第 36 卷 第 1 期 计 算 机 工 程 2010 年 1 月
Vol.36 No.1 Computer Engineering January 2010
·人工智能及识别技术· 文章编号：1000—3428(2010)01—0208—03 文献标识码：A 中图分类号：TP311

一种基于反向有限自动机的多模式匹配算法

关 超，蒋建中，郭军利
(解放军信息工程大学信息工程学院，郑州 450002)

摘 要：在基于有限自动机的多模式匹配算法 DFSA 的基础上，结合改进的 BM 单模式匹配算法的优点，提出一种快速的多模式字符串匹配算法。在一般情况下，该算法不需要匹配目标文本串的每个字符，能充分利用匹配过程中本次匹配不成功的信息和已成功的信息，跳过尽可能多的字符。实验表明，模式串较短时，该算法需要的时间约为 DFSA 的 1/2，模式串较长时，所需时间约为 DFSA 算法的 1/3。
关键词：多模式匹配；有限自动机；匹配算法

Multiple Patterns Match Algorithm Based on Reverse Finite State Automata

GUAN Chao, JIANG Jian-zhong, GUO Jun-li
(Information Engineering College, PLA Information Engineering University, Zhengzhou 450002)

【Abstract】A fast algorithm to perform multiple patterns match in a string is described. The proposed match algorithm is based on the concept of Deterministic Finite State Automata(DFSA), combining the advantages of Boyer Moore's algorithm, the fast algorithms for single pattern matching. In general, it does not need inspect each character of the string. It skips as many characters as possible by making full use of the succeeded match information and failure information during the matching. The proposed algorithm achieves excellent performance in the cases of both short patterns and long patterns. Experimental results show that in the case of short patterns, the time it takes to search a string is 1/2 of the time DFSA method takes, and in the case of long patterns, the time is only 1/3 of the time by DFSA method.
【Key words】multiple patterns match; finite state automata; match algorithm

1 概述

在文本串中查找指定的模式串，即所谓字符串匹配是计算机进行信息处理的一个基本问题[1-2]，一直受到人们关注。多模式匹配是从文本串 $S[1: n]$中一次查找多个模式串 P_1，P_2,…, P_q。所有模式串形成模式串集合$\{P\}$，其中，q 为模式串集合的数目；模式串 P_k 的长度是 m_k；*minlen* 是最短模式串的长度，即 $minlen=\min\{m_k \mid 1\leqslant m_k \leqslant q\}$。3 种经典的单模式匹配算法是：Knuth-Morris-Pratt(KMP)，Boyer-Moore(BM)，Quick-Search(QS)。多模式匹配的经典算法有：基于有限自动机(DFSA)算法，它把模式串集合转化成有限自动机，然后只需要对文本串进行一次扫描就可找出所有模式串；FS 算法把 DFSA 和 BM 算法相结合，在匹配时实现跳跃式查找；Wu-Manber 算法等。

文献[3]对单模式匹配 BM 算法进行了改进。文献[4]对多模式匹配 DFSA 算法进行了改进。本文提出的算法在 DFSA 算法的基础上吸收改进的 BM 算法中充分利用本次匹配不成功的思想，在模式串匹配失败后，跳过尽可能多的字符，实现了更快的匹配过程。

2 现有算法介绍

2.1 BM算法

文献[5]提出了 BM 算法。该算法在匹配过程中模式串 $p[0, 1,\cdots,m-1]$由左向右移动，而字符的比较却自右向左进行，即按 $p[m-1], p[m-2],\cdots,p[0]$次序进行比较，当文中字符与模式不匹配时，根据 2 个预先定义的偏移函数 *Badchar* 和 *Goodsuffix* 计算出偏移量。BM 算法的关键是，对模式串 P 定义一个从字母到正整数的映射函数 *dist*，也称为滑动距离函数，它给出了任意字符 c 在模式串中的位置，*dist* 函数具体定义如下：

$$Dist[c]=\begin{cases} m; c\neq p[j](1\leqslant j\leqslant m) & c在P中未出现 \\ m-j, j=\max\{j \mid p[j]=c, 1\leqslant j\leqslant m-1\} & 其他情况 \end{cases}$$

