

基于通配符和长度约束的近似模式匹配算法

黄国林*, 郭 丹, 胡学钢

(合肥工业大学 计算机与信息学院, 合肥 230009)

(*通信作者电子邮箱 huangguolin0115@126.com)

摘 要:针对近似模式匹配算法在处理带有灵活通配符和长度约束近似模式匹配(APMWL)问题时只能解决替换操作,提出一种基于动态规划的编辑距离矩阵(EDM)构造方法,设计了基于EDM的近似模式匹配算法APM,可以处理近似匹配中的三种编辑操作,即插入、替换和删除操作。此外,根据文本中字符是否允许被重复使用的约束条件,设计APM-OF算法。实验结果表明,APM和APM-OF与同类算法相比具备显著的优势:与Sail_Approx匹配算法实验对比,获取解的平均增长率分别达到8.34%和12.37%;将APM-OF算法应用至模式挖掘中,挖掘出的频繁近似模式个数为OneoffMining算法的2.07倍。

关键词:近似匹配;通配符;长度约束;编辑距离矩阵;one-off条件

中图分类号:TP391.4 **文献标志码:**A

Algorithms for approximate pattern matching with wildcards and length constraints

HUANG Guolin*, GUO Dan, HU Xuegang

(School of Computer and Information, Hefei University of Technology, Hefei Anhui 230009, China)

Abstract: Current works on the Approximate Pattern Matching with Wildcards and Length constraints (APMWL) problem can only cope with replacement operation. This paper proposed an Edit Distance Matrix (EDM) method based on dynamic programming and the Approximate Pattern Matching with EDM (APM) algorithm. APM can handle all approximate operations including insertion, replacement and deletion. Moreover, this paper extended APM to the APM-OF algorithm with a strict constraint condition that each character can be used at most once for pattern matching in a sequence. The experiments verify that both APM and APM-OF have significant advantages on matching solutions against other peers. The average improvement rates of matching compared to SAIL-Approx are up to 8.34% and 12.37% respectively. It also demonstrates an advantage on approximate pattern mining that the number of approximate patterns mined by APM-OF is 2.07 times of that mined by OneoffMining.

Key words: approximate pattern matching; wildcard; length constraint; edit distance matrix; one-off condition

0 引言

近似模式匹配是精确模式匹配的泛化,允许找到的出现与给定模式之间存在一定程度的差异,相比精确模式匹配问题更具灵活性和挑战性,在生物DNA序列分析^[1-2]、文本检索^[3-4]、网络入侵检测^[5-6]等众多领域具有广泛应用。

传统的近似模式匹配算法^[7-9]只能处理简单的字符组合模式,形式过于单一,很大程度上局限了人们的查询需求。1974年, Ficher等^[10]首次将通配符(记为 Φ)的概念引入模式匹配中,增强了问题灵活性,但固定了模式字符间通配符的数量。2006年, Chen等^[11]进一步提出灵活通配符的概念,用户可自由设置相邻字符之间通配符的长度范围,使模式极具弹性,但文中给出的算法只能处理精确模式匹配。2007年, He等^[12]在此基础上提出Sail_Approx算法,该算法可以处理含灵活通配符的在线近似模式匹配问题,但它只能单一地解决近似匹配中的替换操作,并不能处理插入和删除操作。2009年, Huang等^[13]提出OneoffMining算法,该算法可以解决带灵活通配符和长度约束的模式挖掘问题,但无法解决近似模式挖掘问题。

本文针对带灵活通配符和长度约束的近似模式匹配

(Approximate Pattern Matching with Wildcards and Length constraints, APMWL)问题进行研究,设计了APM (Approximate Pattern Matching with EDM)和APM-OF (Approximate Pattern Matching with EDM under One-off Condition)算法,能有效处理带通配符近似模式匹配中的插入、替换和删除三种操作,同时用户能够自行设置各间隙长度的上、下限值及解的长度范围,在与同类算法的实验对比中优势明显。

1 问题定义

本章对APMWL问题的定义进行形式化描述,包括模式局部长度约束条件(Local-Length Constraints)的定义以及文本字符一次性使用原则(One-off Condition)的定义等,并通过示例说明相关定义和概念。

定义1 给定一个模式串

$$P = p_0 \varphi_{N_0}^{M_0} p_1 \cdots \varphi_{N_{i-1}}^{M_{i-1}} p_i \varphi_{N_i}^{M_i} p_{i+1} \cdots \varphi_{N_{m-2}}^{M_{m-2}} p_{m-1}$$

和一个文本串 $T = t_1 \cdots t_n$ 以及模式 P 的最小长度约束 G_N 和最大长度约束 G_M , 其中 $p_i (0 \leq i < m)$ 由单个字符表示, m 和 n 分别为 P 中字符 (不包括通配符) 的个数和文本串的长度, Φ 为通配符, 可以匹配任意给定的一个字符。 $[N_i, M_i]$ 表示

收稿日期:2012-09-04;修回日期:2012-11-07。 基金项目:国家863计划项目(2012AA011005);国家自然科学基金资助项目(61229301);国家博士后科学基金资助项目(2012M511403);中央高校基本科研基金资助项目(2010HGXJ0714)。

作者简介:黄国林(1988-),男,甘肃白银人,硕士研究生,主要研究方向:数据挖掘;郭丹(1983-),女,湖北潜江人,讲师,博士,主要研究方向:机器学习、数据挖掘;胡学钢(1961-),男,安徽当涂人,教授,博士生导师,主要研究方向:知识工程、数据挖掘。

p_i 与 p_{i+1} ($0 \leq i < m-1$) 间的通配符间隔, 称为局部长度约束, $[G_N, G_M]$ 为全局长度约束。令 $g = \max\{M_i - N_i\}$, 表示模式 P 中任意连续两字符间通配符的最大跨度。

