一种基于编码关联的快速多模式匹配算法

朱永强 秦志光

(电子科技大学计算机科学与工程学院 成都 611731)

摘 要 多模式匹配算法经常使用有限自动状态机来实现多个模式串的并行匹配。针对基于自动状态机的多模式匹配算法在应用于中文编码时存在的存储空间膨胀问题,使用中文字符的拆分编码构造自动状态机,以优化算法自动状态机的存储空间,并利用中文编码的编码关联性,设计了一种基于编码关联跳转的失效跳转表,使用启发式跳跃规则提升匹配算法的时间性能。最后通过实验证明,中文编码环境下,相比于其它使用自动状态机的多模式匹配算法,改良算法拥有更小的空间消耗与更快的运行速度。

关键词 多模式匹配,DFSA算法,WM算法,DFSA-QS算法,编码关联

中图法分类号 TP301.6 **文献标识码** A **DOI** 10.11896/j.issn.1002-137X.2016.2.005

Multi-pattern Matching Algorithm Based on Coding Association

ZHU Yong-qiang QIN Zhi-guang

(School of Computer Science & Engineering, University of Electronic Science and Technology of China, Chengdu 611731, China)

Abstract Multi-pattern matching algorithms often use finite state automaton to implement parallel matching of multiple pattern strings. When multi-pattern matching algorithm based on finite state automaton is applied into the Chinese, it will lead to storage space expansion. Aiming to solve this problem, this improved algorithm constructs automatic state machine by using split coding of Chinese characters to save storage space and designs failure jump table based on coding association, and uses heuristic jumping rules to improve time performance of matching. Finally, compared to other algorithm, smaller space consumption and faster speed in Chinese environment of this improved algorithm were proved by simulation.

Keywords Multi-pattern matching, DFSA algorithm, WM algorithm, DFSA-QS algorithm, Coding association

1 引言

多模式匹配广泛地应用在恶意代码匹配、DNA序列检索、文本关键字检索等领域,其整体性能受算法设计思路、编码环境与字符环境等多种因素的影响,一般不存在一种在任何匹配环境下都达到性能最优的多模式匹配算法。

经典的多模式匹配算法包括DFSA算法[1,2]、WM算法[3-6]与上海交通大学的王永成教授提出的DFSA-QS算法[7]。相对于简单的单模式匹配算法,多模式匹配算法往往利用一些辅助结构,如自动状态机、索引表等,来加快匹配进行的速度。

由于模式匹配算法的性能与其所应用的环境关联紧密,因此相对于国外学者设计的算法,王永成教授提出的DFSA-QS算法在中文语义环境下具有更优越的性能。但DFSA-QS算法使用了自动状态机,在中文编码环境下存在着严重的空间膨胀问题,对编码进行拆分的处理虽可解决空间膨胀问题,但会同时降低算法的运行效率。

本文在对比分析了现有经典多模式匹配算法的基础上,结合中文编码环境以及DFSA-QS算法本身的特点,设计并实现了一种基于编码关联的多模式匹配算法:IDFSA-QS算法,以实现中文环境下对算法匹配速度与空间性能的提升。

2 中文环境下的多模式匹配算法分析

中文语义字符集合较大,且单词所含的字符元素个数一般远少于英文单词所含元素个数,可以将中文匹配环境概括为:大字符集、短模式串。以下针对中文环境下模式串与匹配串的特点,对3种经典多模式匹配算法:DFSA算法、WM算法以及DFSA-QS算法进行对比分析。

定义1(最优匹配环境) 对于DFSA-QS算法,最优匹配环境是指每一次比较的第一个字符即发生失配,且失配字符的下一个字符不在任何模式串中的情况;对于WM算法,则是指WM算法窗口内的后缀HASH值每次都取到最大跳转值 m 的情况;对于DFSA算法,由于失效跳转表的设计方式,其时间效率恒为 $O(n)$。

由于中文语义的大字符集特性,匹配中大部分的比较运算都为失配,因此实际中文检索环境接近于理论的最优匹配环境,分析中可以使用最优匹配环境来逼近真实的匹配算法工作环境。

