一种时间复杂度最优的精确串匹配算法*

贺龙涛+, 方滨兴, 余翔湛

(哈尔滨工业大学 计算机网络与信息安全技术研究中心,黑龙江 哈尔滨 150001)

A Time Optimal Exact String Matching Algorithm

HE Long-Tao+, FANG Bin-Xing, YU Xiang-Zhan

(Research Center of Computer Network and Information Security Technology, Harbin Institute of Technology, Harbin 150001, China)

+ Corresponding author: Phn: +86-10-82990825, E-mail: hlt@hit.edu.cn, http://cucme.hit.edu.cn/

Received 2003-09-22; Accepted 2004-11-15

He LT, Fang BX, Yu XZ. A time optimal exact string matching algorithm. *Journal of Software*, 2005,16(5): 676−683. DOI: 10.1360/jos160676

Abstract: Existing string matching algorithms typically set the sliding window size as the pattern length. This paper presents a Linear DAWG Matching (LDM) algorithm, which divides the text into $[n/m]$ overlapping windows of length $2m-1$. In the windows, the algorithm attempts at m positions in batches. It firstly searches pattern prefixes from middle to left with a reversed suffix automaton, shifts to next window directly when it fails, otherwise, scans the corresponding suffixes forward with a finite automaton. Theoretical analysis shows that LDM has optimal time complexities in the worst ($O(m)$), best ($O(n/m)$) and average cases ($O(n(\log_\sigma m)/m)$). Experimental comparison of LDM with the existing algorithms validates this theoretical claims of average case for searching long patterns. It further reveals that LDM is also efficient for searching short patterns on large alphabets. Thus, LDM algorithm not only suits for off-line pattern matching, but also fits in high-speed online pattern matching.

Key words: suffix automaton; finite automaton; LDM (linear DAWG matching) algorithm; string matching; complexity analysis

摘 要: 现有的串匹配算法通常以模式长度作为滑动窗口大小.在窗口移动后,往往会丢弃掉一些已扫描正文的信息.提出了 LDM(linear DAWG matching)串匹配算法,该算法将正文分为$[n/m]$个相互重叠、大小为 $2m-1$ 的扫描窗口.在每个扫描窗口内,算法批量地尝试 m 个可能位置,首先使用反向后缀自动机从窗口中间位置向前扫描模式前缀;若成功,则再使用正向有限状态自动机从中间位置向后扫描剩余的模式后缀.分析证明,LDM 算法的最差、最好、平均时间复杂度分别达到了理论最好结果:$O(n)$,$O(n/m)$,$O(n(\log_\sigma m)/m)$.实际性能测试也验证了

* Supported by the National High-Tech Research and Development Plan of China under Grant No.2001AA147010B (国家高技术研究发展计划(863)); the Defense Pre-Research Project of the 'Ninth Five-Year-Plan' of China under Grant No.41315.7.3 (国家"九五"国防预研基金)

作者简介: **贺龙涛**(1974−),男,贵州遵义人,博士生,主要研究领域为计算机网络与信息安全,网络病毒检测与防范,入侵检测,信息处理,串匹配;**方滨兴**(1960−),男,博士,教授,博士生导师,主要研究领域为信息安全,计算机网络,并行计算;**余翔湛**(1973−),男,博士生,讲师,主要研究领域为计算机网络生存性技术.

平均时间复杂度最优这一理论结果.而且,对于在较大字母表下查找短模式的情况,LDM 算法速度在被测试算法中最快.总之,LDM 算法不但适合进行离线模式匹配,而且还特别适合需要进行在线高速匹配的应用.

关键词: 后缀自动机;有限状态自动机;LDM 算法;串匹配;复杂度分析

中图法分类号: TP301 **文献标识码**: A

在计算机科学领域,串匹配算法一直都是研究焦点之一.在拼写检查、语言翻译、数据压缩、搜索引擎、网络入侵检测、计算机病毒特征码匹配以及 DNA 序列匹配等应用中,都需要进行串匹配.串匹配就是在正文串中查找模式串的一个或所有出现.

在本文中,模式串表示为 $x=x_1x_2\ldots x_m$,长度为 m;正文串表示为 $y=y_1y_2\ldots y_n$,长度为 n;字母表以 Σ表示,大小为 σ.[]表示向下取整.

