j in the pattern string fails to match the corresponding character i in the main string, and its value depends on the pattern itself, independent of the main string. Selecting the appropriate location of the pattern to participate in the starting point of the next match, that is, calculating next[j] is the key in KMP algorithm, and the function is expressed as follows:

```
void GetNext (SString T, int &next[ ])
      {/*calculate the value of next in pattern t and store it
in the next array*/
      i=1;  next[1]=0; j=0;
      while(i<=T[0] )
      {
        if(j= = 0||T[i]= =T[j])
          {
            ++i;
            ++j;
            next[i]=j;
          }
        else
        j=next[j];
       }
   }
```

Algorithm 1. calculate the value of next in KMP

The next array of pattern string is used to pattern matching function of KMP which locates pattern string P in main string T.

Assuming that there is a next array, KMP algorithm is as follows:

978-1-7281-4743-7/19/$31.00 ©2019 IEEE

Authorized licensed use limited to: Birla Inst of Technology and Science Pilani Dubai. Downloaded on February 25,2022 at 11:20:00 UTC from IEEE Xplore. Restrictions apply.

```
int StrIndex_KMP( SString s, SString t,int pos,int next[])
/*Start with the posth character of the string s to find the first substring equal to the string t*/
{
    int i=pos, j=1;
    while (i<=s[0] && j<=t[0] )
    {  /*no end is encountered*/
       if (j==0||s[i]==t[j])
       {
          i++;
          j++;
       }
       else
         j=next[j];
      /*there is a traceback of pattern string pointer*/
    if (j>t[0])  return  i-t[0];
  /*match successful and return to storage location*/
    else
     return  0;
}
```

Algorithm 2. pattern matching process of the KMP

B. Improved KMP Algorithm

1) Improvement in the move distance of main string matching pointer

Although the introduction of pattern matching KMP algorithm avoids the frequent backtracking of BF algorithm and improves the matching efficiency of pattern matching, the scanning part of KMP algorithm can be further improved after analysis. The basic idea of improving KMP algorithm [4] is introduced below.

The basic idea: whenever a match fails, the i pointer does not have to backtrack, but to use the "partial match" result which has been obtained to see if it is necessary to adjust the value of I, and then "slide" the pattern to the right several positions before continuing the comparison. When matching using KMP algorithm, if T[i]!=P[j], add " if(i+j-k≤m &&P[j] ! =T[i+j-k]) i =i +j -k ;" As a result, the value of the main string pointer i increases even when a mismatch happens, so that the matching performance is improved to some extent.

2) Improvement of Next Value of New Matching Location for Pattern String

Suppose i is matching pointer of pattern string and j is matching pointer of main string. If the next value of KMP is improved, the next value corresponding to each character in the pattern string is reduced, and the distance to the right of the matching window j-next[j] is increased, so the matching efficiency of KMP algorithm can be further improved. The next function defined by KMP is flawed in some cases [5]. For example, when the pattern string "aaaab" is matched with the main string "aaabaaaab", when i=4， j=4, P[4] ≠T[4]. As indicated by next[j], three comparisons are needed, such as i =4, j=3, i=4, j=2, i=4 and j=1. In fact, because the first, second, and third characters are the same as the fourth character in the pattern string, there is no need to compare them with the fourth character in the main string. Instead, the pattern string can be directly compared when i =5 and j=1 by moving the position of four characters to the right at one time. That is, if next[j]=k as the definition above says, and p[j] = p[k] in the pattern, then when the character T[i] in the main string is not equal to the pattern string p[j], there is no need to compare it with p[k], but directly compare with p[next[k]].In other words, next[j] should be the same as next[k] at this time. Thus, the algorithm for calculating the nextVal which is next amendment is shown in algorithm 3.

TABLE I. COMPARISON OF THE VALUE OF NEXT AND NEXTVAL

j	1	2	3	4	5
Pattern string	a	a	a	a	b
next[j]	0	1	2	3	4
nextval[j]	0	0	0	0	4

```
void get_nextval(SString T,int &nextval[ ])
{
/*Calculate the corrected value of the next array of the pattern string T and store it in array nextval*/
    i=1; nextval[ 1 ]=0; j=0;
    while ( i<T[ 0 ] ) {
      if ( j==0 || T[i]==T[j] )
        {
          i++; j++;
          nextval[i]=k;
          if  (T[i]!=T[j])
               nextval[i]=j;
          else
             nextval[i]=nextval[j];
          }
       else
          j=nextval[j];
    }
}//get_nextval
```