到稿日期:2015-02-25 返修日期:2015-06-06

朱永强(1987-),男,硕士生,主要研究方向为信息安全,E-mail:demonic_will@foxmail.com;秦志光(1956-),男,教授,博士生导师,主要研究方向为信息安全、网络安全。

在匹配的编码环境确定的情况下，设文本串的长度为 n，模式串集合中的最短模式串长度为 m。上述 3 种算法中，由于 DFSA 算法对文本串只能进行逐个字符的检索，其时间效率恒为 $O(n)$；在最优环境下，WM 算法的时间效率理论上可以达到 $O(n/m)$；而 DFSA-QS 算法的时间效率则可以达到 $O[n/(m+1)]$。

DFSA 算法由于不能利用坏字符思想进行跳转，无法充分利用大字符集的特点，因此不适用于中文环境。WM 算法考虑其主要的跳转表 SHIFT [h]，由于中文环境下的最短模式串长度较小，WM 算法中的滑动窗口取值只能取到很小的范围，即最短模式串长度 m 小于 DFSA-QS 算法的最大跳转距离 $m+1$[8,9]，同时由于中文模式串的短模式串与大字符集特性发生部分匹配的几率较低，因此算法的 PREFIX 表作用被弱化。综上，WM 算法的设计理念并不适用于中文编码环境。

DFSA-QS 算法充分使用了坏字符思想，当当前匹配字符不存在于模式串组中任一模式串时，可以直接安全跳过 $m+1$ 个字符，即最大跳转距离。由于大字符集环境下，失配的情况出现较多，DFSA-QS 算法取得其最大跳转值的几率相对较高，实际运行效率最接近于理论上的最优匹配环境，因此中文环境下，算法的平均时间性能要高于 WM 算法与 DFSA 算法[10-13]。参考文献[10]通过对实际中文文本集的测试，验证了 DFSA-QS 算法在中文环境下的高效性。但是 DFSA-QS 算法直接应用于中文环境时，会产生严重的空间膨胀，造成算法的实用性降低，而编码拆分的方法又将造成算法的时间性能下降，导致存储空间与时间性能无法同时优化解决的问题。

3 改良算法的设计与实现

针对 DFSA-QS 算法存储空间与时间性能无法同时优化解决的问题，本文设计并实现了一种改良算法：IDFSA-QS 算法，该算法使用拆分编码构造自动状态机，以解决存储空间问题；并使用编码关联性构造失效跳转表，以提升算法的时间效率，补偿编码拆分造成的时间性能损失。

3.1 改良算法自动状态机的构建

IDFSA-QS 算法使用自动状态机进行匹配字符的状态跳转。直接在中文环境下使用自动状态机进行匹配时，状态机中标示下一个状态的存储体需存放 256 * 256 个地址，设每个地址使用 32 位指针存储，则每个节点需占用：

$$4\times256\times256=261244\text{byte} \quad (1)$$

即大约占 0.25MB 的空间。设模式串的个数为 100，模式串的平均长度为 5，则自动状态机的节点所占用的空间约为：

$$Space=100\times5\times0.25\text{MB}=125\text{MB} \quad (2)$$

由于消耗空间过大，此时直接构建自动状态机将失去工程可行性。

为解决上述问题，IDFSA-QS 算法将每个中文码值拆分为两个单字节，得到单元码值范围缩小但子模式串长度加倍的拆分模式串组。使用此拆分模式串组生成自动状态机，此时状态机中节点的大小将缩小至原来的1/256，由于匹配串长度增长为原有的 2 倍，因此存储空间实际缩小为原来的 1/128，可以有效解决存储空间膨胀问题。