2.2 FS算法

FS 算法吸取 BM 算法的思想，模式串是从后向前比较在预处理时。首先把模式串集合转换成一个反向的树型有限自动机。构造出 *goto* 函数和 output 函数。然后构造 $skip_1$ 和 $skip_2$ 函数。在匹配过程中。匹配从 $s=0$, $i=minlen$ 开始。根据 *goto* 函数进行。匹配成功。$i=i-1$；匹配失败时，如果 $s=0$，模式串集右移 $skip_1(s[i])$，即 $i=i+skip_1(s[i])$；否则，模式串集右移 $skip_2(s,s[i])$。$skip_1$ 定义如下：

$$skip_1(c)=\begin{cases} \min\{m_k-j \mid p_k[j]=c,\ m_k-minlen\leqslant j\leqslant m_k, 1\leqslant k\leqslant q\} \\ 当c出现在\{p\}中 \\ minlen \\ 当c不出现在\{p\}中 \end{cases}$$

设自动机匹配到状态 s，S 中下一个字符为 c，且 $goto(s, c)<0$，此时已经匹配成功某个模式串 p_i 的最后 m_i-j 个字符，即匹配成功 $p_i[j+1: m_i]$，则 $skip_2(s,c)$定义如下：

$skip_2(s,c)=\min\{d+m_i-j \mid (1\leqslant d\leqslant minlen)$且$(d\geqslant j$ 或 $p_k[j-d]=c)$, for $(j<i\leqslant m), i\leqslant k\leqslant q\}$

作者简介：关 超(1984－)，男，硕士研究生，主研方向：计算机网络与通信；蒋建中，教授；郭军利，副教授
收稿日期：2009-07-30 **E-mail：**mapo1983@163.com

3 改进的多模式匹配算法

进一步提高多模式匹配算法效率的主要途径是利用当前获取的信息进一步增大跳跃距离。综合 BM 算法和 FS 算法的优点，本文对多模式匹配算法做出了改进。它对 BM 的不良字符和良好后缀进行改进，对模式串进行预处理，然后根据当前尝试中匹配失败字符的位置信息，决定是查找好后缀跳越表还是坏字符跳越表，以期获得更大的跳跃距离。算法分为模式的预处理阶段和文本的查找阶段。

3.1 预处理阶段

匹配前的预处理为把模式串集合转换成反向有限自动机，得到 $goto(s,c)$函数和 $output(s)$函数，并计算 $skip_1(c)$，$skip_1(c)$ 的值由 $char$ 在任一模式串中的最右位置决定，$skip_1(c)$ 定义如下：

$$skip_1(c)=\begin{cases}\min\{j+1 \mid p_k[m_k-j]=char, 0\leqslant j\leqslant minlen, 1\leqslant k\leqslant q\}\\ char出现在\{p\}中\\ minlen+1\\ char不出现在\{p\}中\end{cases}$$

$skip_1(c)$ 的值为 $char$ 在模式串中最右出现的位置到该模式串尾的距离加 1，之所以加 1 是因为算法中计算 $skip_1(c)$ 时考虑了文本串对齐模式串右边界右边的一个字符，即计算了 $skip(string[i+1])$，这里，i 是模式串右边界所在位置。因为本次匹配失败后，p 至少移动一个位置，在一般情况下，考虑 $string[i+1]$而不是 $string[i]$，使得最大右移距离是 $minlen+1$，而 BM 算法最大右移距离是 $minlen$。但是 $skip$ 不能超过 $minlen+1$，如果超过 $minlen+1$，最短的模式串就有可能漏过。