现有的串匹配算法,通常是以模式长度 m 为扫描窗口大小,在窗口中使用不同的扫描策略来进行匹配[1].依据扫描策略的不同,串匹配算法可以粗略地分为以下几大类:

(1) 从前往后匹配模式前缀.KMP[2],Shift Or[3]等算法从前往后进行扫描,使用自动机记住已匹配模式前缀的正文内容,并依据这些内容来确定是否已经匹配成功.这类算法的最差时间复杂度都达到了理论最优结果 $O(n)$,但是由于只能逐个字符进行扫描,它们的平均时间复杂度往往都较差.

(2) 从后往前匹配模式后缀.为了提高对已匹配信息的利用率,BM[4]算法及其各种变形[5−7]在窗口中从后往前进行扫描,记住已匹配模式后缀的正文内容,并依据这些内容来进行窗口移动.虽然这类算法的最差时间复杂度为 $O(mn)$,但是由于所匹配的后缀在当前窗口后部,能够更好地增加窗口移动距离,使得算法实际上对正文中的很多字符都不必进行扫描,因此,它们的平均时间复杂度达到亚线性.

(3) 从后往前匹配模式前缀.为了将当前扫描窗口中更多的有用信息挖掘出来,BDM[8],RF[9]等算法在从后往前进行扫描时,反向使用模式逆串的后缀自动机(又称为 directed acyclic word graph,简称 DAWG)[10]来匹配模式的前缀,这样可以增大窗口移动的距离,从而获得更好的平均时间复杂度,达到理论最优结果 $O(n(\log_\sigma m)/m)$[11].

本文提出一种新的串匹配算法,称为 LDM(linear DAWG matching)算法.该算法既有从前往后扫描的算法所具有的线性最差时间复杂度的优点,又有扫描模式后缀的算法所具有的亚线性平均时间复杂度.而且由于在每次尝试时都将当前尝试位置附近的所有有用信息都挖掘出来,使得每次移动时损失的信息量尽量小,更加有效地利用了已识别正文的最后部分.分析表明,该算法最差、最好时间复杂度分别达到了理论最优结果 $O(n)$,$O(n/m)$.而且,平均时间复杂度也达到最优结果 $O(n(\log_\sigma m)/m)$.实验结果也验证了这一点.而且在大字母表的情况下,LDM 算法优于现有算法,随着模式长度的增加,只有 RF 算法能够和 LDM 算法速度相接近.

1 算法预备

1.1 算法思想描述

考察在每次尝试开始时,正文中正在扫描的字符 y_i,它不但与其前面的 $m-1$ 个字符 $y_{i-m+1}y_{i-m+2}\ldots y_{i-1}$ 有关系(BM 类算法正是依据这种思想在大小为 m 的窗口中使用从后向前扫描的方式),而且还与其后的 $m-1$ 个字符 $y_{i+1}y_{i+2}\ldots y_{i+m-1}$ 有关系(KMP 类算法依据这种思想在大小为 m 的窗口中使用从前向后扫描的方式).这样,当前扫描的字符与其前后各 $m-1$ 个字符构成了大小为 $1+(m-1)+(m-1)=2m-1$ 的窗口.在这个窗口中,首先使用后缀自动机从窗口中间位置反向扫描模式前缀,然后使用正向有限状态自动机从中间位置往后扫描相应的模式后缀,这样就能够完全将窗口中心位置附近与之有关的有用信息全部扫描出来,在窗口移动前查找出正文中包含窗口中心位置字符的所有模式出现,减小窗口移动时的信息损失.

在算法中,分别使用了两类自动机:反向最小后缀自动机和正向有限状态自动机.

1.2 最小后缀自动机

一个串 x 的最小后缀自动机 SA(suffix automaton),记为 $SA(x)$,即识别串 x 的所有后缀的最小确定自动机.

如图 1 所示,该自动机可以接受串 *baabbaa* 的所有后缀:*ε*,*a*,*aa*,*baa*,*bbaa*,*abbaa*,*aabbaa*,*baabbaa*.

已经存在以线性复杂度构造 SA 的算法[10],对长度为 *m* 的串 *x*,构造时间为 *O*(*m*),且 *SA*(*x*)的大小(包括节点数与边数)为 *O*(*mσ*).

