分段处理的 1/p 概率字符串匹配*)

吴 玲 秦志光 石竑松 陈兆冲

(电子科技大学计算机科学与工程学院 成都 610051)

摘 要 现有的概率字符串匹配算法通过计算字符串之间的最小失配字符数(编辑距离),可求出字符串之间的相似度。这些算法平等地看待模式串和文本串,虽然可求出二者之间完整的编辑距离,但并不能解决以下问题:即判断是否模式串中至少有 1/p 的字符顺序地出现在文本串中。基于动态规划字符串匹配算法,提出了一个改进算法。该算法通过将字符串分段,在段内执行改进的概率匹配算法可求出段内的编辑距离,再结合回溯策略可以很好地解决上述问题。该算法的复杂性要低于基本动态规划匹配算法,且在某些情况下效率更高。就问题的一般性而言,该算法可广泛地应用于计算生物学、信息安全和信号处理等诸多领域。

关键词 概率字符串匹配,动态规划,分段策略,回溯策略

1/p - Approximate String Matching with Section

WU Ling QIN Zhi-guang SHI Hong-song CHEN Zhao-chong

(School of Computer Science & Engineering, University of Electronic Science and Technology of China, Chengdu 610051, China)

Abstract Most of the existing Approximate String Matching (ASM) algorithms can determine the similarity of strings by computing the minimal number of mismatching characters (edition distance) between them, while the problem that deciding whether at least 1/p of the pattern string falls into the text string with order remains unresolved since these algorithms treat the pattern string and text string equally in computing the full edition distance between them. Based on the dynamic programming string matching algorithm (DP-ASM), an improved algorithm is presented. The algorithm divides the strings into exclusive sections and implements an improved ASM algorithm in every section to compute the sectional edition distance. Combining with a traceback policy, the algorithm can solve this problem as a result. The complexity of the algorithm is comparable with the fundamental ASM algorithm in most cases, while it can get more improvement in some special conditions. As the generality of the problem, this algorithm will find extensive applications in computational biology, information security and signal process fields.

Keywords Approximate string matching, Dynamic programming, Section policy, Trace back policy

1 引言

概率字符串匹配算法(Approximate String Matching)可以对一个包含噪声或遭受某种破坏的文本串执行字符串匹配,允许在匹配过程中出现失配情况。实际应用中有很多将噪声信道上传播的信号还原成初始信号的例子,如在 DNA 序列发生某种突变后找出其子序列,在有打印或拼写错误的文本中进行子串搜索等。由于应用的广泛性,概率匹配技术已大量用于计算生物学、信息安全等领域。本文将考虑一种特殊的概率匹配算法,在叙述我们的目标之前,先对概率匹配算法的相关概念和研究现状进行回顾。

度量两个字符串失配程度的模型有很多,编辑距离(edition distance)(又称为误差)是其中一个很重要的参量,定义为字母表上的一个文本串 $x=x_1x_2\cdots x_n$ 和一个模式串 $y=y_1y_2\cdots y_m$ 之间的最小不同字符数,下文用 $ed(x,y)$ 表示。因此编辑距离实际对应了字符串之间的非匹配情况,有如下三种类型:

(E1)在相同位置上,模式串和文本串对应字符不同。

(E2)在相同位置上,模式串缺少与文本串对应的字符。

(E3)在相同位置上,文本串缺少与模式串对应的字符。

例如,令 x=abcdefg, y=ahcefig,如图 1。由于有 3 个上述误差类型出现,x 和 y 之间的编辑距离等于 3,其中成功发生匹配的一对字符已用垂直线标出。串 x 中的字符'b'对应了 y 中的字符'h'(编辑距离类型(E1)),x 中的'd'在 y 中无对应字符(编辑距离类型(E2)),y 中的'i'在 x 中无对应字符(编辑距离类型(E3))。我们约定,文本串长度用 n 表示,模式串长度用 m 表示,对编辑距离 $k=ed(x,y)$ ($0\leqslant k\leqslant m$),称 $\alpha=k/m$ 为误差率,满足 $0\leqslant\alpha\leqslant 1$。对于一般的字符串匹配算法,如 KMP 或 BM 算法等,其目标是求两字符串是否有 $k=0$ 的匹配情况。

```
x: a b c d e f g
   | | | | |
y: a h c e f i g
```

图 1 x 与 y 之间的编辑距离

最初的概率匹配算法是一种动态规划算法(DP-ASM)。其思想是用一个矩阵 $C_{0\cdots m,0\cdots n}$ 来计算 $ed(x,y)$,其中 $C_{i,j}$ 表示匹配 $x_{1\cdots i}$ 和 $y_{1\cdots j}$ 所需的最小编辑距离。这个矩阵可通过动态规划算法方便计算,由此可计算出两个串之间的最小编辑距离,得出相似度。为了更好地说明本文算法,第 2 节将介绍基本动态规划概率匹配算法的基本思路,它的时间和空间复杂度分别是 $O(mn)$ 和 $O(\min(m,n))$,因此不便于应用。经过后续改进,目前最好的动态规划概率匹配算法是 W. Chang 和 E. Lawer 提出的算法[3],其时间复杂度是 $O(km)$,空间复杂