3.2 匹配阶段

多模式匹配算法利用匹配过程中匹配失败的信息，跳过一些字符不匹配。因此，为了达到最大的跳跃，设某一次匹配从 $j=i+m-1$, $string[j]$开始，按照 $goto$ 函数向左匹配如果匹配成功，则令 $j=j-1$，匹配下一个 $string[j]$；如果匹配失败，则按照 $skip_1(c)$ 或者 $skip_2$ 向后跳跃，重新开始一次匹配。在匹配过程中，如果已经匹配成功某个模式串，则输出模式串在文本串中的位置，如图 1 所示。

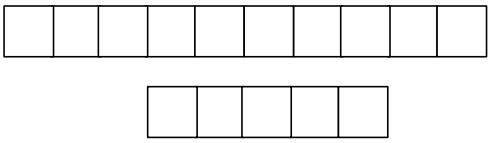

图 1 某次匹配过程

假设模式与文本在文本的 i 处对齐。令 $j=m-1$，用模式字符 $p[j]$与文本中相应位置的字符进行比较。若成功，则 j 递减并继续进行比较；不成功则分别计算：

(1)根据当前文本窗的下一个字符判断可安全跳跃的距离 $skip_1(i+m)$。

(2)寻找文本的后缀和模式中的前缀相匹配部分，并逐渐增大文本后缀的长度，直到找到所有的最大匹配 v_i。采用最大后缀树搜索算法的思想，将所有前缀从短到长排列，然后和文本窗相应后缀比较。由于短前缀是长前缀的一个子串，因此前缀的长度可以逐步增加直到最大长度。在寻找 v_i 的过程中，计算每个模式串相应的跳跃值 $shift_i=m_i-|v_i|$。其中，$|v_i|$表示最大匹配前缀 v_i 的长度；m_i 是模式串 p_i 的长度。在这些跳跃值 $shift_i$ 中，最小的跳跃值 $shift_{\min}$ 就是最大跳跃值 $skip_2$。

比较 $skip_1(c)$ 和 $skip_2$，选择较大值作为最终的跳跃距离。即 $skip=\max\{skip_1(c), skip_2\}$。将文本指针 i 向右移动 $skip$ 距离后，再对模式自右向左进行新一轮的匹配。以上述方式处理文本至文本末尾，可以找出模式的所有出现位置。

3.3 算例分析

在文本串“sregtheyermewherent”中查找模式串集合$\{p\}=\{her, where, redo\}$中的任一模式串。模式串集合转换成的反向有限自动机如图 2 所示，$minlen$=3。图中的箭头及箭头上的字符表示 $goto$ 函数，如果 $goto(s, char)$函数不存在，则指定 $goto(s, char)$=-1；图中双圈的状态表示有输出。$skip_1$如图 3 所示。匹配过程如图 4 所示，每一行表示从状态 0 开始，按照 $goto$ 函数从后向前比较，直到 $goto(state, string[j])\leqslant 0$。竖线内的字符表示当前文本窗。

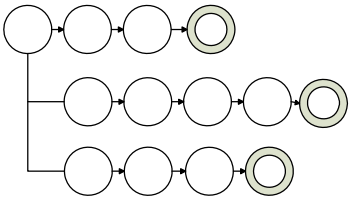

图 2 自动机的输入函数和输出函数

$char$	$skip_1(char)$
d	2
e	1
h	3
o	1
r	1
w	4
其他	4

图 3 $skip_1$ 的构造

字符比较	i	$string[i+1]$	$skip_1$	$shift_1$	$shift_2$	$shift_3$	$skip_2$	$skip$	输出串
|sre|gtheyermewherent	3	g	4	3	5	2	2	4	
sreg|the|yermewherent	7	y	4	1	5	4	1	4	
sregthey|erm|ewherent	11	e	1	3	5	4	3	3	
sregtheyerm|ewh|erent	14	e	1	2	3	4	2	2	
Sregtheyermew|her|ent	16	e	1	0	5	3	0	1	her
Sregtheyermewh|ere|nt	17	n	4	3	0	2	0	4	where