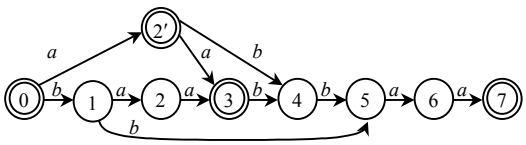

Fig.1 *SA*(*baabbaa*)

图 1 *SA*(*baabbaa*)

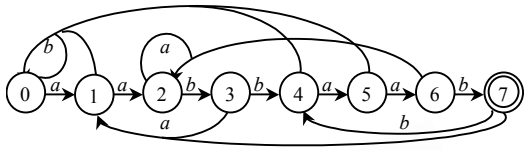

Fig.2 *FFA*(*aabbaab*)

图 2 *FFA*(*aabbaab*)

1.3 有限状态自动机

本文所用到的正向有限状态自动机 FFA(forward finite automata)实际上就是正向识别模式的一般有限状态自动机,只是在运行方式上有所不同.由于在反向后缀自动机运行结束时算法已经匹配了模式的一个(或多个)前缀,所以正向有限状态自动机运行时不是从传统的开始状态开始,而是从已匹配的最大前缀所对应的状态开始.开始状态集合中不包含传统的初始状态,这是因为当前一阶段识别的前缀为*ε*时,不需要使用 FFA 进行扫描.在实现中,常常用已识别前缀的长度来表示状态.如图 2 所示,该自动机使用不同的开始状态 1,2,3,4,5,6,7,分别可以接受串 *aabbaab* 的不同后缀:*abbaab*,*bbaab*,*baab*,*aab*,*ab*,*b*,*ε*.

FFA 的构造也是一个简单的问题,使用与 DFA 匹配算法[10]相同的构造算法构造模式的 FFA,其构造时间复杂度是 *O*(*m*+*σ*),空间复杂度是 *O*(*mσ*).

2 LDM 串匹配算法

本文算法主要使用 DAWG 来扫描正文,且除了与 BDM 算法一样达到平均时间复杂度最优以外,还达到了线性最差时间复杂度,因此称为 LDM 算法.

2.1 LDM算法

算法初始化主要是引用已有算法来构造模式逆串 x^r 的后缀自动机 $SA(x^r)$,以及模式 *x* 的正向有限状态自动机 *FFA*(*x*).在介绍算法扫描部分之前,我们先引入以下定义.

定义 1. 本算法中,尝试位置集定义为正文中的一些特殊位置:{*km*|1≤*k*≤⌈*n*/*m*⌉};尝试窗口定义为正文中以尝试位置为中心、大小为 2*m*−1 的滑动窗口(除了第⌈*n*/*m*⌉个窗口,它只有 *n*−*m*⌈*n*/*m*⌉+*m* 个字符,可以简单地使用不属于*Σ*的符号 *c* 来填充补足 2*m*−1 个符号,而不影响匹配结果.由于它对算法没有太大影响,以后将不再对它作特殊描述),其中尝试位置字符及之前的 *m*−1 个字符定义为前窗口,尝试位置之后的 *m*−1 个字符定义为后窗口.

这样,算法将正文串分为相互重叠的⌈*n*/*m*⌉个大小为 2*m*−1 的窗口,每个窗口和其前后的窗口都分别有 *m*−1 个字符相同,而尝试位置只在其所代表的尝试窗口中出现.算法依次对窗口 $y_{(k-1)m+1}\cdots y_{km}\cdots y_{(k+1)m-1}$ 进行尝试,其中 *k* 从 1 到⌈*n*/*m*⌉.

在扫描窗口内,算法分为两个阶段.首先使用反向后缀自动机 $SA(x^r)$在前窗口中从尝试位置反向扫描模式前缀;然后根据扫描结果决定是否使用正向有限状态自动机 *FFA*(*x*)在后窗口中向后扫描剩余的模式后缀.算法使用 *l*,*r* 为正文扫描指针,*l* 指示 $SA(x^r)$在前窗口中从后向前扫描的位置,*r* 则表示 *FFA*(*x*)在后窗口中从前向后扫描的位置.*L*,*R* 分别表示 $SA(x^r)$和 *FFA*(*x*)的状态.另外,在反向扫描阶段,*R* 还记录已匹配的最大后缀的长度.由于 *FFA*(*x*)以已匹配前缀的长度作为状态,当反向扫描结束时,*R* 中保存的即为 *FFA*(*x*)的开始状态.以 $y_{(k-1)m+1}\cdots y_{km}\cdots y_{(k+1)m-1}$ 为例:

阶段 1.如图 3 所示,使用 $SA(x^r)$从后往前扫描前窗口 $y_{(k-1)m+1}\cdots y_{km}$,直到 $SA(x^r)$没有动作.其中,当自动机状态为终态时,记录下一扫描位置 *l*(即已匹配最大前缀的长度)到变量 *R*.$SA(x^r)$停止动作时,如果 *R*>0,则 $y_{km-R+1}\cdots y_{km}$ 为算法在前窗口中扫描到的最大前缀,算法进入第 2 阶段;当 *R*=0 时,表示扫描到的最大前缀为*ε*,算法可以直接向后跳跃,继续对下一个尝试窗口进行扫描.