定义2 对于插入一个字符、删除一个字符, 或替换一个字符的行为, 称为一个编辑操作。给定字符串 P_1 和 P_2 , 将 P_1 转换为 P_2 (或将 P_2 转换为 P_1) 所需的最小编辑操作次数称为它们之间的编辑距离, 记为 $Edit(P_1, P_2)$ 。例如: $Edit(\text{annual}, \text{annealing}) = 4$ 。

定义3 若在文本 T 中存在一个位置序列 $L = \langle j_{h_0}, j_{h_1}, \dots, j_{h_s} \rangle$, 其中 $j_{h_i} < j_{h_{i+1}}, 0 \leq h_0 < \dots < h_i < \dots < h_s, k$ 为编辑误差, 若该位置序列满足如下三个条件:

1) 局部约束。

j_{h_k} 表示 p_{h_k} 在文本 j_{h_k} 处对应出现, 同理 $j_{h_{k+1}}$ 表示 $p_{h_{k+1}}$ 在文本 $j_{h_{k+1}}$ 处对应出现, 则子模式 $p_{h_{k+1}}$ 与 p_{h_k} 之间需满足局部长度约束:

$$\sum_{h_k \leq g \leq h_{k+1}} N_g \leq j_{h_{k+1}} - j_{h_k} - 1 \leq \sum_{h_k \leq g \leq h_{k+1}} M_g$$

2) 全局约束。

$$G_N \leq j_{h_s} - j_{h_0} - 1 \leq G_M$$

3) 编辑距离。

$Edit(P, T) \leq k, T' = \{t_{j_{h_0}}, t_{j_{h_0+1}}, \dots, t_{j_{h_s}}\}$ 为 T 中子文本, 则称该位置序列 $\langle j_{h_0}, j_{h_1}, \dots, j_{h_s} \rangle$ 为模式 P 在文本 T 的一个近似出现, 记为 A_Occ , 文本子串 $t_{j_{h_0}} t_{j_{h_0+1}} \dots t_{j_{h_s}}$ 为模式 P 在文本 T 中的一个近似匹配。

定义4 为了去除冗余信息, 本文引入了文本字符匹配的约束条件——“一次性条件”(One-off 条件), 即文本中字符至多只能被一次匹配。若近似出现 $A_Occ = \langle a_0, \dots, a_r \rangle$, $\dots, a_r \rangle, A_Occ' = \langle b_0, \dots, b_j, \dots, b_s \rangle$, 对任意的 a_i 和 b_i , 都满足 $a_i \neq b_j$, 则称它们满足 One-off 条件。

例1 若模式串 $P = a\varphi_1^2 c\varphi_0^2 t$, 文本串 $T = \text{aacctt}$, 全局长度约束为 $[4, 5]$, 编辑误差 $k = 1$, 则如图1所示, $\langle 1, 3, 5 \rangle, \langle 2, 4, 6 \rangle$ 和 $\langle 1, 4 \rangle$ 均为满足条件的近似出现, 它们与模式 P 的编辑距离分别为 $0, 0, 1$ 。由于 $\langle 1, 3, 5 \rangle$ 和 $\langle 1, 4 \rangle$ 共享字符 $t_1, \langle 2, 4, 6 \rangle$ 和 $\langle 1, 4 \rangle$ 共享字符 t_4 , 在 One-off 条件下, 文本中字符只能被一次使用, 故在共享字符位置处需进行筛选, $\langle 1, 3, 5 \rangle$ 和 $\langle 2, 4, 6 \rangle$ 即为 One-off 条件下的一组近似出现。

A_Occ	T						$Edit(P, A_Occ)$
	1	2	3	4	5	6	
A_Occ1	a	a	c	c	t	t	0
A_Occ2							0
A_Occ3							1

图1 模式 P 在文本 T 中的出现(例1)

2 算法设计

本章首先提出基于动态规划编辑距离矩阵 $D_{[0, \dots, m-1, 0, \dots, n]}$ 的编辑距离矩阵(Edit Distance Matrix, EDM)构造方法, 其中 $D[i][j]$ 为模式 P 的子串 $p_0 \varphi_{N_0}^M p_1 \dots \varphi_{N_{i-1}}^M p_i$ 与文本 T 中以 t_j 结尾的任意子串间的最小编辑距离。其次, 设计了 APM 算法, 该算法可以解决 APMWL 问题中的三种编辑操作, 即插入、替换和删除操作。最后, 为满足 One-off 约束条件, 将 APM 算法进一步扩展为 APM-OF 算法。

2.1 EDM 构造方法

EDM 构造方法采用贪心算法的思想, 通过查找模式子串的优化匹配, 递归地得到整个模式串的优化匹配。构造编辑距离矩阵 D 时, 主要分为以下两个步骤:

1) 元素初始化。 D 中首行元素表示 p_0 与 t_j 间的编辑距离, 即 $D[0][j] = Edit(p_0, t_j), 1 \leq j \leq n$; 首列元素表示模式 P 的子串与空字符串间的编辑距离, 即 $D[0][0] = 1, D[i][0] = D[i-1][0] + N_{i-1} + 1$, 其中 $1 \leq i \leq m-1$ 。