*)基金项目:国家自然科学基金(No. 60673075),国家高技术研究发展计划(863) (No. 2006AA01Z428)资助。吴 玲 硕士,主要研究方向为网络安全;秦志光 教授,博导;石竑松 博士;陈兆冲 硕士。

度是 $O(m)$。

另外两种概率字符串匹配算法[2]是基于自动机的算法和过滤算法(Filter Algotithm)。前者给出了在最坏情况下的优化算法,其时间复杂度是 $O(n)$。但是,这种算法的时间和空间复杂度与 m 和 k 成指数关系,因此通常无法在实际中应用。过滤算法虽然在实际应用中效率最高,但这种算法通常无法发现匹配发生的文本位置,它必须与另一个过程同时运行,该过程用于验证所有不应被过滤器丢弃的文本位置。重要的是,过滤算法的性能对误差级非常敏感,因此该算法的应用程度也受到了很大限制。

总的来说,目前的字符串概率匹配(ASM)算法可以求出 x,y 之间的最小编辑距离,即判断二者之间的相似性。但是这些算法无法处理这种情况:即求模式串 y"按序"至多有 $k(k<m)$个字符不在文本串 x 中出现的情况。其中"按序"是指字符以在 y 中出现的顺序在 x 中出现。这种匹配可以认为是求模式串整体按序以概率 $p=1-k/m$ 落入文本串的情况,也即求文本串按序是否包含了模式串中至少 $m-k$ 个字符的情况。这和求 x,y 之间的最小编辑距离的目的是不一致的。直觉上,我们可以将模式串中的每一个字符和文本串比较,判断其出现与否来解决这个问题。但是这个算法的时间复杂性是 $O(mn)$,显然不适合应用。我们也可以用正则字符串匹配算法(Regular String Matching)来考察这个问题,但是为了设计正则表达式,RSM 算法在执行前需要先对模式串进行预处理,静态地划分字符串,而在无法预知模式串的哪些部分将在文本串中匹配的情况下,RSM 算法显然是无效的。

通常在 DNA、攻击特征字符串及信号匹配这些理论生物、信息安全和错误检测技术的研究中经常需要这种匹配技术。我们以信息安全的一些应用问题为例,在网络攻击技术中,攻击者经常将攻击特征分成若干片段,嵌入到不同的数据包中以逃避一般的字符串扫描技术。为了检测这种分片攻击,需要重组攻击包,再运用一般的字符串匹配算法检测这个重组后的数据包。该方案的问题在于攻击检测算法无法预知需要重组几个包,以及按何种顺序重组。其次,在垃圾邮件检测系统中,一些垃圾信息往往散落在正常的信件文本中,一般的字符串匹配算法,包括正则匹配、基本的(概率)字符串匹配算法,都可以认为是求解两个字符串之间的最大相似度,因此都会因算法目标的问题而使准确性无法得到保证。如果将这些问题抽象为"求模式串整体按序以概率 $1-k/m$ 落入文本串的情况"(见前文对符号的论述),则可以设计出较好的算法以解决问题。为此,本文在基本 DP-ASM 算法的基础上设计了一个改进算法,其平均时间复杂度大致等于基本 DP-ASM 算法,具体可见本文第 5 节的算法分析,其复杂度与匹配过程中的字符串分段数有关,当分为 $\log m$ 段时,最优的时间复杂度为 $O(m^2/\log m)$,空间复杂度为 $O(m^2/\log^2 m)$,最坏情况下也只与基本 DP-ASM 算法相等,为 $O(mn)$。由于本文算法对 DP-ASM 算法的改进使得并不总是需要进行完整的 DP-ASM 计算过程,因此其复杂性不会超过所使用的 DP-ASM 算法的复杂性。特别地,当字母表 Σ 较大时,如中文信息、或在将单词进行 Σ 上的唯一性编码的某些应用中,由于单个字符出现的概率较小,本文算法对解决这些问题将具有很大的价值。

2 基本动态规划算法

约定以 x_i 表示字符串 x 中的第 i 个字符,$x_{i,\cdots,j}$ 表示 x 中从第 i 个字符到第 j 个字符的一个子串。若模式串 y 与文本串 x 中至 x_j 结束的任意子串之间的最小失配字符数小于 k,就说 y 以至多 k 个误差在 x 的位置 j 出现。这种包含 k 个误差的字符串匹配的目标为:给定长度为 n 的文本串 x,长度为 m 的模式串 y 及整数 $k(k\leqslant m\leqslant n)$,找出 y 以至多 k 个误差出现在 x 中的所有位置。