阶段 2.如图 4 所示,在后窗口 $y_{km+1}\ldots y_{(k+1)m-1}$ 中,由状态 R 开始,使用 $FFA(x)$从前往后扫描.由于从当前状态到达终态需要至少 $m-R$ 次移动,而当前窗口中还有 $m-r+1$ 个字符(y_{km+r}为下一要读字符),当 $m-R>m-r+1$(即 $R\leq r$)时,当前窗口中不再存在模式出现,结束扫描.其中,当自动机状态为终态时,输出当前扫描位置.

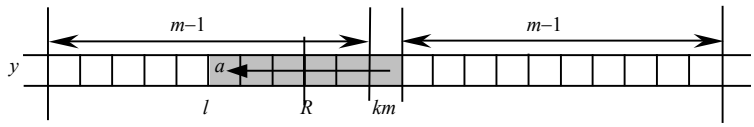

Fig.3 Searching backward for the pattern prefixes with $SA(x^r)$, until no movement at a. Record the current position in R when a terminal state is reached

图 3 使用 $SA(x^r)$从后向前查找模式前缀,直到在 a 处自动机没有动作,其中终态时记录当前位置到 R

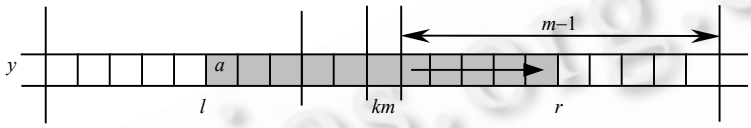

Fig.4 Searching forward for the pattern suffixes with $FFA(x)$, until no pattern exists in right window, output the position when the terminal state is reached

图 4 使用 $FFA(x)$从前向后查找模式前缀,直到确定当前窗口中没有其他模式串存在,其中遇到终态时输出当前位置

算法的伪代码如图 5 所示.其中,$\delta_{SA}(q,c)$是反向后缀自动机 $SA(x^r)$的状态转移函数,表示在 $SA(x^r)$中从状态 q 读取符号 c 到达的状态,若该状态不存在,则$\delta_{SA}(q,c)=\varepsilon.\delta_{FFA}(q,c)$表示正向有限状态自动机的状态转移函数.为简洁起见,忽略了一些优化和边界检查代码.

```
LDM (x=x1x2…xm,y=y1y2…yn)
1.   Preprocessing
2.       Build SA(x^r) and FFA(x)
3.   Search
4.       For k∈1...[n/m] do
5.           L←0,R←0,l←0,r←0
6.           While L≠ε do
7.               L←δSA(L,ykm−l)
8.               l←l+1
9.               If L is terminal then R←l
10.          End of while
11.          If R=0 then continue
12.          While true do
13.              If R is terminal then
14.                  Output((k−1)m+r)
15.              End of if
16.              r←r+1
17.              If R≤r then break
18.              R←δFFA(R,ykm+r)
19.          End of while
20.      End of for
```

Fig.5 Pseudo-Code of the LDM algorithm

图 5 LDM 算法伪代码

2.2 一个匹配实例

以在正文 $y=abbabaabbaababbabbab$ 中查找模式 $x=aabbaab$ 的所有出现为例:

算法初始化,构造如图 1 所示的 $SA(x^r)$和如图 2 所示的 $FFA(x)$.

在扫描阶段,算法初态为$\langle abbabaS_0abbaaba\rangle bbabbab$,其中$\langle\rangle$表示当前窗口位置,$S_k$ 表示当前使用后缀自动机,状态 k 来扫描正文,F_k 表示当前使用正向有限状态自动机,状态 k 来扫描正文,S 或 F 位置后的第 1 个字符为

当前读入字符.另外,带下划线表示该字符已扫描过.

(1) 开始第 1 次窗口扫描,当前尝试位置为 1m=7.