2) 计算 D 中其他元素 $D[i][j] (1 \leq i \leq m-1, 1 \leq j \leq n)$, 见式(1)。

$$\omega = (j-1 \geq start) ? D[i][j-1] : D[i][0]$$

$$D[i][j] =$$

$$\begin{cases} \min\{D[i-1][s] + Edit(p_i, t_j), \omega + 1, \\ D[i-1][j] + N_{i-1} + 1\}, \\ \max\{j - M_{i-1} - 1, start\} \leq s \leq j - N_{i-1} - 1 \\ \min\{D[i-1][0] + N_{i-1} + Edit(p_i, t_j), \omega + 1, \\ D[i-1][j] + N_{i-1} + 1\}, \\ j - N_{i-1} - 1 < start \end{cases} \quad (1)$$

其中: m 为模式 P 中字符(非通配符)的个数, n 为文本 T 的长度, M_{i-1} 和 N_{i-1} 分别为 p_{i-1} 与 p_i 间通配符的上、下限值, s 为满足间隔约束且与 p_{i-1} 对应的文本字符位置, $start = j - l - k + 1$ 为动态规划的开始位置, l 为全局长度最大约束, 当 $j-1 < start$ 时, 令 $j-1 = 0$ 。

由式(1)可看出, $D[i][j]$ 的值由以下三部分共同决定, 选取其中最小值作为最优编辑距离: ① 替换操作的编辑代价, 即 $D[i-1][s] + Edit(p_i, t_j)$ 或 $D[i-1][s] + N_{i-1} + Edit(p_i, t_j)$; ② 插入操作的编辑代价, 即 $D[i][j-1] + 1$; ③ 删除操作的编辑代价, 即 $D[i-1][j] + N_{i-1} + 1$ 。

例2 给定模式 $P = a\varphi_0^2 c\varphi_1^3 g\varphi_2^2 t$, 文本 $T = \text{agactgt}$, 编辑误差 $k = 1$, 则它们之间的编辑距离矩阵如图2所示。

		T						
		a	g	c	a	t	g	t
P	a	1	0	1	1	0	1	1
	c	2	1	1	0	1	1	1
	g	4	3	3	2	2	<1>	0
	t	5	4	4	3	3	2	1
	t	5	4	4	3	3	2	1

图2 模式 P 与文本 T 的编辑距离矩阵(例2)

1) 替换操作。

文本区间 $[j - M_{i-1} - 1, j - N_{i-1} - 1]$ 满足 p_i 与 p_{i-1} 局部长度约束条件, 称为 $D[i][j]$ 的替换区间, 该区间内的有效元素所构成的集合, 即 $\{D[i-1][s] \mid s \in [j - M_{i-1} - 1, j - N_{i-1} - 1] \&\& s \geq start\}$, 称为 $D[i][j]$ 的替换集合。

① 当 $D[i][j]$ 的替换集合非空时, 表明在 t_s 处存在满足局部长度约束且与模式子串 $p_0 \varphi_{N_0}^M p_1 \dots \varphi_{N_{i-2}}^M p_{i-1}$ 匹配的文本子串, 故 $D[i][j] = \min\{D[i-1][s] + Edit(p_i, t_j), s \in [j - M_{i-1} - 1, j - N_{i-1} - 1] \&\& s \geq start\}$;

② 当 $D[i][j]$ 的替换集合为空时, 表明模式子串 $p_0 \varphi_{N_0}^M p_1 \dots \varphi_{N_{i-2}}^M p_{i-1}$ 只能与空字符串匹配, $D[i][j]$ 的替换元素为 $D[i-1][0]$ 。在匹配字符 p_i 时, 需要留出至少 N_{i-1} 个字符以满足 p_{i-1} 与 p_i 间的局部长度约束, 故此时 $D[i][j] = D[i-1][0] + N_{i-1} + Edit(p_i, t_j)$ 。

如例2中 $D[2][6]$ 所示, $D[2][6]$ 的替换集合为 $\{D[1][2], D[1][3], D[1][4]\}$ (见图2中虚线框所示), 最小值为 $D[1][3] = 0$, 故由替换操作得到 $D[2][6]$ 的值为 $D[1][3] + Edit(p_2, t_6) = 0$ 。

2) 插入操作。

$D[i][j]$ 对应的插入元素为 $D[i][j-1]$, 插入操作也分为两种情形: ① 当 $j-1 < start$ 时, 表明匹配在模式子串

$p_0\varphi_{N_0}^{M_0}p_1\cdots\varphi_{N_{i-2}}^{M_{i-2}}p_{i-1}$ 与空字符串匹配的基础上, 在空字符串后插入字符 t_j 得到, 故 $D[i][j] = D[i][0] + 1$; ② 当 $j - 1 \geq start$ 时, 表明存在插入元素 $D[i][j - 1]$, 匹配可在文本子串 $SubText_{i,j-1}$ 后插入字符 t_j 得到, 其中 $SubText_{i,j-1}$ 表示以 t_{j-1} 结尾且与模式子串 $p_0\varphi_{N_0}^{M_0}p_1\cdots\varphi_{N_{i-1}}^{M_{i-1}}p_i$ 满足最小编辑距离的文本子串, 则有 $D[i][j] = D[i][j - 1] + 1$ 。

例2中 $D[2][6]$ 的插入元素为 $D[2][5]$ (见图2中间括号内元素), 开始位置 $start = 1$, 有插入代价为 $D[2][6] = D[2][5] + 1 = 2$ 。