本质上,DP-ASM 算法就是一个计算模式子串和文本子串之间编辑距离表的过程。其基本思路如下:令 $y_1\cdots y_i$ 和 x 中至 x_j 结束的任意子串之间的编辑距离的最小值为 $D(i,j)$,其中 $0\leqslant i\leqslant m,0\leqslant j\leqslant n$。由动态规划算法,当 $0\leqslant i\leqslant m,0\leqslant j\leqslant n$ 和 $1\leqslant h\leqslant j$ 时,$y_1\cdots y_i$ 与 $x_h\cdots x_j$ 之间的编辑距离递归地计算为:

(1)$y_1\cdots y_{i-1}$ 与 $x_h\cdots x_{j-1}$ 之间的距离 $+y_i$ 与 x_j 之间的距离(y_i 与 x_j 处出现编辑距离类型(E1)),或

(2)$y_1\cdots y_i$ 与 $x_h\cdots x_{j-1}$ 之间的距离 $+1$(x_j 处出现编辑距离类型(E2)),或

(3)$y_1\cdots y_{i-1}$ 与 $x_h\cdots x_j$ 之间的距离 $+1$(y_i 处出现编辑距离类型(E3))。

此后利用动态规划算法通过 $D(i-1,j)$,$D(i,j-1)$ 和 $D(i-1,j-1)$ 算出 $D(i,j)$ 为:

$$D(i,j)=\min(D(i-1,j-1)+\delta_{ij},D(i,j-1)+1,D(i-1,j)+1)$$

其中当 $x_j=y_i$ 时 $\delta_{ij}=0$,否则 $\delta_{ij}=1$。根据编辑距离类型(E3),由于 $y_1\cdots y_i$ 与空文本串的编辑距离为 i,因此对于 $0\leqslant i\leqslant m$,$D(i,0)=i$。通过上述算法可计算出由 $D(i,j)$ 构成的整张 D 表。D 表元素的值反映了模式子串和文本子串之间的最小编辑距离。由此可以找出所有满足失配上限值的文本子串。在 D 表中,称满足条件 $j-i=l$ 的所有元素 $D(i,j)$ 为第 l 条对角线上的元素,它的值反映了当 $l\geqslant 0$ 时文本子串 $x_l\cdots x_{l+m}$ 和整个模式串在匹配过程中编辑距离的变化情况以及当 $l<0$ 时 $x_0\cdots x_{m+l}$ 和 $y_{|l|}\cdots y_m$ 在匹配过程中编辑距离的变化情况。在同一对角线上,连续相同的一串元素说明文本子串与模式串发生了连续匹配,连续相同的元素越多表明匹配越长。若 $D(m,j)\leqslant k$ 且 $j<m$(或 $j\geqslant m$),则文本子串 $x_1\cdots x_j$(或 $x_{j-m}\cdots x_j$)和模式串 y 的编辑距离 $\leqslant k$,从而可找到满足匹配误差上限的所有文本子串。例如表 1 中第 $l=-1$ 条对角线(即从 $D(1,0)$ 至 $D(5,4)$ 的对角线)的值体现了模式串 y ="adbbc"与文本子串"abbd"比较过程中编辑距离的变化情况。而子串 $x_5\cdots x_9$ ="adcbc"就是求出的满足编辑距离 $\leqslant 1$ 的惟一的文本子串。

由于基本动态规划算法通过整个 D 表来计算模式串和文本串的最小编辑距离,其时间和空间复杂度都为 $O(mn)$。为了压缩存储空间,Ukkonen[4] 提出了一种新的方法来构建计算矩阵,仅存储 D 表中的值开始增加的位置。其空间复杂度降为 $O(kn)$,但时间复杂度仍为 $O(mn)$。Ukkonen [5] 在 1985 年又提出了"截断启发式"算法,算法保存并在计算过程中不断更新 D 表上最后一个值不大于 k 的元素,以此摒除不必要的运算,使得在平均状况下的时间复杂度达到 $O(kn)$。随后,Chang 和 Lampe[6] 在 Ukkonen 算法的基础上,提出了"列划分"算法,该算法通过预处理,在每列计算到 $D_{i+1}\neq D_i+1$ 之前即停止计算,使复杂度降低到 $O(kn/\sqrt{\sigma})$,其中 σ 是字母表 Σ 的大小。不过为论述清楚起见,我们还是以基本的动态规划算法为基础说明本文算法。由于改进的动态规划算法都需要计算简化形式的 D 表,实际上都可以应用到我们的算法中。