IDFSA-QS 算法使用拆分后的拆分模式串组，倒序生成自动状态机的关键代码如下：

代码 1 自动状态机构建关键代码(输入模式串组中一个模式串时)：

```
TreeNode * ptr=Tree_First; //指针指向自动机入口
int length=wcslen(substr) * sizeof(wchar_t);//获取拆码串长度
strncpy(substr_get,(char * )substr,substrlen);//拆码转换
for(i=length-1;i >= 0;i--) //反向输入自动状态机
{  index_code=substr_get [i]
   if(ptr->next[index_code] == NULL)
       {index_tree ++;
       ptr->next[index_code]=& Tree_First[index_tree];
       ptr=ptr->next[index_code];
       }//添加新的状态进入自动状态机
   else
       ptr=ptr->next[index_code];//已存在状态,进行切换
}
```

IDFSA-QS 算法执行时，读入匹配串当前字符，若自动状态机的下一个状态(或第一层入口)包含此字符，则自动状态机跳转至与此字符对应的状态；如不包含，则考察匹配串中当前位置加上最小模式串长度处字符，读取此字符对应跳转值以确定下一次匹配点。

3.2 算法失效跳转表的构建

首先定义中文编码方式的一个特点。

定义 2(编码关联) 中文字符编码一般需使用两个单字节，将中文码值作为两个单字节组合进行匹配时，若匹配串中一个单字节不存在于任一模式串中，即可判断与其属于同一中文字符的另一个字节在模式串中必不存在，此特点定义为编码关联。

IDFSA-QS 算法利用中文编码拆分后的编码关联性生成失效跳转表(DFSA-QS 算法中的 skip 函数)。设模式串组在未拆分前的最小串长度为 $minLen$，则拆分后此值变为 $2minLen$，按 DFSA-QS 算法的生成规则，其 skip 函数每个字符的初始跳转值为 $2minLen+1$，也即最大跳转值。

IDFSA-QS 算法利用了编码关联性后，若失效后的考察字符不在任何模式串中，则可以将此字符与此字符的编码关联字符一起跳过，相比于原 DFSA-QS 算法，失配后可以多跳跃一个字符。因此 IDFSA-QS 算法的 FailedTable 表的初始值可提升为 $2minLen+2$，从而可有效提升算法的时间性能。

具体的 FailedTable 表生成算法如下：

1)将组中各个模式串的中文字符拆分为两个单字节字符，生成拆码模式串，构造长度为当前编码范围(0～255)大小的失效跳转表 FailedTable，跳转表序号为各个单字节的码值，对应的值为此字符的跳转距离。

2)使用拆码模式串中每个原属于中文整体编码的低半位单字节的对应跳转值，来替换 FailedTable 表中序号与此字节码值相等的位置的默认值。跳转值计算方法如下：

设 $JumpLen$ 为当前码值对应的跳转值，$PatterLen$ 为当前拆码模式串的长度，i 为当前码值位置到当前拆码模式串最末尾字节的字节距，则拆码模式串中每个码值对应的跳转值计算方程为：

$$JumpLen=PatterLen-i$$

如同一序号位置可以得到多个跳转值，则选择最小跳转值替换 FailedTable 表的对应项。

生成 FailedTable 的关键代码如下：

代码 2 失效跳转表构建关键代码(输入模式串组中一个模式串、big endian 存储方式):

```
substrlen=wcslen(word) * sizeof(wchar_t);//获取拆码串长度
strncpy(substr_get,(char * ) word,substrlen);//拆码转换
for(i=0;i<substrlen;i+=2)//只使用低位字符生成跳转表
{  jump=substrlen-i;//计算对应的跳转值
   index=substr_get[i]
   if (jump_tab [index]>jump)
     jump_tab [index]=jump;//取同字符最小跳转值
}
```

当字符发生失配时,算法考察失配字符下一个中文整体编码的低半位字符,获取对应的跳转值,按照其跳转值进行跳转。

此处用于生成失效跳转表的字节应根据编译环境对字节的存储方式(big endian 方式或 little endian 方式)选择两个字节中取值范围更大的字节,以更好地发挥坏字符的优势。如:编码使用 Unicode 方式,则选择整体编码的低字节,这是因为 Unicode 汉字编码的低字节有着更大的取值范围(高位字节范围为 0x4E 至 0x9F,低位字节范围为 0x00 至 0xFF),可以使实际匹配过程更接近最优匹配环境。