3) 删除操作。

$D[i][j]$ 对应的删除元素为 $D[i - 1][j]$, 若 $D[i][j]$ 是在 $p_0\varphi_{N_0}^{M_0}p_1\cdots\varphi_{N_{i-2}}^{M_{i-2}}p_{i-1}$ 与 $SubText_{i-1,j}$ 匹配的基础上删除字符 p_i 得到的, 其中 $SubText_{i-1,j}$ 为以 t_j 结尾且与模式子串 $p_0\varphi_{N_0}^{M_0}p_1\cdots\varphi_{N_{i-2}}^{M_{i-2}}p_{i-1}$ 满足最小编辑距离的文本子串。为满足 p_{i-1} 与 p_i 间的局部长度约束下限 N_{i-1} , 删除操作的计算公式为 $D[i][j] = D[i - 1][j] + N_{i-1} + 1$ 。

例2中 $D[2][6]$ 的删除元素为 $D[1][6]$ (见图2中方括号), 删除代价为 $D[2][6] = D[1][6] + N_1 + 1 = 3$ 。

2.2 APM 算法

基于 EDM 构造方法, 本章设计了 APM 算法, 伪码如算法1所示。

1) 寻找回溯位置(即可能存在解的位置) trigger position。自左向右扫描文本 T 的同时采用 EDM 方法构造编辑距离矩阵 D , 伪码如算法2所示。若存在 $D[m - 1][j]$ 小于等于编辑误差 k , 则表明在 t_j 处存在一个 trigger position。

2) 正向验证, 判定在 trigger position 处是否真正存在解。由于全局长度约束条件的引入, 找到的 trigger position 并不一定是存在解的位置。在编辑距离矩阵中截取一个以 trigger position 为终点、满足全局长度约束的窗口, 利用 EDM 方法重新计算该矩阵中的元素值, 若经计算仍有 $D[m - 1][j] \leq k$, 则表明在 t_j 处确实存在满足条件的匹配, 反之则不存在。

3) 若经过2)的正向验证, 在 trigger position 处不存在解, 则返回1)继续寻找其他的回溯位置, 直到扫描至文本的结尾; 否则输出满足条件的匹配解。在确定匹配各个位置时, 首先需要判定当前元素值是经哪种操作得到, 然后输出对应的匹配位置。

算法1 APM。

输入 P, T, k 。

输出 A_Occ 。

```

1)  $j \leftarrow 1$ ; //  $j$  用于扫描文本
2) while  $t_j \neq \text{NULL}$  do
3)  $D_{[0,\dots,m-1;0,\dots,n]} = \text{EDM}(j, 0, 0, D)$ ;
   // 计算第  $j$  列编辑距离矩阵的值
4) if  $D_{[m-1,j]} \leq k$ 
   // 若  $j$  为 trigger position, 则进一步处理
5)  $end = j$ ;  $start = j - l + k + 1$ ;
   //  $l$  为全局长度约束上限
6)  $D_{[0,\dots,m-1;0,\dots,n]} = \text{EDM}(j, start, end, D)$ ;
   // 更新矩阵自  $start$  到  $end$  的元素值, 进行正向验证
7) if  $end > 1$  &&  $D_{[m-1][end]} \leq k$ 
   // 若经正向验证, 存在满足条件的匹配
8) then Output( $end$ );
   // 返回终点位置为  $end$  的匹配
9)  $j++$ ;
10) end while

```

算法2 EDM。

输入 $j, start, end, D_{[0,\dots,m-1;0,\dots,n]}$ 。

输出 $D_{[0,\dots,m-1;0,\dots,n]}$ 。

```

1)  $D[0][j] \leftarrow \text{Edit}(p_0, t_j)$ ;
2) for  $i \leftarrow 0$  to  $m - 1$  do
3)   if  $j - l_{\min} - 1 < start$  then
       //  $l_{\min}$  为  $p_{i-1}$  与  $p_i$  间的通配符下限
4)    $R_{\min} \leftarrow D[i][0] + l_{\min}$ ;
       //  $R_{\min}$  为替换操作的编辑代价
5)   else for  $s \leftarrow j - l_{\max} - 1$  to  $j - l_{\min} - 1$  do
       //  $l_{\max}$  为  $p_{i-1}$  与  $p_i$  间的通配符上限
6)   if  $D[i - 1][s] < R_{\min}$  then
        $R_{\min} \leftarrow D[i - 1][s]$ ;
7)   if  $j - 1 \geq start$  then  $I_{\min} \leftarrow D[i][j - 1] + 1$ ;
       //  $I_{\min}$  为插入操作的编辑代价
8)   else  $I_{\min} \leftarrow D[i][0] + 1$ ;
9)    $D_{\min} \leftarrow D[i][j - 1] + 1$ ;
       //  $D_{\min}$  为删除操作的编辑代价
10)   $D[i][j] \leftarrow \min\{R_{\min}, I_{\min}, D_{\min}\}$ ;
11) end for

```

2.3 APM-OF 算法

在实际应用和理论研究中, One-off 条件都具有重要的价值。例如, 用户在一篇文档 T 中搜索模式 $P = \text{英语 } \varphi_0^4$ 四级, 当计算模式 P 在文档 T 中的出现次数时, 采用 One-off 条件限制显然更加合理。因此, 为满足 One-off 条件约束, 本节对 APM 算法进行扩展, 设计了 APM-OF 算法。

APM-OF 算法也分为三个步骤: 1) 寻找 trigger position。2) 进行正向验证, 判断是否存在解。3) 若存在解, 则回溯输出; 否则, 继续寻找回溯位置。该算法同 APM 算法相比, 主要有两个区别:

1) 在寻找回溯位置时, 在算法1中代码行3)和6)处构造编辑距离矩阵时, 需判定文本字符是否被匹配的 $used[]$ 数组, 若文本字符 t_j 已被匹配, 则 $used[j] = \text{true}$, $D[i][j] = \infty$, 从而消除在 One-off 条件下文本中已匹配字符对 $D[i][j]$ 的影响。