图 4 匹配过程

4 算法分析

4.1 预处理阶段的时间复杂度分析

设字母表的大小为 σ，所有模式串的长度和为 $\sum_{k=1}^{q} m_k$。根据 DFSA 算法，$goto$ 函数值的构造需要的时间与 $\sum_{k=1}^{q} m_k$ 成线形关系，时间复杂度为 $O(\sum_{k=1}^{q} m_k)$。$skip_1$ 的计算分 2 步，第 1 步的计算量与 σ 成线形关系，第 2 步的计算量与 $\sum_{k=1}^{q} m_k$ 成线形关系，$skip_1$ 的复杂度为 $O(\sigma+\sum_{k=1}^{q} m_k)$，由上可知预处理阶段的复杂度为 $O(\sum_{k=1}^{q} m_k)$。

4.2 匹配阶段的时间复杂度

模式串集合中，模式串的最小长度为 $minlen$，最大长度为 $maxlen$。在最优情况下，时间复杂度为 $o(n/(minlen+1))$；在最坏的情况下，时间复杂度为 $o(n\times maxlen)$；在平均情况下，任一字符出现的概率是：$p_c=1/a=1/256$。假设有 k 种模式，所有的模式长度都为 m，在匹配过程中，由于模式集中的每个字符假设是随机的，因此连续出现 t 个字符匹配。一个字符不匹配的概率是

$$p_r(t)=\begin{cases}(1-p_c)(1-(1-p_c^t)^{k(m-t+1)}) & t<m\\ (1-(1-p_c^t)^k) & t=m\end{cases}$$

这样，匹配字符数 t 的期望值为

$$E(t)=\sum_{x=0}^{m-1}x(1-p_c)(1-(1-p_c^x)^{k(m-x+1)})+m(1-(1-p_c^m)^k)$$

出现不匹配后，跳跃的长度为 m-$E(t)$。这样，最大后缀匹配算法对长度为 n 的文本进行匹配的平均比较次数为 $\frac{E(t)+1}{m-E(t)}n$。

5 对比实验及分析

为测试 FS 算法和本文算法的性能，并与 DFSA 算法进行比较，选取了 100 MB 的英文文本，查找多个模式串，找出所有出现的位置。从文本中选取 3 组不同长度的模式串.为测试在不同模式串长度下算法的性能，这 3 组模式串中每组内模式串的长度是一致的。在实验中，每组分别取 l 个~7 个模式串，形成模式串集合进行查找，目的是为了测试在相同模式串长度但不同模式串数目下算法的性能。实验在 P4 2.4 GHz, 512 MB 内存，Win XP 下进行，算法用 C++实现。L 表示模式串长度，查找时间如表 1 所示。

表 1　不同算法的模式串查找时间　　ms

L 值	算法	不同模式串数目 N 下的查找时间						
		N=1	N=2	N=3	N=4	N=5	N=6	N=7
3	DFSA	166	174	180	186	194	203	211
	FS	112	117	124	129	133	145	149
	本文算法	92	103	109	114	121	132	137
8	DFSA	169	183	198	203	211	224	227
	FS	87	94	98	106	112	117	123
	本文算法	68	73	79	84	93	97	102
12	DFSA	173	181	195	207	214	220	225
	FS	63	72	79	84	91	94	99
	本文算法	51	57	66	79	83	87	92

从表 1 可以看出，FS 算法和本文算法的共同特点是在模式串较长、模式串数目较少时，比 DFSA 算法的性能改进比较大；本文算法与 FS 算法相比，在模式较短、模式串数目较小时，性能的改进相对较大。因为当模式串较短、模式串数目较小时，模式串集台中出现的字符较少，在文本串中这些字符出现的概率就小，使得 *skip* 值大的概率增加，本文的算法比 FS 算法性能优越；反之，*skip* 的值小的概率增加，本文算法的优势不明显。在字母表比较大时，如中文字符表，模式串中的字符在文本中出现的概率相对较小，也会使得 *skip* 的值大的概率增加，从而本文算法的优点也就更加明显。