表1 基本动态规划算法示例

D		0	1	2	3	4	5	6	7	8	9
			a	b	b	d	a	d	c	b	c
0		0	0	0	0	0	0	0	0	0	0
1	a	1	0	1	1	1	0	1	1	1	1
2	d	2	1	1	2	1	1	0	1	2	2
3	b	3	2	1	1	2	2	1	1	1	2
4	b	4	3	2	1	2	3	2	2	1	2
5	c	5	4	3	2	2	3	3	2	2	1

3 算法的基本思路

上述DP-ASM算法可以求出 x,y 之间的最小编辑距离，即判断二者之间的相似性。但是若要求解模式串 y 按序至多有 $k(k<m)$ 个字符不在文本串 x 中出现的情况，则需要对该算法进行改进。其中“按序”是指字符以在 y 中出现的顺序和 x 匹配。因此就将问题转化成求模式串中有多少字符按序出现在文本串中的问题。我们提出了如下算法：

算法基本思想：计算 D 表的某些部分，当计算过程中发现模式子串和文本子串间的连续匹配不能继续时对文本串和模式串分段，并视剩下的两个子串为新的问题实例，采用同样的方法匹配这个实例，直至找出一个满足条件的匹配，或者计算到文本串的结束位置。

算法描述：基本算法由两部分组成，一部分是基于DP-ASM算法改进的在每个分段内执行概率匹配的匹配函数 $match(PS_{round}, TS_{round})$，即整个匹配过程中的一轮匹配进行处理；另一部分是主函数 $split()$，根据 $match$ 函数的返回值对两个字符串进行分段处理，并将新产生的字符串作为新的问题实例，再以该实例作为参数调用匹配函数，以此循环直至确定匹配成功或失败。

```
Procedure match(PS_round, TS_round)
sum ← 0
while sum < (m-PS_round+1) + (n-TS_round+1) do
    i← 0, sum ← 0
    while i ≤ sum do
        j = sum − i
        if  i=0  then  D(i,j) ← 0 end if
        if  j=0  then  D(i,j) ← i end if
        if  (i≠0  and  j≠0)  then
            row ← D(i-1,j) + 1, col ← D(i,j-1) + 1
            if  x_{TSround+j} = y_{PSround+i}  then
                diag ← D(i-1,j-1)
            else  diag ← D(i-1,j-1) + 1  end if
            D(i,j) ← min(row, col, diag)
1.          if  D(i,j) = D(i-1,j-1)  then
2.              MismatchNum ←D(i,j)
3.              PSPos ← PS_round + i
                TSPos ← TS_round + j
                While x_{TSround+j+1} = y_{PSround+i+1}  do
                        D(i+1,j+1) ← D(i,j)
                        i ← i + 1, j ← j + 1
                end while
4.              break while
                end if
            end if
            i ← i + 1
        end while
        sum ← sum + 1
end while
end Procedure

Procedure split()
round ← 1, k_mismatch ← 0, PS_round ← 1, TS_round ← 1, MismatchNum ← m
while ( PS_round ≤ m and TS_round ≤ n ) do
    match (TS_round, TS_round)
    round ←round + 1
5.  k_mismatch ← k_mismatch + MismatchNum
6.  PS_round ← PSPos+ 1
7.  TS_round ← TSPos + 1
    k_sum ←k_mismatch + ( m - PS_round + 1 )
    if k_sum ≤ k then (break while, match succeed) end if
end while
end Procedure
```

图2 基本算法

图2中，sum 表示反对角线的序号，等于DP-ASM算法计算矩阵 D 中行号与列号之和，即 $sum=i+j$。PS_{round} 和 TS_{round} 分别表示参与第 $round$ 轮匹配的模式串和文本串起始字符的位置，$PSPos$ 是返回值所对应的模式串位置，$TSPos$ 是返回值所对应的文本串位置，下一轮匹配开始的文本串位置即为返回值的下一个位置，即 $PS_{round+1}=PSPos+1$，$TS_{round+1}=TSPos+1$（见图2标号6,7处）。k 表示既定的误差上限，$MismatchNum$ 表示当前一轮匹配产生的失配字符数，当前总误差数记为 $k_{mismatch}$，同时用 k_{sum} 表示 $k_{mismatch}$ 与模式串中剩下的还未参与匹配的字符数 $(m-PS_{round}+1)$ 之和。