2) 在回溯输出满足条件的出现(算法1中代码7))后, 对文本中匹配过的字符添加一次性使用标记(One-off 标记 $used[j] = \text{true}$), 避免重复使用。由于元素可能出现的位置有多个, 本文采用 Left-Most 策略(即最左最优, 当满足条件的位置有多个时, 选用最左端的位置), 从而尽可能寻找更多的匹配。

2.4 运行实例

例3 给定模式 $P = a\varphi_0^3e\varphi_0^2f\varphi_0^3b\varphi_0^3g$, 文本 $T = \text{aaecfehbb}$, 全局长度约束为 $[6, 12]$, 编辑误差 $k = 1$ 。

1) APM 算法构建的编辑距离矩阵 D 如图3所示。在 t_8 处, APM 构造矩阵 $D_{[0,\dots,4;0,\dots,9]}$, 找到的近似出现为 $\langle 1, 3, 5, 8 \rangle$, 见图3箭头所示。随后, 在 t_9 处, EDM 构造矩阵 $D_{[0,\dots,4;0,\dots,12]}$, 满足全局长度约束, 在 t_{12} 处输出满足条件的近似出现 $\langle 1, 3, 5, 9 \rangle$, 故 APM 输出解为 $\{\langle 1, 3, 5, 8 \rangle, \langle 1, 3, 5, 9 \rangle\}$ 。

2) APM-OF 算法构建的编辑距离矩阵 D 。在 t_8 处, APM-OF 构造矩阵 $D_{[0,\dots,4;0,\dots,9]}$ 如图3所示, 找到的近似出现为 $\langle 1, 3, 5, 8 \rangle$ 。随后, 对 t_1, t_5, t_9, t_{11} 处进行 One-off 标记令这些位置处对应的编辑距离矩阵元素值为 ∞ , 从而消除它们对其他元素值的影响。在 t_9 处, 构造的编辑矩阵 $D_{[0,\dots,4;0,\dots,9]}$ 如图

4所示。此时,在 t_9 处的最小编辑距离为2,故不存在满足条件的匹配,APM-OF输出解为 $\{ \langle 1,3,5,8 \rangle \}$ 。

		T								
		t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	t_9
		a	a	e	e	f	e	h	b	b
P	p_0 a	1	1	0	1	1	1	1	1	1
	p_1 e	2	1	1	1	0	1	0	1	2
	p_2 f	4	3	3	2	2	1	1	1	1
	p_3 b	5	4	4	3	3	1	1	1	0
	p_4 g	6	5	5	4	4	2	2	1	1

图3 模式P与文本T的编辑距离矩阵(例3,APM)

		T								
		t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	t_9
		a	a	e	e	f	e	h	b	b
P	p_0 a	1	∞	0	∞	1	∞	1	1	∞
	p_1 e	2	∞	1	∞	0	∞	0	1	∞
	p_2 f	4	∞	3	∞	2	∞	1	2	∞
	p_3 b	5	∞	4	∞	3	∞	2	2	∞
	p_4 g	6	∞	5	∞	4	∞	3	3	∞

图4 模式P与文本T的编辑距离矩阵(例3,APM-OF)

3 算法复杂度

时间复杂度:APM和APM-OF时间复杂度均为 $O(gmn + clgm)$ 。其中: g 为模式P中任意两个连续字符间允许的最大间隔, m 为模式P中字符(非通配符)个数, n 为文本长度, l 为全局最大长度, c 为回溯位置的个数。

空间复杂度:APM和APM-OF算法的空间复杂度为 $O(lm)$ 。

4 实验结果与分析

测试序列来自2010年3月30日公布的猪流感H1N1病毒中的全部8个DNA片段(从<http://www.ncbi.nlm.nih.gov/genomes/FLU/SwineFlu.html>下载),选用的模式见表1。

实验运行的软硬件环境为:酷睿2双核E7500主频2.93 GHz,内存1.96 GB,Windows XP SP2操作系统。目前,处理带灵活长度约束通配符的代表性算法有Sail_Approx。为了便于比较,根据文本中字符是否一次性使用原则,本文将Sail_Approx算法^[12]拓展为SAW(Sail_Approx Without the One-off Condition)算法和SA-OF(Sail_Approx Under the One-off Condition)算法。为便于分析APM和APM-OF算法的性能,引入解的增长率的概念,记为:

$$\delta = \frac{S_{\text{APM/APM-OF}} - S_{\text{SAW/SA-OF}}}{S_{\text{SAW/SA-OF}}}$$

其中: $S_{\text{APM/APM-OF}}$ 表示APM或APM-OF算法找到的匹配数量, $S_{\text{SAW/SA-OF}}$ 表示SAW或SA-OF算法找到的匹配数量。

表1 人工构造模式串

序号	模式	最小长度	最大长度
p_1	c[0,5]t[0,5]g[0,5]a	4	19
p_2	a[0,5]t[0,7]c[0,9]g	4	25
p_3	a[0,3]c[0,3]a[0,3]t	4	13
p_4	a[0,3]c[0,4]g[0,4]t[0,5]g	5	23
p_5	g[0,3]c[0,2]a[0,2]t[0,3]c[0,2]a	6	18
p_6	t[0,2]a[0,3]c[0,4]t[0,5]a[0,6]c	6	26
p_7	a[0,9]t[0,9]c[0,9]g[0,9] c[0,9]t[0,9]a[0,9]g[0,9]c	11	41
p_8	g[1,5]t[1,5]c[1,5]a[1,5]t	9	25