反向扫描阶段:初始化 L,R,l,r 为 0,使用后缀自动机 $SA(x)$反向扫描:

1. $\langle abbabaS_0abbaaba\rangle\ldots,L=\delta_{SA}(0,a)=2'.l=1.2'$是一个终态,$R$=1.
2. $\langle abbabS_{2'}aabbaaba\rangle\ldots,L=\delta_{SA}(2',a)=3.l=2.3$ 是一个终态,R=2.
3. $\langle abbaS_3baabbaaba\rangle\ldots,L=\delta_{SA}(3,a)=4.l=3$.
4. $\langle abbS_4abaabbaaba\rangle\ldots,L=\delta_{SA}(4,a)=\varepsilon$.反向扫描阶段结束.

正向扫描阶段:$FFA(x)$初始状态 R=2,表明已识别长度为 2 的前缀,即 aa:

5. $\langle abbabaaF_2bbaaba\rangle\ldots,r=1.R=\delta_{FFA}(2,b)=3$.
6. $\langle abbabaabF_3baaba\rangle\ldots,r=2.R=\delta_{FFA}(3,b)=4$.
7. $\langle abbabaabbF_4aaba\rangle\ldots,r=3.R=\delta_{FFA}(4,a)=5$.
8. $\langle abbabaabbaF_5aba\rangle\ldots,r=4.R=\delta_{FFA}(5,a)=6$.
9. $\langle abbabaabbaaF_6ba\rangle\ldots,r=5.R=\delta_{FFA}(6,b)=7.7$ 为终态,表明已匹配到一个模式串,输出位置 6.
10. $\langle abbabaabbaabF_7a\rangle\ldots,r=6.R=\delta_{FFA}(7,a)=1<r$,正向扫描结束.

(2) 开始第 2 次窗口扫描,当前尝试位置为 2m=14.

反向扫描阶段:初始化 R,L,r,l 为 0,使用后缀自动机 $SA(x)$反向扫描:

11. $\ldots\langle bbaabaS_0bbabbab\rangle,L=\delta_{SA}(0,b)=1.l=1$.
12. $\ldots\langle bbaabS_1abbabbab\rangle,L=\delta_{SA}(1,a)=2.l=2$.
13. $\ldots\langle bbaaS_2babbabbab\rangle,L=\delta_{SA}(2,b)=\varepsilon$.反向扫描阶段结束.且 R=0,无须进行正向扫描,全部扫描结束.

这样,LDM 算法一共读取 13 个正文字符,在位置 6 识别出模式串的一个出现.

3 复杂度分析

LDM 算法用到了两个自动机 $SA(x^r)$和 $FFA(x)$,它们的空间复杂度都是 $O(\sigma m)$,所以 LDM 算法的空间复杂度为 $O(\sigma m)$.而算法的预处理主要就是构造两个自动机 $SA(x^r)$和 $FFA(x)$,它们的构造时间都是 $O(m)$,所以 LDM 算法的预处理时间为 $O(m)$.

以下证明在扫描时间上,LDM 算法的时间复杂度在各个方面都达到了理论最佳结果.

定理 1. LDM 算法最差时间复杂度为 $O(n)$.

证明:首先,算法只进行$[n/m]$次尝试;其次,在尝试窗口中,由于窗口大小为 $2m-1$,且依据算法,最多对每个位置进行 1 次扫描,也就是说,算法最多进行$(2m-1)[n/m]\leq 2n-n/m<2n$ 次正文字符扫描,即,LDM 算法最差时间复杂度为 O(n). □

在 a^n 中查找 a^m 的所有出现就达到了$(2m-1)[n/m]$次正文字符扫描的上界.

定理 2. LDM 算法最优时间复杂度为 $O(n/m)$.

证明:当每次在尝试窗口中,对尝试位置字符进行扫描都无下动作的时候,只进行 1 次扫描,于是将总共进行$[n/m]$次扫描.即,LDM 算法最优时间复杂度为 $O(n/m)$. □

在 a^n 中查找 b^m 的所有出现就达到了$[n/m]$次正文字符扫描的下界.

串匹配算法平均时间复杂度通常是在等概率模型下讨论的,即假设字母表中的所有字符在正文和模式中出现的概率相等.

定理 3. 在等概率模型下,LDM 算法平均时间复杂度为 $O(n(\log_\sigma m)/m)$.