3.3 算法的匹配流程分析

以下通过实例分析改良算法的匹配流程,将 DFSA-QS 算法同样应用于编码拆分环境作为对比,设匹配串为:"梦里不知身是客,一晌贪欢",模式串组为:成都;重庆。中文字符使用通用的 Unicode 编码、16 进制、big endian 方式。以下分析 DFSA-QS 算法以及 IDFSA-QS 算法的实际匹配流程。

3.3.1 DFSA-QS 算法的匹配流程

拆码环境中,DFSA-QS 算法的匹配流程为:

(1)状态机第一层入口为 90 与 5e。

(2)对应的失效跳转表如表 1 所列。

表 1 DFSA-QS 算法的失效跳转表

码值	10	5e	62	86	90	91	cd	fd	其它码值
跳转	4	1	3	2	1	3	4	2	2minLen+1=5

(3)此时前 3 次匹配过程如表 2 所列。

表 2 DFSA-QS 算法的匹配过程

第一次比较位置	↓
	a6 68 cc 91 0d 4e e5 77 ab 8e 2f 66
	a2 5b 0c ff 00 4e cd 54 2a 8d 22 6b
第二次比较位置	↓
	a6 68 cc 91 0d 4e e5 77 ab 8e 2f 66
	a2 5b 0c ff 00 4e cd 54 2a 8d 22 6b
第三次比较位置	a6 68 cc 91 0d 4e e5 77 ab 8e 2f 66
	↓
	a2 5b 0c ff 00 4e cd 54 2a 8d 22 6b

箭头表示当前对比位置,第一次进入状态机的索引值为 91,因为 91 不匹配第一层入口字符,所以考察 91 之后的下一个字符 0d,读取 0d 的跳转值为 5,则模式串从 ab 继续开始匹配。DFSA-QS 算法通过一次比较,在拆码匹配串中的跳转距离为 9。

类似第二次比较,通过第三次匹配运算,模式串组可跳转至 5b 处,跳转距离为 14。

显然,第二次比较中,ab 是"是"字编码的第二个字符,而状态机的第一层则为各个模式串的编码的第一个字符,此次比较为冗余的错位匹配,将影响算法的时间性能。

3.3.2 IDFSA-QS 算法的匹配流程

IDFSA-QS 算法利用中文编码的关联性,只使用每个字符中的低半位字节生成失效跳转表,在匹配失效后,直接考察当前窗口的右侧下一个字符,如此字节在任一模式串中都不出现,则可直接将此字节与它右侧的字节一起跳过。此时每次跳转的最大距离将变为 $2N+2$,算法的匹配过程为:

(1)IDFSA-QS 算法状态机第一层入口为:90 与 5e。

(2)对应的失效跳转表如表 3 所列。

表 3 所设环境下 IDFSA-QS 算法的失效跳转表

码值	10	86	cd	fd	其它码值
跳转	4	2	4	2	2minLen +2= 6

(3)此时前 3 次匹配过程如表 4 所列。

表 4 IDFSA-QS 算法的匹配过程

第一次比较位置	↓
	a6 68 cc 91 0d 4e e5 77 ab 8e 2f 66
	a2 5b 0c ff 00 4e cd 54 2a 8d 22 6b
第二次比较位置	↓
	a6 68 cc 91 0d 4e e5 77 ab 8e 2f 66
	a2 5b 0c ff 00 4e cd 54 2a 8d 22 6b
第三次比较位置	a6 68 cc 91 0d 4e e5 77 ab 8e 2f 66
	↓
	a2 5b 0c ff 00 4e cd 54 2a 8d 22 6b

第一次比较时,91 不在状态机的第一层入口,则考察 91 之后的下一个字符 0d,通过失效跳转表获取 0d 的跳转值为 6,则从当前比较位置向后跳 6 个字节,即字符 8e 处继续开始匹配,在拆码匹配串中的跳转距离为 10。