Algorithm 3. evaluation algorithm of nextVal

3) Improved algorithm of I_KMP

Aiming at the two improvements of KMP algorithm, which uses the modified nextVal value and the increased value of main string pointer i when a mismatch happens, an improved I_KMP algorithm is designed as follows:

```
int StrIndex_IKMP( SString s, SString t,int pos, int nextVal[])
/* Start with the character pos of the string s to find the first substring equal to the string t */
{  int i=pos, j=1;
    while (i<=s[0] && j<=t[0] ) /*no end is encountered*/
     if (j==0||s[i]==t[j])
     {
        i++; j++;
     }
     else
      {
        k=j-nextVal[j];
        if(i +j -k ≤m &&P[ j ] ! =T[ i +j -k] )
         i=i+j-k;
        //the matching position of text string is moved
```


Authorized licensed use limited to: Birla Inst of Technology and Science Pilani Dubai. Downloaded on February 25,2022 at 11:20:00 UTC from IEEE Xplore. Restrictions apply.

```
            j=nextVal[j];
      /* there is a traceback of pattern string pointer */
      }
   if (j>t[0])    return  i-t[0];
    /* match successful and return to storage location */
    else
    return  0;
    }
```

Algorithm 4. the matching process of I_KMP algorithm

III. BMH AND BHMS2 PATTERN MATCHING ALGORITHM

A. BMH Algorithm

Aiming to avoid the complex calculation of the right shift amount of the pattern string in the BM algorithm, BMH algorithm [6] use the last character in the matching window as the distance of the pattern string to the right when there is a mismatch in the matching process. The idea of this algorithm is that the right shift amount of the pattern string is not related to the mismatch character, and the right shift amount is calculated without considering the character in the text that causes the mismatch, but simply using the character aligned with the right end of the pattern in the text to determine distance of the right shift.

Matching process: if the pattern string mismatches a character in the string and the character aligned to the rightmost end of the match window is not in the pattern string, the pattern string moves pLen positions to the right (assuming the length of the pattern string is pLen). If the character is in the pattern string, pattern string moves to the right for a certain distance according to the calculated skip array of pattern string.

The algorithm to calculate the function of skip array is as follows:

```
int *GetSkip(char*p, int  pLen)
  {
     int i, *skip=(int*)malloc(|∑|× sizeof(int));
    for(i=0;i<|∑|;i++)
       skip[i]=pLen;
    for(i=0;i<=pLen-2;i++)
     skip[p[i]]=(pLen-1)-i;
    return skip;
}
```

Algorithm 5. The calculation algorithm of skip in BMH algorithm

In the algorithm, |∑| is the number of character sets in the string. for example, in a string composed of ASCII codes, |∑| is 256.

Algorithm analysis: theoretically, the complexity of BHM algorithm in the worst case is O(mn), but in general, it has better performance than BM. In practice, the time complexity of the algorithm is O(m+n).

B. BMHS2 Algorithm

The algorithm [7] is to calculate distance of the right shift without considering which character in the string causes the mismatch. Instead the distance of right shift depends on the rightmost character of the match window and its next character. That is to use character t[i] and t[i+1] as a substring to determine the amount of shift.

Assuming that the character is coded with ASCII and the space range is [0, 255], which is the range represented by a byte. So for a substring whose length is 2, there are 256×256 possibilities, a two-dimensional array can be used to represent the shift of each substring [8].

Here is an algorithm to calculate the jump distance in a two-dimensional array:

```
for(i=0;i<256;i++)
  for(j=0;j<256;j++)
  skip[i][j]=pLen+1  //pLen is the length of pattern string
for(i=0;i<plen-1;i++)
 skip[p[i]][p[i+1]]=plen-i-1;
```

/*Considering that t[i] and t[i+1] is not in the pattern string when mismatch happens, and the t[i+1] of main string may be the same as the first character of the pattern string, the right shift amount needs to be corrected under this special condition.*/

```
for(i=0;i<256;i++)
   if(skip[p[0]][i]= =pLen+1)
      skip[p[0]][i]=pLen;
```

BMHS2 single-mode matching algorithm is as flow:

```
 int BMHS2Search(char *t, int tlen, char *p, int plen, int
skip[256][256])
  {
   int j, i=0;
   while(i<=tlen-plen)
   {
      for(j=plen-1;j>=0&&p[j]= =t[j+i];j--) ;
      if(j<0)
      return i;
     else
       i+=skip[t[i+plen-1]][t[i+plen]];}
     return  -1;
  }
```

Algorithm 7. matching process of BMHS2

The BMHS2 algorithm combines the advantages of BMH and BMHS algorithms [9]. The distance of average shift calculated by two characters is larger than that calculated by one character, because for the basic ASCII character set, the probability of each character appearing at a certain position in the pattern string is 1/256; for a substring of which length is 2, the probability of appearing in a position of the pattern string is drastically reduced to 1/65536. It is obvious that, the probability of that substring of length 2 occurs near right of the pattern string is lower when mismatch happens. Most substrings of length 2 are hard to appear in the pattern string, so the best value to jump right is pLen.