证明:首先,计算在每次尝试时,在尝试窗口中扫描正文字符数的平均值:

设 $d=[2\log_\sigma m]$,假设满足条件 $m>d$(除了 σ=2 且 $m\leq 4$ 这种情况之外都满足这个条件),分以下情况讨论:

(1) 在第 1 阶段从尝试位置反向扫描时,扫描超过 d 个字符.由于 x 的长度为 d 的子串有 $m-d+1$ 个,而长度为 d 的串总数为 σ^d 个,则这种情况发生的概率为$(m-d+1)/\sigma^d$.而这种情况下最坏的结果为扫描了整个窗口的 $2m-1$ 个字符.

(2) 在第 1 阶段从尝试位置反向扫描时,未超过 d 个字符,但扫描到了至少 1 个 x 的前缀,且在第 2 阶段扫描结束时,两个阶段扫描的字符总数超过 d.这种情况的概率为 $1/\sigma^d$,而且最多扫描 $m+d-1$ 个字符.

(3) 其他情况.包括第 1 阶段扫描字符数不超过 d,且未扫描到 x 的前缀,以及两个阶段扫描字符总数不超过 d.总之,两种情况下扫描的字符数都不超过 d.这种情况发生的概率为 $1-(m-d+1)/\sigma^d-1/\sigma^d$.

综合以上情况,在一次尝试中,扫描的字符数期望值不超过:

$$\frac{m-d+1}{\sigma^d}\times(2m-1)+\frac{1}{\sigma^d}\times(m+d-1)+\left(1-\frac{m-d+1}{\sigma^d}-\frac{1}{\sigma^d}\right)\times d \tag{1}$$

整理,得到:

$$d+\left(\frac{2m^2-3md-d^2+d-1}{\sigma^d}\right) \tag{2}$$

因为 $\sigma^d=\sigma^{|2\log_\sigma m|}>\sigma^{2\log_\sigma m-1}=m^2/\sigma$,所以括号内算式的值小于某个常数 C.因此,尝试窗口内扫描字符数的平均值为 $O(d)=O(\log_\sigma m)$.又由于算法只进行$[n/m]$次尝试,因此算法的平均时间复杂度为 $O(n(\log_\sigma m)/m)$,达到最优. □

4 实验结果

为了检测 LDM 算法离线匹配性能(即平均时间复杂度),将 LDM 算法与现有的其他 3 类算法中具有代表性的算法进行测试:从前往后扫描前缀的 KMP、从后往前扫描后缀的 BM、从后往前扫描前缀的 RF.测试中分别对大小为 2,4,8,16,32,64,128 和 256 的字母表进行实验.在每个字母表中,分别随机生成了长度为 10M 的正文.分别将长度从 2~64 的模式在每个正文中进行匹配.对每个长度,随机生成了 5 000 个模式.实验条件为 Linux 主机,PIII,双 CPU 主频 933MHz,内存 1G.算法都使用 C 语言实现,并使用相同接口调用.实验中打开 gcc 编译器 -O3(最优化)开关,为了计算每个算法的平均运行时间,还打开了-pg 开关.测试结果按照模式长度分类统计处理,得到结果如图 6 所示.图中横轴为模式长度,纵轴为每模式每兆正文平均匹配时间,单位为 s.

实验结果显示,在大字母表下(32 以上),LDM 算法最快.运行时间最快的是 KMP,8%;然后是 BM 的 55%;RF 的 64%.随着模式长度的增加,RF 算法达到了 LDM 算法相同的运行时间,这是因为它们的平均时间复杂度相同.

在中字母表下(8 和 16),LDM 算法与 RF 算法运行时间基本相同,都是最快.随着模式长度的增加,它们的运行时间与 KMP 算法和 BM 算法的运行时间差距加大,与 KMP 算法的运行时间相比,由 70%减少到 13%;与 BM 算法相比,由 82%减少到 44%.

在小字母表(2 和 4)下,LDM 算法比 RF 算法稍慢,但随着模式长度的增加,两者的差距逐渐减少,而与 KMP 算法和 BM 算法的差距扩大,运行时间最快的是 KMP,12%;然后是 BM 的 40%.

总之,对于在大字母表下查找较短模式的情况,LDM 算法在测试算法中最快.在各种字母表大小的情况下,随着模式长度的增加,LDM 算法与 RF 算法运行时间越来越接近,而比 KMP 和 BM 算法运行时间的优势逐渐加大,这与 LDM 与 RF 算法平均时间复杂度同为最优的理论分析结果相一致.