$match$ 函数计算 D 表时采取从左上至右下的顺序计算反对角线上的值，当找到一个元素 $D(i,j)$ 与其所在对角线上的前一个元素 $D(i-1,j-1)$ 的值相等，则将该元素作为返回的候选元素，更新 $MismatchNum$ 的值，同时计算该对角线上的后一个元素 $D(i+1,j+1)$，此时中止反对角线上的计算。由于 $D(i+1,j+1)$ 的取值只有 $D(i,j)$ 和 $D(i,j)+1$ 两种可能性[1]，同时算法只考虑特定的对角线上是否出现连续匹配的元素，则可规定当找到候选元素后计算特定对角线时，当 $x_j+1=y_i+1$ 时 $D(i+1,j+1)$ 等于 $D(i,j)$，否则等于 $D(i,j)+1$。因此计算 $D(i+1,j+1)$ 的方法可以简化成只判断 x_j+1 与 y_i+1 是否相等（见图2标号1处）。若 $D(i+1,j+1)$ 与 $D(i,j)$ 相等，则继续计算同对角线上的下一个元素的值，否则即认为找到了合适的分段位置，中止对该轮匹配的计算（见图2标号4处），返回到 $split$ 函数，更新 $k_{mismatch}$ 的值，并对字符串进行分段处理，产生新字符串。我们规定 $k_{mismatch}$ 的初值为0，每一轮匹配结束使得 $k_{mismatch}$ 的值增加 $MismatchNum$（见图2标号5处），再将新字符串输入 $match$ 函数开始新一轮的计算，以此循环下去。由于算法只关心两个字符串是否发生满足指定误差的匹配，因此用 k_{sum} 的值来判断匹配成功与否。当 k_{sum} 不大于 k 时即说明匹配成功，算法结束，模式串以不低于 $1-k_{sum}/m$ 的概率在文本串中出现；否则匹配过程将一直进行到两个字符串中任意一个已经匹配到字符串末尾，此时则说明匹配失败。

DP-ASM算法是用一张 D 表计算出所有匹配可能发生的情况，而本算法是将DP-ASM算法作为其中每一轮匹配的基本思想。本算法在平均情况下并不需要像DP-ASM算法那样计算出整张 D 表，当分段发生时，只计算了两个匹配子串之间的部分 D 表。此后只需求解剩余子串之间的部分 D 表，直至下一个分段发生。因此不论是时间复杂度还是空间复杂度本算法都要小于基本动态规划算法。例1描述了基本算法的执行过程。

例1：令文本串 x=“acabkhgjerdxhgt”，长度 $n=15$，模式串 y=“abdxyehgj”，长度 $m=9$，求模式串 y 是否以误差上限 $k\leqslant 4$ 按顺序出现在 x 中。

说明：我们规定 D 表中对角线与反对角线的编号如下。两对角线均从0开始编号，满足 $j-i=k$ 的所有元素 $D(i,j)$ 形成的对角线为第 k 条对角线（即上三角矩阵中的对角线为正，下三角矩阵中的对角线为负）；满足 $i+j=k$ 的所有元素 $D(i,j)$ 形成的对角线为第 k 条反对角线。为表述方便，用 $k^{\#}$ 表示第 k 条对角线（或反对角线）。

第一轮：初始值：$round=1$，$PS_{round}=1$，$TS_{round}=1$，$k_{mismatch}=0$。根据匹配算法部分建立 D 表如表2所示。表2说明，当计算到 $2^{\#}$ 反对角线时，$0^{\#}$ 对角线出现了连续相同的元素0，说明有匹配发生，则将 $D(1,1)$ 作为返回的候选元素，并验证 $0^{\#}$ 对角线上的下一个元素 $D(2,2)$，由于 $D(2,2)$ 的值为1，

中止计算 D 表，返回 $D(1,1)$ 和编辑距离 $MismatchNum=0$。这里 $D(1,1)$ 所对应的模式串字符位置 $PSPos=1$，文本串字符位置 $TSPos=1$。新字符串的起始位置为 $PS_2=PSPos+1=2$，$TS_2=TSPos+1=2$，结束判断标志 $k_{sum}=k_{mismatch}+8=8>k$。

第二轮：初始值：$round=2$，$PS_{round}=2$，$TS_{round}=2$，$k_{mismatch}=0$。计算过程与第一轮相似。根据其返回值 $D(1,3)$ 计算出 $PS_3=3$，$TS_3=5$，$MismatchNum=0$，$k_{sum}=7>k$，并进入下一轮匹配。

第三轮：初始值：$round=3$，$PS_{round}=3$，$TS_{round}=5$，$k_{mismatch}=0$。根据匹配算法部分建立 D 表如表 3 所示。如表 3 所示，当计算到 7# 反对角线时，−3# 对角线出现连续相同的值 4，则计算该对角线下一个元素 $D(6,3)$，$D(6,3)$ 的值也为 4，则继续计算 $D(7,4)$。$D(7,4)$ 的值也相等，应继续计算 −3# 对角线的下一个元素，但此时模式串已匹配完毕，结束本轮匹配，返回主函数，该轮匹配所得 $MismatchNum=4$。主函数根据返回值算出 $k_{sum}=4=k$，因此算法结束，匹配成功，最后找到的匹配文本子序列是“abhgj”。