IV. AN IMPROVED ALGORITHM COMBINED WITH I_KMP AND BMHS2

A. The Idea of the Improved Algorithm

In KMP single-mode matching algorithm, pattern string and text string are matched from left to right. If there is a

Authorized licensed use limited to: Birla Inst of Technology and Science Pilani Dubai. Downloaded on February 25,2022 at 11:20:00 UTC from IEEE Xplore. Restrictions apply.

mismatch, the matching pointer of text string remains unchanged [10] , and the pattern string is re-matched by the character corresponding to the nextVal[j] value and the character of the current mismatch position of the text string. The BMHS2 algorithm generally matches from left to right, but every time it matches from the rightmost part of the pattern string to the left part with the text string. If there is a mismatch, it does not consider where the mismatch is, but to use the right-most character and the next character of the current matching window to calculate the right-moving distance by which the matching window moves to the right. Therefore, based on the characteristics of these two algorithms and their respective advantages, I_KMP and BMHS2 are adopted to move to the right with larger moving distance during the mismatch.

The match process of the improved algorithm is as follows:

The pattern string matches the text string from left to right, but the matching window is matched from left to right each time. When there is a mismatch, the values of shift=skip[i-j+pLen][i-j+pLen+1] and j are calculated. If shift is less than j, the improved I_KMP algorithm is adopted for the next round of matching. Otherwise, i=i-j+shift, j=1, the BMHS2 algorithm is used to move matching window to match pointer i of text string. After that, next round of I_KMP matching is conducted, where i is the position of the first character in the current matching window of text string, and j is the current matching position of the pattern string. The above matching idea can accelerate the moving speed of pattern string to the right and further improve the efficiency of pattern matching.

Therefore, the improved algorithm combined with I_KMP and BMHS2 can ensure the maximum distance to the right each time. Compared with I_KMP and BMHS2, the performance of the improved algorithm is improved to a certain extent.

B. The Implementation of the Improved Algorithm

```
    int StrIndex_I_KMP_BMHS2( SString s, SString t , int skip[][], int nextVal[])
/* Start with the posth character of the string s to find the first substring equal to the string t */
    {  int i=1, j=1;
       while (i<=s[0] && j<=t[0] ) /*no end is encountered*/
        {
         if (j==0||s[i]==t[j])
         {
            i++; j++;
         }
        else
         {
            k=j-nextVal[j];
            shift= skip[i-j+pLen][i-j+pLen+1]);
            d= skip[i-j+pLen][i-j+pLen+1])-j;
            if(d<0)  //I_KMP algorithm is used for matching
            {
               if(i +j -k ≤m &&P[j] ! =T[ i +j -k] )
               i=i+j-k;
               //the matching position of text string is moved
                  j=nextVal[j];
              /*there is a traceback of pattern string pointer*/
          }
          else
   //BMHS2 is used to move the text string matching pointer i
          {
           i=i+d;
           j=1;
          }
       }
    }
     if (j>t[0])  return  i-t[0];
     /* match successful and return to storage location */
      else
       return  0;
     }
```

Algorithm 8. Matching process of I_KMP_BMHS2

Algorithm implementation process: the overall matching idea of the improved algorithm: The improved KMP algorithm is adopted for each match, and every match is matched from left to right in the matching window. In order to improve the distance to the right during each mismatch, the improved KMP, the larger distance move to the right of the improved KMP algorithm and BMHS2 algorithm are adopted, so as to further improve the matching efficiency under the same environment.

V. EXPERIMENT AND ANALYSIS

In order to verify the time performance of the improved matching algorithm I_KMP_BMHS2, we compare it with I_KMP and BMSH2 matching algorithms. The comparisons are conducted respectively when the number of pattern strings increased and the length increased. The pattern string is derived from 1000 rules in Snort intrusion detection system. The target text string to be texted is 20M bytes of professional English news downloaded from an Internet website. The pattern string collection consists of names, addresses and computer jargon commonly used in English news.

Experimental test environment: in order to ensure the accuracy and the validity of text experimental test data, BMHS2 and I_KMP and their improved combination are all implemented by Visual Studio 2018 integrated development tool and C++ language. The configuration of the test host is Intel® Core™ i7-7500U CPU，2.90GHz main frequency, 8G memory, Windows 10, and clock() which is a time function accurate to microseconds is used.

A. Test for the Effect of Pattern String Number on Algorithm Performance

Experimental data: keep the average length of pattern string to 20 characters and the length of text string to be matched to 10M bytes.

Test scheme: The time performance of I_KMP, BMHS2 and I_KMP_BMSH2 algorithms is tested by taking 100, 200, 300, 400, 500, 600, 700 and 800 pattern strings as selected.