5 结束语

本文提出了使用相互重叠的大窗口来扫描正文的思想.在窗口中,先从中间位置往前扫描模式前缀,然后再扫描相应的模式后缀.这样,算法并行地扫描 m 个模式出现位置,最大化地挖掘出当前扫描位置附近的有用正文信息,使得在每次窗口移动时丢失的信息达到最小,提高了算法效率,使得最差、最好和平均时间复杂度都达到了最优结果:$O(n)$,$O(n/m)$以及 $O(n(\log_\sigma m)/m)$.性能测试实验也验证了平均时间复杂度最优这一理论结果,而且在查找较短模式时,LDM 算法在大字母表情况下是最快的.LDM 算法的平均时间复杂度最优这一特性表明它适合进行离线模式匹配的应用,而且由于其最差时间复杂度也最优,因此它还特别适合需要进行在线高速匹配(不能存在瞬时耗时过长的尖峰现象)的应用,例如高速网络下入侵检测系统.

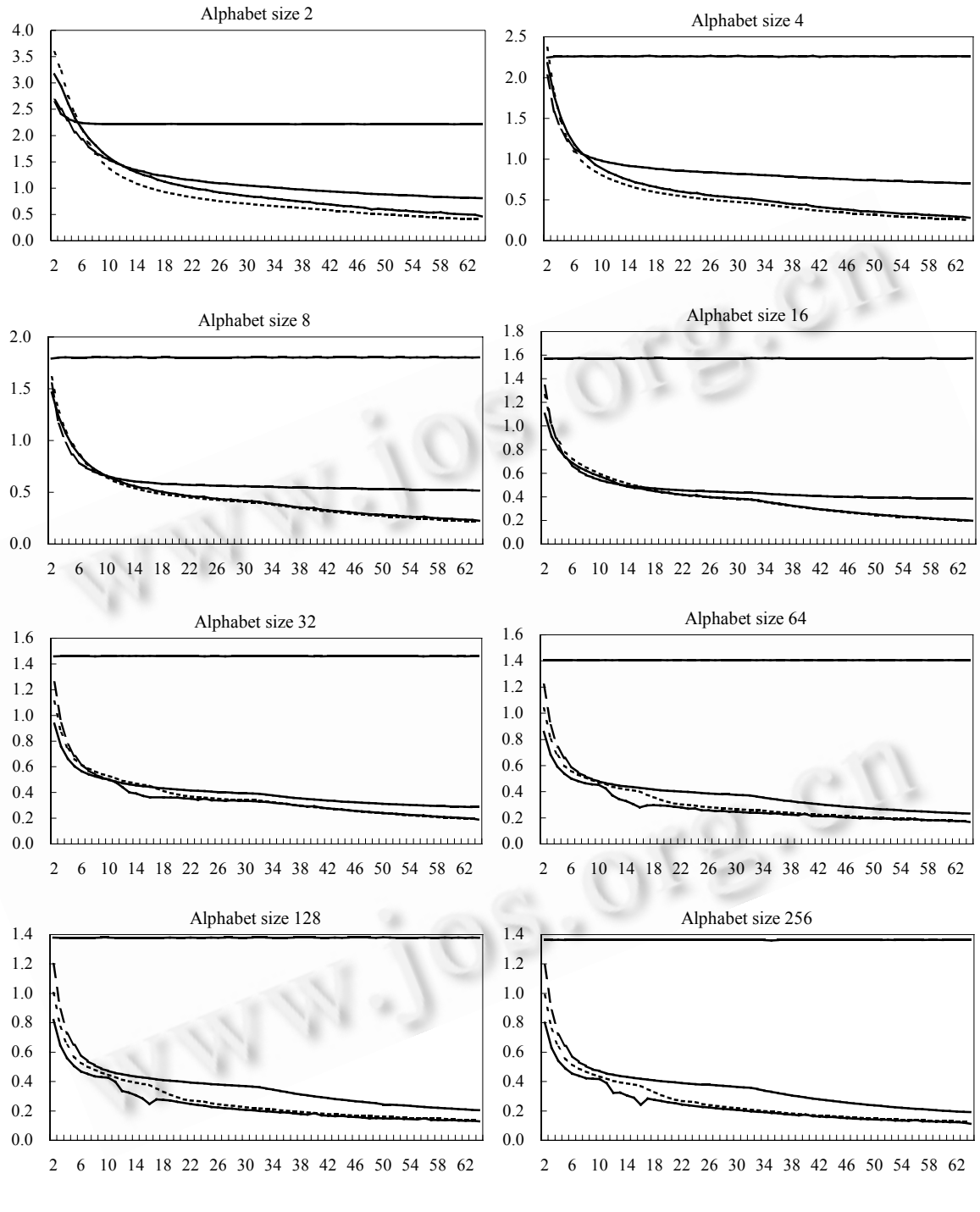

Fig.6 Experimental results in running time of the string matching algorithms on random texts of size 10Mb and 5 000 rand patterns length from 2 to 64 on alphabets of size 2, 4, 8, 16, 32, 64, 128 and 256