类似第二次比较,通过第三次匹配运算,模式串组可跳转至 5b 处,跳转距离为 16,大于 DFSA-QS 算法 14 的跳转距离。

可见,在相同的比较次数下,IDFSA-QS 算法比 DFSA-QS 算法跳过了更多的字符,且避免了模式串与匹配串的错位匹配,拥有更好的时间性能。

此环境下,IDFSA-QS 算法相对于 DFSA-QS 算法的速度提升率为:

$$\begin{aligned} RateAdd &= (1/Time)/(1/Time_2)-1 \\ &= [t\times 2M/(2N+1)]/[t\times M/(N+1)]-1 \\ &= 1/(2N+1) \end{aligned} \tag{3}$$

显然,$RateAdd$ 为 N 的递减函数。理论上,最短模式长度越小,改良算法获得的性能提升越大。举例来说,设最短模式长度为 2 时,$RateAdd=0.2$,即相较于同环境下的 DFSA-QS 算法,其时间性能提升率为 20%。

4 改良算法的性能分析

以下分析改良算法的时间性能与空间性能。由于中文环境下 DFSA 算法与 WM 算法的效率都相对较低,因此此处略去这两种算法的分析。分析中,C_i 代表 i 号算法的空间消耗,$Time_i$ 代表 i 号算法的时间消耗,下标 i 的标示含义如下:0 号代表本文所提 IDFSA-QS 算法,1 号代表运行于中文编码空间的 DFSA-QS 算法,2 号代表运行于拆码空间的 DFSA-QS 算法,3 号代表文献[14]中提出的一种对 DFSA-QS 算法的改良算法即 SDFSA-QS 算法。

4.1 改良算法的空间性能分析

设中文模式串个数为 n,平均长度为 m,自动状态机无交

叉结点，则 DFSA-QS 算法的空间消耗 C_1 为：

$$C_1=65536\times(m\times n+1) \tag{4}$$

IDFSA-QS 算法运行于编码拆分环境下，其空间消耗 C_0 为：

$$C_0=256\times(2\times m\times n+1) \tag{5}$$

因此 DFSA-QS 算法与 IDFSA-QS 算法的空间消耗比为：

$$C_1/C=128+128/(2\times m\times n+1)>128 \tag{6}$$

可见，IDFSA-QS 算法的空间消耗远小于 DFSA-QS 算法。

若将编码拆分理解为一种匹配环境的映射变换（由双字节环境拆分映射为单字节环境），则编码拆分的思想也可应用于 DFSA-QS 算法。将编码拆分的方式用于 DFSA-QS 后，DFSA-QS 算法的空间消耗 C_2 与改良算法的空间消耗 C 相同，为：

$$C_2=C_0=256\times(2\times m\times n+1) \tag{7}$$

SDFSA-QS 算法同样用于解决 DFSA-QS 算法在中文环境下的存储空间膨胀问题，它也使用了编码拆分的方法解决空间问题。然而对于失效跳转表的处理，SDFSA-QS 算法采用了自动状态机编码拆分而跳转表不拆分的分治方式，即失效跳转表的基本字符单位为占两个字节的中文编码字符。因此 SDFSA-QS 算法的空间消耗 C_3 为：

$$C_3=256\times 2\times m\times n+65536 \tag{8}$$

可知，SDFSA-QS 算法与 IDFSA-QS 算法的空间消耗比为：

$$C_3/C=1+255/(2\times m\times n+1) \tag{9}$$

由于 m、n 值一般不会很高，因此 SDFSA-QS 算法的空间消耗相对于 IDFSA-QS 算法也较大，一般是 IDFSA-QS 算法的 5～10 倍。

综上，可知空间消耗由大向小排序为：

$$C_1>C_3>C_2=C_0 \tag{10}$$

4.2 改良算法的时间性能分析

在中文字符匹配、最优匹配环境下，设原模式串组中最短长度为 N，匹配串长度为 M，每次比较操作耗时为 t，编码拆分后，最短模式长度变为 $2N$，匹配串长度为 $2M$，则：