4.1 实验1:时间性能比较

如表2~3所示,APM算法和SAW算法、APM-OF算法 and SA-OF算法在时间运行结果上与算法时间复杂度分析一致,均为 $O(clgm + gmn)$,属于同一个数量级。其中 $S_i(i=1, 2, \dots, 8)$ 表示8个H1N1的DNA序列。

表2 SAW和APM算法平均运行时间对比 ms

算法	S_1	S_2	S_3	S_4	S_5	S_6	S_7	S_8
SAW	31	31	31	31	31	31	16	16
APM	31	31	31	31	31	31	16	16

表3 SA-OF和APM-OF算法平均运行时间对比 ms

算法	S_1	S_2	S_3	S_4	S_5	S_6	S_7	S_8
SA-OF	31	31	31	31	31	31	16	16
APM-OF	31	31	31	32	31	31	16	16

4.2 实验2:匹配数量比较

设编辑误差 $k=1$,如表4所示,APM算法全部占有优势,且解的平均增长率为8.34%,64组比较中55组APM-OF算法占有优势,解的平均增长率达12.37%。

表4中:

δ_{APM} 表示APM算法相比SAW算法在匹配数量上的增长率:

$$\delta_{\text{APM}} = \frac{S_{\text{APM}} - S_{\text{SAW}}}{S_{\text{SAW}}} \times 100\%$$

$\delta_{\text{APM-OF}}$ 表示APM-OF算法相比SA-OF算法在匹配数量上的增长率:

$$\delta_{\text{APM-OF}} = \frac{S_{\text{APM-OF}} - S_{\text{SA-OF}}}{S_{\text{SA-OF}}} \times 100\%$$

4.3 实验3:参数分析

本节针对影响匹配数的三个主要参数进行实验:编辑误差 k ,模式P中字符(非通配符)的个数 m 以及局部长度约束下限 $N_i(0 \leq i < m-1)$,记APM和APM-OF的增长率分别为 δ_{APM} 和 $\delta_{\text{APM-OF}}$ 。实验结果表明:当编辑误差 k 值较大,如图5(a)中 $k=3$ 时 $\delta_{\text{APM-OF}}$ 达到18.78%;当 N_i 很大时,如图5(c)中 $N=3$ 时 δ_{APM} 达到31.43%。

1) 特征1:编辑误差 k 。拟定 $m=8$,每个 k 取值上随机采用10组模式进行测试。当 $k=0$ 时,问题退变为精确匹配求解,故算法找到的解的数目一致。

如图5(a)所示,APM解的增长率 δ 呈先增后降的趋势:APM可处理三种操作,相对于SAW更加灵活,故随着 k 值的增大 δ 不断增大。当 k 值增大到一定程度后,APM中的插入或删除操作逐渐能被替换操作取代,APM趋向于和SAW求解一致,故 δ 会有所下降。APM-OF解的增长率 δ 持续呈单调递增的趋势:因为One-off条件强调文本中字符一次性使用原则,使得删除、插入、替换操作彼此独立,APM-OF处理三种操作的灵活性仍然得以体现。

2) 特征2:模式裸长 m_0 。拟定 $k=1$,局部长度约束为 $[0, 3]$,每个 m 取值随机选择10组模式进行实验。如图5(b)所示,随着 m 的增大,APM和APM-OF解的增长率 δ 均大致呈先增后减的趋势。当 m 较小时,模式长度较短较容易替换匹配,本文算法在处理插入操作和删除操作上的灵活性未能得到充分体现;随着 m 增大,算法处理三种编辑操作的灵活性增大, δ 升高。当 m 进一步增大时,模式复杂性增加,寻找满足查询条件的解更加苛刻,故解优势也会有一定程度的变缓。

表4 四种算法在匹配数量方面的对比

模式	S_1						S_2					
	SAW	SA-OF	APM	APM-OF	$\delta_{APM}/\%$	$\delta_{APM-OF}/\%$	SAW	SA-OF	APM	APM-OF	$\delta_{APM}/\%$	$\delta_{APM-OF}/\%$
p_1	1536	407	1673	488	8.92	19.90	1632	405	1789	477	9.62	17.78
p_2	1844	439	1932	562	4.77	28.02	1993	437	2078	562	4.26	28.60
p_3	1256	329	1421	383	13.14	16.41	1343	349	1533	407	14.15	16.62
p_4	1206	270	1328	304	10.12	12.59	1191	262	1318	295	10.66	12.21
p_5	584	159	695	176	19.01	10.69	581	164	698	181	20.14	10.37
p_6	1001	172	1070	175	6.89	1.74	1143	179	1222	193	6.91	7.82
p_7	1970	149	2028	157	2.94	5.37	2016	156	2072	163	2.78	4.49
p_8	1210	268	1229	268	1.57	0.00	129	273	1269	273	1.60	0.00

模式	S_3						S_4					
	SAW	SA-OF	APM	APM-OF	$\delta_{APM}/\%$	$\delta_{APM-OF}/\%$	SAW	SA-OF	APM	APM-OF	$\delta_{APM}/\%$	$\delta_{APM-OF}/\%$
p_1	1549	391	1684	476	8.72	21.74	1192	295	1308	365	9.73	23.73
p_2	1874	421	1946	536	3.84	27.32	1498	326	1553	423	3.67	29.75
p_3	1183	312	1343	373	13.52	19.55	1006	266	1140	311	13.32	16.92
p_4	1120	259	1232	293	10.00	13.13	898	201	998	224	11.14	11.44
p_5	571	149	666	165	16.64	10.74	451	121	532	132	17.96	7.44
p_6	1064	173	1137	184	6.86	6.36	903	140	962	152	6.53	8.57
p_7	1954	150	1998	160	2.25	6.67	1500	119	1538	119	2.53	0.00
p_8	1146	264	1162	264	1.40	0.00	943	200	959	200	1.40	0.00