我们注意到若给定 $k=3$，例 1 的计算结果说明匹配失败，而实际上，文本串 text 中存在子序列“abdxhg”与模式串相比满足 $k_{mismatch}=3$。出现该问题是由于算法在截断字符串后不会再去前面的分段中搜索，而前面的分段有可能在后面出现编辑距离更小的匹配，从而使整体的编辑距离更小，以致满足要求。为此，我们提出了带回溯策略的如下算法。

表 2　例 1 在基本算法第一轮 DP-ASM 匹配的 D 表

D		0	1	2
			a	c
0		0	0	0
1	a	1	0	
2	b			1

表 3　例 1 在基本算法第三轮 DP-ASM 匹配的 D 表

D		0	1	2	3	4	5	6	7
			k	h	g	j	e	r	d
0		0	0	0	0	0	0	0	0
1	d	1	1	1	1	1	1	1	
2	x	2	2	2	2	2	2		
3	y	3	3	3	3	3			
4	e	4	4	4	4				
5	h	5	5	5					
6	g	6			4				
7	j					4			

4　完整的改进算法

第 3 节描述的基本算法不足以解决我们提出的问题，原因在于分段过程可能丢弃了产生更小编辑距离的匹配，为此我们采用一个特定的回溯策略改进这个算法。概率匹配的目标是在文本串中找到一个满足 k 个误差上限的模式串序列。因此一旦出现 $k_{mismatch}>k$ 的情况，我们就立即中止当前搜索并进行回溯，需要解决的问题是应该回溯到哪个位置。

回溯算法基本思想：匹配误差的增加是由于模式串字符的失配，因此当 $k_{mismatch}>k$ 时模式串必须回溯，同时因为最近一轮匹配新产生的失配字符数是导致回溯的直接原因，于是我们对该轮的模式串与不同的文本串重新进行匹配，即回溯到该轮匹配所对应的 PS_{round} 处，在图 3 所示算法中用 PS_{back} 记录可能的回溯位置。对于文本串，我们不进行回溯，而是继续与分段后产生的新文本串进行匹配。此时需更新当前总误差数 $k_{mismatch}$，消去新产生的失配数，回退到最近一轮匹配前的状态，即 $k_{mismatch}$ 的值减少为 $k_{mismatch}-MismatchNum$（见图 3 中标号 3 处）。

```
Procedure split()部分：
    ...(前面部分与基本算法相同)
1.  k_mismatch ← k_mismatch + MismatchNum
2.  if  k_mismatch > k  then
            PS_round ← PS_back
3.          k_mismatch ← k_mismatch - MismatchNum
    else  PS_round ← PSPos + 1
    end if
4.  TS_round ← TSPos + 1
    ...(后面部分与基本算法相同)

Procedure match(PS_round, TS_round) 部分：
    ...(前面部分与基本算法相同)
5.  MismatchNum ← D(i,j)
6.  if MismatchNum ≠ 0 then
            PS_back ← PS_round
    end if
7.  PSPos ← PS_round + i
    ...(后面部分与基本算法相同)
```

图 3　加入了回溯策略的完整算法

图 3 描述了加入了回溯策略的完整算法的改进部分。在基本算法的基础上，增加了一个变量 PS_{back} 用以保存可能的模式串回溯位置。当 $k_{mismatch}>k$ 时将产生回溯（见图 3 中标号 2 处）。对于主函数，图 3 中标号 1,4 分别对应图 2 中标号 5,7，图 3 中标号 2 至 3 之间的部分为改进算法在原算法中插入的部分。匹配函数修改同上，图 3 中标号 5,7 分别对应图 2 中的标号 2,3，标号 6 所引导的部分即为在两者之间新增的代码。

根据该算法，当要求 $k\leqslant 3$ 时，对例 1 的计算过程进行如下修改。当第三轮计算结束后，由于产生了失配，故记录 $PS_{back}=3$。此时 $MismatchNum=4$，$k_{mismatch}=4>k$，立即进行回溯，开始第四轮匹配。

第四轮：初始值：$round=4$，$PS_{round}=PS_{back}=3$，$TS_{round}=9$，$k_{mismatch}$ 更新为 0。根据匹配算法部分建立 D 表如表 4。根据第四轮的返回值 $D(2,4)$ 计算出 $PS_5=5$，$TS_5=13$，$MismatchNum=0$，$k_{mismatch}=0$，$k_{sum}=5>k$，并进入下一轮匹配。

第五轮：初始值：$round=5$，$PS_{round}=5$，$TS_{round}=13$，$k_{mismatch}=0$。根据匹配算法部分建立 D 表如表 5。根据第五轮的返回值 $D(4,2)$ 计算出 $PS_6=9$，$TS_6=15$，$MismatchNum=2$，$k_{sum}=3=k$，算法结束，匹配成功，找到的匹配文本子串是“abdxhg”。