非编码拆分环境下，匹配串长度为 M，DFSA-QS 算法最大跳转值为 N，则匹配时间消耗 $Time_1$ 为：

$$Time_1=t\times M/(N+1) \tag{11}$$

令 DFSA 同样应用于编码拆分环境，则匹配串长度变为 $2M$，DFSA-QS 算法通过一次失配比较可以跳过 $2N+1$ 个字节，整个匹配的时间消耗 $Time_2$ 为：

$$Time_2=t\times 2M/(2N+1) \tag{12}$$

由前文分析，可知 IDFSA-QS 算法在最坏情况下，一次可以跳跃 $2N+2$ 个字节，因此匹配耗时为：

$$Time_0=t\times 2M/(2N+2)=t\times M/(N+1)=Time_1<Time_2 \tag{13}$$

由于 SDFSA-QS 对于失效跳转表未进行编码拆分，即失效跳转表的基本字符单位为占两个字节的中文编码字符，因此以单字节为单位，SDFSA-QS 算法的最大跳转值为 $2(N+1)$ 即 $2N+2$，与本文算法相同，故此算法的时间性能与本文算法理论值相同，即同环境下 SDFSA-QS 算法的消耗时间为：

$$Time_3=Time_1=Time_0$$

综上可知，算法的时间消耗排序为：

$$Time_2>Time_1=Time_3=Time_0 \tag{14}$$

结合式(10)与式(14)，可知 IDFSA-QS 算法拥有最低的时间消耗期望与空间消耗期望，且为唯一最优算法。

5 实验结果与分析

对算法进行实验测试，除 SDFSA-QS 算法、DFSA-QS 算法与改良算法外，还加入了 DFSA 算法与 WM 算法进行测试对比。样本选取文学作品《基督山伯爵》的中文版本（上海译林出版社），共计 1005772 个汉字字符。实验查找在字典中随机抽取的 4 组模式串，每组串包含 7 个模式串，模式串最短长度从 2 到 5（未拆码前长度）递增，任何一个模式串发生匹配，则计数器加 1，此外不做任何操作，算法预处理与文件读入的时间不计入测试时间。

实验测试使用的关键代码如下。

代码 3 实验测试关键代码

```
QueryPerformanceCounter(&BeginTime);//获取匹配开始执行时间
MathSerch
(
  file_content,//模式串内容
  filesize,//模式串长度
  MatchingStruct,//匹配串结构体
  FuncNum//当前执行匹配算法编号
);//匹配执行模板函数
QueryPerformanceCounter(&EndTime);//获取匹配完成时间
MathTime=EndTime-BeginTime;//计算当前算法匹配执行时间
```

实验通过控制 MathSerch 函数的 FuncNum 参数，调入对应的匹配算法函数（1 代表 DFSA 算法，2 代表 WM 算法，3 代表 DFSA-QS 算法，4 代表 SDFSA-QS 算法，5 代表 IDFSA-QS 算法），并记录当前匹配算法函数执行所用时间。

实验统一在编码拆分环境下进行。实验运行于 Windows XP 平台，CPU 主频为3.09GHz，内存为 3GB，空间单位采用 32 位存储字长，时间单位采用计算机基本计时单位 Count（由 QueryPerformanceCounter 函数获得），空间单位采用 32 位字长。算法的时间与空间实验数据如表 5、表 6 所列。

表 5 不同最短模式串长度的时间性能（单位：Counts）

最短模式串长度	2	3	4	5
DFSA 算法	25012	25857	26601	26901
WM 算法	23701	15291	10285	7980
DFSA-QS 算法	14683	12021	10049	7453
SDFSA-QS 算法	11389	9763	8592	5991
IDFSA-QS 算法	11397	9757	8596	5994

表 6 不同最短模式串长度的空间性能（单位：32 位字长）

最短模式串长度	2	3	4	5
DFSA 算法	9216	13824	16640	22016
WM 算法	98310	98312	98312	98310
DFSA-QS 算法	9472	14080	16896	22272
SDFSA-QS 算法	74752	79360	82176	87552
IDFSA-QS 算法	9472	14080	16896	22272