模式	S_5						S_6					
	SAW	SA-OF	APM	APM-OF	$\delta_{APM}/\%$	$\delta_{APM-OF}/\%$	SAW	SA-OF	APM	APM-OF	$\delta_{APM}/\%$	$\delta_{APM-OF}/\%$
p_1	1036	269	1127	325	8.78	20.82	1028	256	1119	310	8.85	21.09
p_2	1293	288	1345	363	4.02	26.04	1247	276	1301	340	4.33	23.19
p_3	789	215	904	257	14.58	19.53	760	194	866	244	13.95	25.77
p_4	754	182	835	204	10.74	12.09	753	168	830	186	10.23	10.71
p_5	416	106	500	114	12.43	7.55	342	94	411	105	20.18	11.70
p_6	648	102	697	108	7.56	5.88	676	113	725	122	7.25	7.96
p_7	1327	101	1361	110	2.56	8.91	1199	97	1241	102	3.50	5.15
p_8	805	176	824	176	2.36	0.00	766	169	781	169	1.96	0.00

模式	S_7						S_8					
	SAW	SA-OF	APM	APM-OF	$\delta_{APM}/\%$	$\delta_{APM-OF}/\%$	SAW	SA-OF	APM	APM-OF	$\delta_{APM}/\%$	$\delta_{APM-OF}/\%$
p_1	728	178	797	221	9.48	24.16	585	154	636	185	8.72	20.13
p_2	895	193	921	247	2.91	27.98	716	167	743	213	3.77	27.54
p_3	555	141	623	161	12.25	14.18	450	122	511	141	13.56	15.57
p_4	592	126	650	138	9.80	9.52	444	106	490	117	10.36	10.38
p_5	280	75	331	85	18.21	13.33	200	54	238	59	19.00	9.26
p_6	511	177	555	83	8.61	7.79	388	65	412	74	6.19	3.85
p_7	907	73	924	79	1.88	8.22	715	58	738	62	3.21	6.9
p_8	596	123	605	123	1.51	0.00	446	106	462	106	3.59	0.00

3) 特征3:局部长度约束下限 $N_i (0 \leq i < m-1)$ 。拟定 $k=2, m=7, N=N_0=N_1=\dots=N_i=\dots=N_{m-2}$, 每个 N 取值上随机选择10组模式进行实验。如图5(c)所示, 解增长率 δ 均呈先减后增的趋势。当 N 较小时, 插入和删除代价较小, 故 δ 较大。随着 N 的增大, 替换操作得到的解所占比重增大, 故 δ 下降。但随着 N 的进一步增大, 模式复杂性增加, APM 和 APM-OF 较 SAW 和 SA-OF 可以处理三种操作的优势趋于明显, δ 也会随之增大。

4.4 实验4:与 OneoffMining 算法在模式挖掘中对比

本节将 APM-OF 算法扩展至近似模式挖掘, 记为 MAPM 算法, 并与 OneoffMining 算法^[13]进行对比, 算法均主要由以下三个阶段组成:1)生成 k 项候选集;2)计算候选模式的支持度, 判断是否为频繁模式, 得到频繁 k 项集;3)将频繁 k 项集扩展为 $k+1$ 项集, 用支持度参数判断新的 $k+1$ 项集是否为频繁项集, 重复1)~2), 直到候选集为空。MAPM 算法和 OneoffMining 算法的主要区别在于匹配操作处理上的不同,

MAPM 算法调用 APM-OF 算法(插入、替换和删除操作),而 OneoffMining 算法调用 Quicksearch 算法^[13](仅精确匹配),故 MAPM 算法在模式近似挖掘方面要明显优于 OneoffMining 算法。令最小支持度 $\min_sup = 0.013 * |T|$, $|T|$ 为随机文本串的长度,分别为 1000, 2000, 3000, 4000, 间隔约束为 [9, 11], 实验结果如图 6 所示, MAPM 算法找到的频繁模式个数约为 OneoffMining 算法的 2.07 倍。