表 4　例 1 在完整算法第四轮 DP-ASM 匹配的 D 表

D		0	1	2	3	4	5
			e	r	d	x	h
0		0	0	0	0	0	0
1	d	1	1	1	0		
2	x	2	2			0	
3	y	3					1
4	e						

表 5　例 1 在完整算法第五轮 DP-ASM 匹配的 D 表

D		0	1	2	3
			h	g	t
0		0	0	0	0
1	y	1	1	1	1
2	e	2	2	2	
3	h	3	2		
4	g			2	
5	j				3

5　算法分析

本文提出了一个对动态规划字符串概率匹配算法的改进算法，其思想是通过计算和操作 D 表，以解决本文提出的问题。通常，DP-ASM 通过计算整个 D 表以找出所有满足给定误差的情况，而本文算法只需要计算部分 D 表，而不需要计算完整的 D 表。

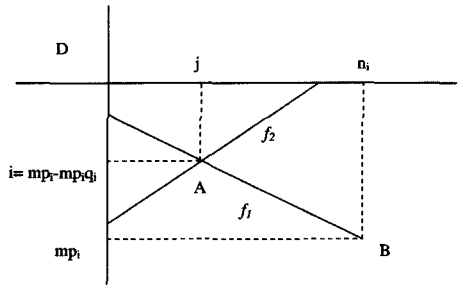

图 4　每一轮匹配过程的计算情况

在基本算法中，分段位置直接影响到算法的效率。对于一个误差上限为 k 的问题实例，假定模式串可分为 s 段，各段长度遵循分布 $D1=(p_1,\cdots,p_s)$，满足 $p_i\geqslant 1/m$，同样文本串也可分为 s(或 $s+1$)段。由算法的分段策略，对应的模式段和文本段满足后部匹配，即第 i 个模式段的后 $l(l\leqslant mp_i)$ 个字符和对应文本段的后 l 个字符应完全匹配，否则将分段。假定匹配字符在模式串中的满足分布 $D2=(q_1,\cdots,q_s)$，满足 $\sum_{i=1}^{s} mp_iq_i\geqslant m-k$ 且 $q_i\geqslant 1/(m-k)$，则 $l=mp_iq_i$。在这种设置下，算法对分段 i 所需要的计算量是 L_i。图 4 显示了算法在每一轮匹配过程中 D 表的计算情况。图 4 中 A 点表示计算 $f_2{}^{\#}$ 反对角线时第一次出现连续相同的元素 $D(i,j),D(i,j)$ 位于 $f_1{}^{\#}$ 对角线上。由此沿着 $f_1{}^{\#}$ 对角线继续计算至 B 点，B 点在 $f_1{}^{\#}$ 对角线上的下一个元素即为编辑距离增加的元素，在 B 点处分段，则 B 点元素即为 $D(mp_i,n_i)$，n_i 表示第 i 轮匹配时文本串的分段位置。由于 A,B 在同一对角线上，则有 $(j-i)=(n_i-mp_i)$，可得 $j=n_i-mp_iq_i$。至 A 点共计算了 $(i+j)$ 条反对角线 $(i+j\leqslant mp_i)$，故其总计算量 $\sum_{i=1}^{s} L_i$ 不大于 $\sum_{i+j=0}^{i+j=mp_i}(i+j)$，则可表示为 $\sum_{i=1}^{s} L_i=mp_iq_i+(1+2+\cdots+mp_i)=mp_iq_i+mp_i(mp_i+1)/2$ 个 $D(i,j)$，因此整个计算过程的总计算量为 $\sum_{i=1}^{s} L_i=m\sum_{i=1}^{s} p_iq_i+\frac{m}{2}\sum_{i=1}^{s} p_i+\frac{m^2}{2}p_i^2$，其上下限范围是 $3m/2-k+m^2/2s\leqslant \sum_{i=1}^{s} L_i\leqslant 3m/2+m^2/2$，即 $O(m^2/s)\leqslant\sum_{i=1}^{s} L_i\leqslant O(m^2)$，其中 L_i 的期望值为 $E(L_i)=(1/s)\sum_{i=1}^{s} L_i$，即每一轮的平均计算量为 $(O(m^2/s^2),O(m^2/s))$。算法所需空间为 $Max_{i=1}^{s}(L_i)$，计算可得 $O(m^2/s^2)\leqslant Max_{i=1}^{s}(L_i)\leqslant O(m^2)$。当分布 $D1$ 和 $D2$ 为均匀分布时，算法的时间复杂度和空间复杂度最低，分别为 $O(m^2/s)$ 和 $O(m^2/s^2)$，此时二维随机变量 $p_i=q_i=1/s$。若令 s 等于 $\log m$，此时算法最优的时间复杂度为 $O(m^2/\log m)$，空间复杂度为 $O(m^2/\log^2 m)$。