图 6 在大小分别为 2,4,8,16,32,64,128 和 256 的字母表上,对长度分别为 10Mb 的随机正文与长度从 2~64 的随机模式各 5 000 个进行匹配的实验结果

最近出现了直接使用 Bit-Parallelism 技术模拟的不确定后缀自动机来匹配模式前缀,进行串匹配的 BNDM 算法[12],我们已将 Bit-Parallelism 技术与本文思想结合[13],继续作深入的研究.后缀自动机在多模式匹配中也有良好的应用[14].同样,本文思想也可以扩展到多模式匹配的问题中,我们将继续进行深入的研究,以获得各项时间复杂度最优[15]的多模式匹配算法.本文算法还可以推广到模糊匹配中[16].另外,本文算法在各窗口内的操作是完全独立的,因此,加以简单的改进就可以应用到并行串匹配中.

致谢 在此,我们向对本文的工作给予支持的罗浩博士生以及对本文提出有益建议的审稿专家们表示感谢.

References:

[1] Charras C, Lecroq TT. Handbook of Exact String Matching Algorithms. London: King's College London Publications, 2004.

[2] Knuth DE, Jr. Morris JH, Pratt VR. Fast pattern matching in strings. SIAM Journal on Computing, 1977,6(1):323−350.

[3] Baeza-Yates RA, Gonnet GH. A new approach to text searching. Communications of the ACM, 1992,35(10):74−82.

[4] Boyer RS, Moore JS. A fast string searching algorithm. Communications of the ACM, 1977,20(10):762−772.

[5] Sunday DM. A very fast substring search algorithm. Communications of the ACM, 1990,33(8):132−142.

[6] Wang YC, Chen GL, Han KS. Faster algorithm for single pattern matching. Journal of Shanghai Jiaotong University, 2001,35(2):192−196 (in Chinese with English abstract).

[7] Cantone D, Faro S. Fast-Search: A new efficient variant of the Boyer-Moore string matching algorithm. In: Alberto A, Massimo M, eds. Proc. of the 2nd Int'l Workshop on Experimental and Efficient Algorithms (WEA 2003). Lecture Notes in Computer Science 2647, Heidelberg: Springer-Verlag, 2003. 47−58.

[8] Crochemore M, Rytter W. Text Algorithms. Oxford: Oxford University Press, 1994.

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

[10] Crochemore M, Hancart C. Automata for matching patterns. In: Rosenberg G, Salomaa A, eds. Handbook of Formal Languages, volume 2: Linear Modeling: Background and Application. Heidelberg: Springer-Verlag, 1997. 399−462.

[11] Yao AC. The complexity of pattern matching for a random string. SIAM Journal on Computing, 1979,8(3):368−387.

[12] Navarro G, Raffinot M. Fast and flexible string matching by combining bit-parallelism and suffix automata. ACM Journal of Experimental Algorithmics, 2000,5(4):1−36.

[13] He LT, Fang BX. Linear nondeterministic dawg string matching algorithm. In: Alberto A, Massimo M, eds. Proc. of the 11th Int'l Symp. on String Processing and Information Retrieval (SPIRE 2004). Lecture Notes in Computer Science 3246, Heidelberg: Springer-Verlag, 2004. 70−71.

[14] Raffinot M. On the multi backward dawg matching algorithm (MultiBDM). In: Baeza-Yates R, ed. Proc. of the 4th South American Workshop on String Processing. Valparaíso: Carleton University Press, 1997. 149−165.

[15] Navarro G, Fredriksson K. Average complexity of exact and approximate multiple string matching. Theoretical Computer Science, 2004,321(2-3):283−290.

[16] Navarro G. A guided tour to approximate string matching. ACM Computing Surveys, 2001,33(1):31−88.

附中文参考文献:

[6] 王永成,陈桂林,韩客松.一种快速的多模式字符串匹配算法.上海交通大学学报,2001,35(2):192−196.