实验数据曲线如图 1 与图 2 所示。

根据表 5 中实验数据，WM 算法、SDFSA-QS 算法、DFSA-QS 算法与 IDFSA-QS 算法的时间性能会随着最短模式长度降低而下降，这是由于这 3 种算法的最大跳转距离都取决于最短模式串长度，当最短模式串减小时，算法所能获取的跳转期望值也将下降。虽然 DFSA 算法对于最短模式串长度

不敏感，但各个环境下该算法的时间效率都是最低的，因此在中文环境下，DFSA 并不适用。

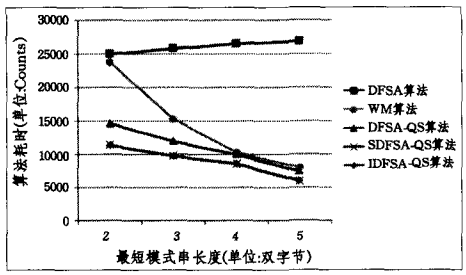

图 1　实验测试算法的时间消耗曲线

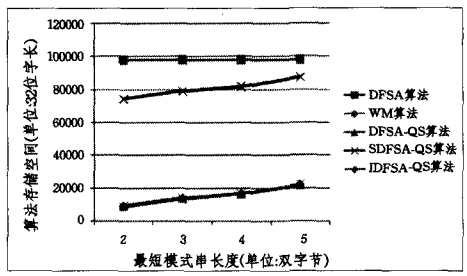

图 2　实验测试算法的空间消耗曲线

通过表 5 还可以看到，IDFSA-QS 算法在各测试组的时间性能都与 SDFSA-QS 算法基本相同(在图 1 中表现为两条性能曲线基本重合)，且明显高于其它算法。随着最短模式串长度的增大，IDFSA-QS 算法相对于 DFSA-QS 算法的速度提升率呈现下降趋势，这是由于随着 N 值的增长，提升率 RateAdd 将递减。实验结果与理论预测相符。

虽然 IDFSA-QS 算法的相对提升率随着最短模式串长度的增加表现出递减趋势，但其时间效率仍优于同等环境下的其它算法，证明了在中文拆分编码环境下，IDFSA-QS 算法的时间性能优于 DFSA-QS 算法与其他算法。

从图 2 中可以看到，WM 算法的空间消耗基本与模式串最短长度无关，这是因为 WM 算法的空间主要由所选 HASH 算法的散度决定，而与编码范围无关。DFSA 算法、IDFSA-QS 算法与 DFSA-QS 算法的空间消耗曲线基本重合，且明显优于另两种算法。

通过表 6 可以进一步发现，IDFSA-QS 算法与 DFSA-QS 算法的空间消耗完全相同，略优于 DFSA 算法，而相对于另两种算法，则有着明显的空间优势。

综上，IDFSA-QS 算法在中文检索环境下，其时间消耗与空间消耗都较小，即有最快的算法运行效率与最小的空间资源消耗。

结束语　本文结合实际的中文语义特点与其常用编码特征，优化改良了多模式匹配算法——DFSA-QS 算法，提出了一种适用于中文编码环境下的改良算法，在利用编码拆分降低了空间消耗、提升了算法实用性的同时，利用中文编码关联性提升了算法的时间效率。改良算法可使用在中文电子文档检索、网络数据包关键信息过滤、网络恶意行为匹配等中文多模式匹配环境中。

参 考 文 献

[1] Aho A V, Corasick M J. Efficient String Matching: An Aid to Bibliographic Search[J]. Communication of the ACM, 1975, 18(6): 333-340

[2] Wang Pei-feng, Li Li. Research on multi-pattern matching algorithms based on Aho-Corasick algorithm[J]. Application Research of Computers, 2011, 28(4): 1251-1254(in Chinese)
王培凤，李莉. 基于 Aho-Corasick 算法的多模式匹配算法研究[J]. 计算机应用研究，2011，28(4)：1251-1254