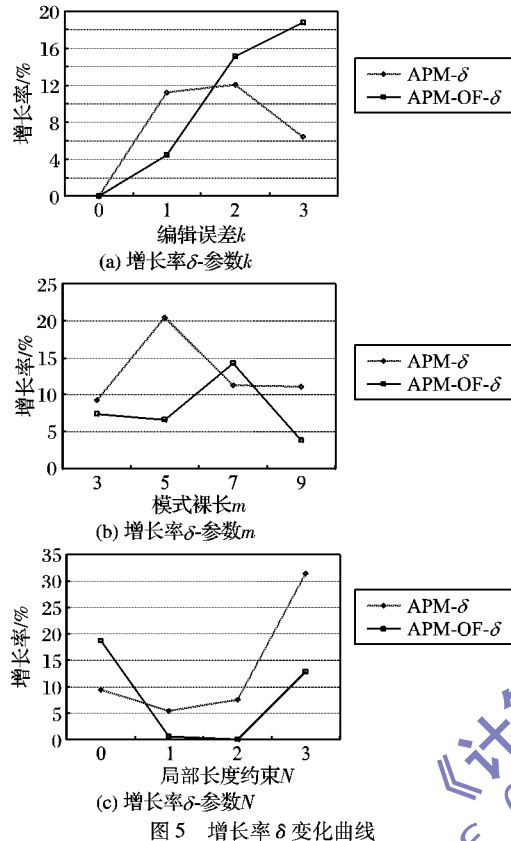


图5 增长率 δ 变化曲线

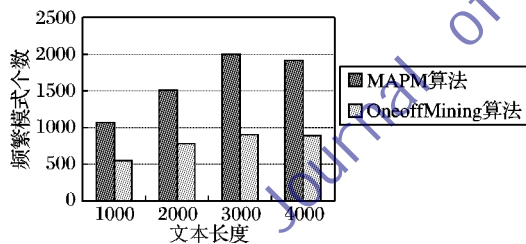


图6 MAPM 算法与 OneoffMining 算法对比

5 结语

针对 APMWL 问题, 本文提出的 APM 和 APM-OF 算法与现有算法相比所处理的问题更加灵活和复杂。在真实生物数据上的实验结果显示算法在解的平均增长率上分别达到 8.34% 和 12.37%, 反映了其有效性。同时, 本文分析了算法求解 APMWL 问题的主要因素, 发现当编辑误差 k 值较大,

模式 P 中字符(非通配符)的个数 m 适中, 局部长度约束下限 N_i 很小或很大时解增长率最为明显, 可分别达到 31.43% 和 18.78%。此外, 本文将 APM-OF 算法推广至模式挖掘中, 与同类算法 OneoffMining 相比, 挖掘出的频繁模式个数为其 2.07 倍, 这也意味着可以挖掘出更多的有效信息。以后我们将试图对算法时间及空间性能进行改进。如何在线输出质量更高的解(即编辑误差较小, 解的总个数更多), 也是我们未来的研究方向。

参考文献:

- [1] CHANG Y I, CHEN J R, HSU M T. A Hash Trie filter method for approximate string matching in genomic databases[J]. Applied Intelligence, 2010, 33(1): 21-38.
- [2] BHUKYA R, SOMAYAJULU D V L N. 2-jump DNA search multiple pattern matching algorithm[J]. International Journal of Computer Science Issues, 2011, 8(3): 320-329.
- [3] 龚才春, 黄玉兰, 许洪波, 等. 基于多重索引模型的大规模词典近似匹配算法[J]. 计算机研究与发展, 2008, 45(10): 1776-1781.
- [4] 范立新. 改进的中文近似字符串匹配算法[J]. 计算机工程与应用, 2006, 42(34): 172-207.
- [5] 樊爱京, 杨照峰. 用于网络入侵检测的模式匹配新方法[J]. 计算机应用, 2011, 31(11): 2961-2985.
- [6] QU Z Y, HUANG X B. The improving pattern matching algorithm of intrusion detection[C]// 2011 International Conference on Advanced in Control Engineering and Information Science. Oxford: Elsevier, 2011: 2841-2846.
- [7] ALTSCHUL S F, BOGUSKI M S, FISH W, et al. Issues in searching molecular sequence databases[J]. Nature Genetics, 1994(6): 119-128.
- [8] WU S, MANBER U, MYERS E W. A subquadratic algorithm for approximate limited expression matching[J]. Algorithmica, 1996, 15(1): 50-67.
- [9] NAVARRO G. Regular expression searching over Ziv-Lempel compressed text[C]// Proceedings of the 12th Annual Symposium on Combinatorial Pattern Matching. Berlin: Springer, 2001: 1-17.
- [10] FISCHER M J, PATERSON M S. String matching and other products[C]// Proceedings of the 7th SIAM-AMS Complexity of Computation. New York: American Mathematical Society, 1974: 113-125.
- [11] CHEN G, WU X D, ZHU X Q, et al. Efficient string matching with wildcards and length constraints[J]. Knowledge and Information Systems, 2006, 10(4): 399-419.
- [12] HE D, WU X D, ZHU X Q. SAIL-APPROX: an efficient on-line algorithm for approximate pattern matching with wildcards and length constraints[C]// IEEE International Conference on Bioinformatics and Biomedicine. Washington, DC: IEEE Computer Society, 2007: 151-158.
- [13] HUANG Y M, WU X D, HU X G, et al. Mining frequent patterns and one-off condition[C]// Proceedings of the 12th IEEE International Conference on Computational Science and Engineering. Washington, DC: IEEE Computer Society, 2009: 180-186.

(上接第 799 页)

- [6] CHAI-EAD N, AUNGKULANON P, LUANGPAIBOON P. Bees and firefly algorithms for noisy non-linear optimisation problems [C]// Proceedings of the International MultiConference of Engineers and Computer Scientists. Hongkong: [s. n.], 2011, 2.
- [7] 刘长平, 叶春明. 一种新颖的仿生群智能优化算法: 萤火虫算法[J]. 计算机应用研究, 2011, 28(9): 3295-3297.
- [8] 胥小波, 郑康锋, 李丹, 等. 新的混沌粒子群优化算法[J]. 通信学报, 2012, 33(1): 24-37.

- [9] 王凌, 刘波. 微粒群优化与调度算法[M]. 北京: 清华大学出版社, 2008: 38-39.
- [10] 王翔, 李志勇, 许国艺, 等. 基于混沌局部搜索算子的人工蜂群算法[J]. 计算机应用, 2012, 32(4): 1033-1036.
- [11] 周燕, 刘培玉, 赵静, 等. 基于自适应惯性权重的混沌粒子群算法[J]. 山东大学学报: 理学版, 2012, 47(30): 1-6.
- [12] 黄凯, 周永权. 带交尾行为的混沌人工萤火虫优化算法[J]. 计算机科学, 2012, 39(3): 231-234.