6 结束语

本文把单模式匹配算法中的 BM 算法与基于有限自动机的多模式匹配算法相结合，提出了一个快速的多模式匹配算法。在匹配过程中能够尽可能多地跳过待查文本串字符。该算法的查找时间对模式串数目的增加不敏感。并且在模式串较长的条件下，该算法有相当好的性能。在应用中，可对模式串和目标文本串的特点作进一步的分析，使算法的速度能进一步提高。

参考文献

[1] Lecroq T. Experimental Results on String Matching Algorithm[J]. Software Practice and Experience, 1995, 25(7): 727-765.

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

[3] 蔡晓妍. 一种快速的单模式匹配算法[J]. 计算机应用研究, 2008, 25(1): 45-46.

[4] 许一震. 一种快速的多模式字符串匹配算法[J]. 上海交通大学学报, 2002, 36(4): 516-520.

[5] Cole R. Tight Bounds on the Complexity of the Boyer-Moore String Matching Algorithm[J]. SIAM Journal on Computing, 1994, 23(5): 1075-1091.

编辑　顾逸斐

（上接第 207 页）

定义如下函数：

眼睛对称性函数=左右眼区域最大极值点位置误差绝对值

鼻子位置函数=鼻子区域最大极值点位置-(左眼区域最大极值点位置+右眼区域最大极值点位置)/2

眼睛相似度函数=鼻子位置函数-2×眼睛对称性函数

眼睛识别算法概述：由眼睛垂直位置识别算法，得到最多 2 个可能位置，通过眼睛水平位置识别算法，得到最多 4 组可能的眼睛对位置。对上述眼睛对位置，分别计算眼睛相似度函数，选取相似度最大的组作为识别结果输出。

6 实验图例

Yale 照片库收集了 15 个人，每人 11 种表情光照条件的照片，包括佩戴眼镜、闭眼、张嘴、笑、悲伤、左右方向的侧光等各种变化，总共 165 张。表 1 是几种算子测试的统计数据。

表 1　3 种算法在 Yale 照片库上的统计数据

算法	测试总数	眼睛定位成功数	成功率/(%)
本文算法	165	142	86.1
平均值投影算法	165	105	63.6
方差值投影算法	165	84	50.9

通过以上图例和统计数据表可以看出，本文提出的眼睛定位算法具有较高的成功率。在绝大多数情况下有效。算法原理简单、计算快速，并且有较大的改进空间。

7 结束语

本文提出了一种基于 Haar 小波的眼睛定位方法，用圆差异算子的平均波动值代替常规的投影函数，显著增强了抗光照干扰能力，对多个疑似眼镜点，通过比较左右对称性，筛选出最可能的眼睛点。在 Yale 照片库上的实验证明了该方法的有效性，未来将进一步改进利用人脸的几何结构特点识别出眼睛的方法。

参考文献

[1] Dieckmann U, Plankensteiner P, Wagner T. SESAM: A Biometric Person Identification System Using Sensor Fusion[J]. Pattern Recognition Letters, 1997, 18(9): 827-833.

[2] Feng Guocan, Yuen P C. Variance Projection Function and Its Application to Eye Detection for Human Face Recognition[J]. Pattern Recognition Letters, 1998, 19(9): 899-906.

[3] Zhou Zhihua, Geng Xin. Projection Functions for Eye Detection[J]. Pattern Recognition, 2004, 37(5): 1049-1056.

[4] 章　玲, 蒋建国, 齐美彬. 一种微分与积分投影相结合的眼睛定位方法[J]. 合肥工业大学学报: 自然科学版, 2006, 29(2): 182-185.

[5] 陈雪云, 张振荣, 冷　松. 一种基于圆差异算子的眼睛垂直定位方法[J]. 广西大学学报: 自然科学版, 2008, 33(2): 185-188.

编辑　顾逸斐