[3] Wu Sun, Udi M. A fast algorithm for multi-pattern searching: TR 94-17[R]. University of Arizona at Tuscon, 1994

[4] Wu Sun, Udi M. Agrep—A Fast Approximate Pattern Matchin Tool[C]//Usenix Winter Technical Conference. 1992: 153-162

[5] Wu Sun, Udi M. GLIMPSE: A Tool to Search Through Entire FileSystem[C]//Usenix Winter Technical Conferenee. 1994

[6] Liu Wei-guo, Hu Yong-gang. DHSWM: An improved multi-pattern matching algorithm based on WM algorithm[J]. Journal of Central South University (Science and Technology), 2011, 42(12): 3765-3771(in Chinese)
刘卫国，胡勇刚. DHSWM：一种改进的 WM 多模式匹配算法[J]. 中南大学学报(自然科学版)，2011，42(12)：3765-3771

[7] Wang Yong-cheng, Shen Zhou, Xu Yi-zhen. Improved Algorithms for Matching Multiple Patterns[J]. Journal of Computer Research and Development, 2002, 39(1): 55-60(in Chinese)
王永成，沈州，许一震. 改进的多模式匹配算法[J]. 计算机研究与发展，2002，39(1)：55-60

[8] Wang Yong-jin, Gu Nai-jie, Ren Chang-xin. A Word-length Based Wu-manber Multi-pattern Matching Algorithm[J]. Journal of Chinese Computer Systems, 2013, 34(7): 1650-1653(in Chinese)
汪永进，顾乃杰，任长新. 一种按字长匹配的 Wu-Manber 多模式匹配算法[J]. 小型微型计算机系统，2013，34(7)：1650-1653

[9] Liu Wei, Guo Yuan-bo, Huang Peng. Multi-Pattern Matching Engine Based on Bloom Filter[J]. Acta Electronica Sinica, 2010, 38(5): 1095-1099(in Chinese)
刘威，郭渊博，黄鹏. 基于 Bloom filter 的多模式匹配引擎[J]. 电子学报，2010，38(5)：1095-1099

[10] Zhu Yong-qiang, Jiang Xue. Analysis and research of Chinese multi-pattern matching algorithm performance[J]. Computer Technology and Development, 2013, 24(2): 67-70(in Chinese)
朱永强，江雪. 中文多模式匹配算法性能的分析与研究[J]. 计算机技术与发展，2013，24(2)：67-70

[11] Li Wei-nan, E Yue-peng, Ge Jing-guo, et al. Multi-Pattern Matching Algorithms and Hardware Based Implementation[J]. Journal of Software, 2006, 17(12): 2403-2415(in Chinese)
李伟男，鄂跃鹏，葛敬国，等. 多模式匹配算法及硬件实现[J]. 软件学报，2006，17(12)：2403-2415

[12] Sun Qin-dong, Huang Xin-bo, Wang Qian. Multiple Pattern Matching on Chinese/English Mixed Texts[J]. Journal of Software, 2008, 19(3): 674-686(in Chinese)
孙钦东，黄新波，王倩. 面向中英文混合环境的多模式匹配算法[J]. 软件学报，2008，19(3)：674-686

[13] Gao Chao-qin, Chen Yuan-yan, Li Mei. Fast multi-pattern matching algorithm for intrusion detection[J]. Journal of Computer Applications, 2008, 28(1): 82-84(in Chinese)
高朝勤，陈元琰，李梅. 一种面向入侵检测的快速多模式匹配算法[J]. 计算机应用，2008，28(1)：82-84

[14] Shen Zhou, Wang Yong-cheng, Xu Yi-zhen. A Fast Multiple Pattern Algorithm for Chinese String Matching[J]. Journal of Shanghai Jiaotong University, 2001, 35(9): 1285-1289(in Chinese)
沈州，王永成，许一震. 一种面向中文的快速字串多模式匹配算法[J]. 上海交通大学学报，2001，35(9)：1285-1289