Authorized licensed use limited to: Birla Inst of Technology and Science Pilani Dubai. Downloaded on February 25,2022 at 11:20:00 UTC from IEEE Xplore. Restrictions apply.

The time performance is evaluated by the time required to complete the matching. The total time obtained by the above test is divided by the number of pattern strings and the results are averaged.

TABLE II. MATCHING TIME WHEN THE NUMBER OF PATTERN STRINGS VARIES (MS)

Algorithm	Number of Pattern Strings							
	100	200	300	400	500	600	700	800
I_KMP	316	303	301	302	304	299	307	302
BMHS2	302	291	289	289	296	289	299	296
I_KMP_BMHS2	248	246	242	250	243	246	251	248

The data in TABLE II show that the processing time of the improved pattern matching algorithm I_KMP_BMHS2, I_KMP and BMSH2 will keep relatively stable with the change of the number of pattern strings. When the same pattern string matches the same text string, the average performance of the improved pattern matching algorithm can be improved to about 15% compared with the BMSH2 algorithm, and 20% compared with the I_KMP algorithm.

B. Test for the Effect of Pattern String Length on Algorithm Performance

Experimental data: keep the number of pattern strings to 100 and the length of text string to be matched to 2M bytes.

Test scheme: The time performance of I_KMP, BMHS2 and I_KMP_BMHS2 algorithms is tested by taking pattern strings with lengths of 30, 40, 50, 60, 70, 80, 90 and 100. The working efficiency is evaluated by the time required to complete the matching. The above tests are repeated for 10 times each and the results are averaged.

TABLE III. MATCHING TIME WHEN PATTERN STRING LENGTH VARIES (MS)

Algorithm	Length of Pattern String							
	30	40	50	60	70	80	90	100
I_KMP	281	253	232	223	201	176	150	145
BMHS2	268	242	221	212	189	163	139	132
I_KMP_BMHS2	236	215	206	172	149	132	112	104

The data in Table III indicates that the processing time of the improved pattern matching algorithm I_KMP_BMHS2, I_KMP and BMSH2 will be reduced with the increase of pattern string length if the number of pattern strings remains unchanged. When matching the same text string, the average performance of the improved pattern matching algorithm can be improved to about 16% compared with the BMSH2 algorithm, and 20% compared with the I_KMP algorithm.

VI. CONCLUSION

The performance of the improved KMP algorithm combined with BMHS2 algorithm can be improved, mainly lying in the matching window of text string is moved to the right by using the larger jump distance between KMP and BMHS2 in each mismatch, which can improve the matching time performance to a certain extent. Considering the special requirements of KMP matching algorithm, the improved pattern matching algorithm I_KMP_BMHS2 is required to move from left to right in each match.

Compared with the multi-pattern matching algorithm, An improved algorithm combining KMP with BMHS2 algorithm still has the problem of low efficiency. How to apply the improved single-mode matching algorithm to the existing multi-mode matching algorithm and further improve the performance of the multi-mode matching algorithm is the problem to be solved in the next step.

REFERENCES

[1] Xiao Zhao. An efficient pattern string matching algorithm [J]. Journal of Shanxi University of Science and Technology, 2017, 35(1):183-187.

[2] Huiming He etc. A multiple pattern string matching algorithm based on substring recognition [J]. Computer Applications and Software, 2011, 28(11):10-14.

[3] Wenzhi Li etc. A string matching technique for large-scale collection of short feature [J]. Computer Engineering and Applications,2014,50(1): 105-110.

[4] Shibo Dong etc. An improved string pattern qppmatching algorithm [J]. Computer Engineering and Applications, 2013,49(8):133-137.

[5] Shengdong Dai etc. A matching algorithm for computing pattern features of DNA sequence [J]. Journal of Electronic University of Science &Technology of Hangzhou,2015,35(1):88-92.

[6] Gaochao Sun etc. Method for improving pattern matching performance by using parallel computing[J]. Journal of Information Engineering University, 2011, 12(6):750-753.

[7] Xiao Zhao etc. An efficient pattern string matching algorithm [J]. Journal of Shanxi University of Science & Technology, 2017,35(1):183-187.

[8] YI Zhuliang. On improved BM pattern matching algorithm based on network intrusion detection system[J]. Computer Application and Software, 2012, 29(11) :193- 196.

[9] WANG H, ZHANG L. Quick pattern matching algorithm based on bad character sequence checking[J]. Computer Applications and Software, 2012, 29(5) :114-116.

[10] YAN H. Research on improved BMH single-pattern matching algorithm based on Snort[J]. Computer Engineering and Applications, 2012, 48(31):78-81.

Authorized licensed use limited to: Birla Inst of Technology and Science Pilani Dubai. Downloaded on February 25,2022 at 11:20:00 UTC from IEEE Xplore. Restrictions apply.