对于加入了回溯策略的完整算法，其复杂度相对于基本算法可能会有增加。从算法描述中可以发现，回溯的位置影响到算法效率，该位置是随机的。由于加入了回溯策略后，任一文本子串也只可能和模式子串进行一次比较，即算法只会计算一次它们之间的 D 表，因此最坏情况下的时间和空间复杂度也只等于基本的动态规划算法，为 $O(mn)$。

因此，本文算法在最好的情况下(即无需回溯的情况下)的复杂度为 $O(m^2/\log m)$，在需要多次回溯的情况下其最坏复杂度也不高于基本的动态规划概率匹配算法。形式上，我们的算法是对基本 DP-ASM 算法的改进，由于目前所有基于动态规划的字符串概率匹配算法都需要计算 D 表(或简化的 D 表)，因此和本文算法在底层思想上是一致的，任何加快 D 表计算速度的改进算法都能为我们服务。目前已知最有效的 DP-ASM 算法是计算简化的 D 表，需要 $O(km)$ 时间和 $O(m)$ 空间复杂度，通过这些简化的 D 表也能实现我们的算法策略，因此我们的算法也能达到这个效率，即本文算法能够兼容 DP-ASM 算法效率的进一步发展。

结束语　本文提出了一个改进的字符串概率匹配算法，该算法能求解模式串以不低于 p 的概率按序落入文本串的问题，并能找出这个匹配实例。该算法在最好的情况下(即无需回溯的情况下)的复杂度为 $O(m^2/\log m)$，m 为模式串长度。在需要多次回溯的情况下其最坏复杂度也不高于基本的动态规划概率匹配算法，且能利用最新概率匹配算法的改进结果以改进算法的效率。就问题的一般性来说，本文算法将在计算生物学、信息安全和信号处理等领域有广阔的应用价值。在后续的工作中我们将进一步研究该算法的应用问题，并结合最有效的基于动态规划策略的概率匹配算法提高算法效率。

参 考 文 献

[1] Galil Z, Park K. An improved algorithm for approximate string matching, SIAM J. Comput, 1999, 19(6): 989-999
[2] Navarro G. A guided tour to approximate string matching. ACM Comouting Surveys, 2001, 33(1): 31-88
[3] Chang W, Lawer E. Sublinear approximate string matching and biological applications. Algorithmica, 1994, 12(4): 327-344
[4] Ukkonen E. Algorithms for approximate string matching. Information and Control, 1985, 64: 100-118
[5] Ukkonen E. Finding approximate patterns in strings. Journal of Algorithms, 1985, 6(1): 132-137
[6] Chang W, Lampe J. Theoretical and empirical comparisons of approximate string matching algorithms // Proceedings of the 3d Annual Symposium on Combinatorial Pattern Matching (CPM '92). LNCS, vol. 644, Springer-Verlag, Berlin, 1992: 172-181

(上接第 90 页)

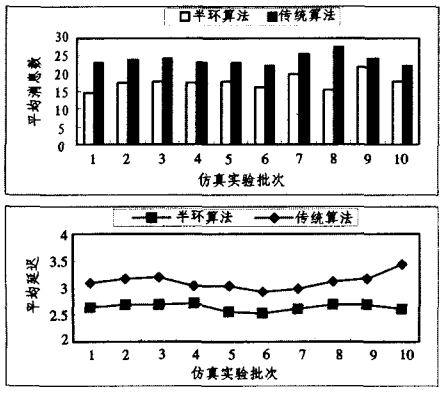

图 2　两种算法性能比较

结束语　本文根据环网的特点，提出了新型的分布式互斥算法 RNDME。该算法基于环网的半环结构生成分布式互斥仲裁集；并且用 Lamport 逻辑时戳保证消息的时序性。分析与仿真证明，和 Maekawa 算法相比，该算法具有较低的消息复杂度、较短的响应延迟和较好的容错性能。

参 考 文 献

[1] 舒继武，等. 大规模问题数据并行性能的分析[J]. 软件学报，2000, 11(5): 628-633
[2] 黄铠，徐志伟，著. 可扩展并行计算技术、结构与编程[M]. 陆鑫达，等译. 北京：机器工业出版社，2000: 145-223
[3] 尹俊文，邹鹏，等. 分布式操作系统[M]. 长沙：国防科技大学出版社，2001: 68-82
[4] Maekawa M. A logN Algorithm for Mutual Exclusion in Decentralized Systems [J]. ACM Trans. Computer Systems, 1985, 3(2): 145-159
[5] 刘丹，刘心松. 基于读写特征的分布式互斥算法 [J]. 电子学报，2004, 32(2): 326-329
[6] Fu A W. Delay-optimal Quorum Consensus for Distributed Systems[J]. IEEE Transactions on Parallel and Distributed Systems, 1997, 1(8): 59-69