带任意长度通配符的模式匹配

强继朋 [1] 谢 飞 [2] 高 隽 [1] 胡学钢 [1] 吴信东 [1,3]

摘 要 基因序列中, 许多病毒并不是简单的直接复制自己, 而是相邻字符间插入或者删除序列片段, 如何从序列数据中检索这些病毒具有重要的研究价值. 提出了一个更普遍的问题, 带任意长度通配符的模式匹配问题 (Pattern matching with arbitrary-length wildcards, PMAW), 这里模式中不仅可以有多个通配符约束, 而且每个通配符的约束可以是两个整数, 也可以从整数到无穷大. 给定序列 S 和带通配符的模式 P, 目标是从 S 中检索 P 的所有出现和每一次出现的匹配位置, 并且要求任意两次出现不能共享序列中同一位置. 为了有效地解决该问题, 设计了两个基于位并行的匹配算法 MOTW (Method of ocurrence then window) 算法和 MWTO (Method of window then ocurrence) 算法. 同时, MWTO 算法进行细微改动就可以满足全局长度约束. 实验结果既验证了算法求解问题的正确性, 又验证了比相关的模式匹配算法具有更好的时间性能.

关键词 通配符, 模式匹配, 位并行, 基因序列

引用格式 强继朋, 谢飞, 高隽, 胡学钢, 吴信东. 带任意长度通配符的模式匹配. 自动化学报, 2014, **40**(11): 2499−2511

DOI 10.3724/SP.J.1004.2014.02499

Pattern Matching with Arbitrary-length Wildcards

QIANG Ji-Peng[1] XIE Fei[2] GAO Jun[1] HU Xue-Gang[1] WU Xin-Dong[1,3]

Abstract In genetic sequences, many viruses rarely reproduce themselves, but rather appear with a slightly different form in each of the occurrences. That is, sequence fragments may be inserted or deleted in adjacent characters. How to search for these viruses from the sequences has become an important research task. The paper presents a more general problem, named pattern matching with arbitrary-length wildcards (PMAW). Here, a pattern can have many wildcard constraints where the range of the wildcards may vary between two integer bounds or from an integer lower bound to infinity. Given sequence S and pattern P with arbitrary-length wildcards, this paper aims to search for all occurrences of P in S, and locate matching positions of each occurrence, where any two occurrences can not share the same position of S. In order to solve the problem effectively, two algorithms, MOTW (Method of ocurrence then window) and MWTO (Method of window then ocurrence), are proposed based on the bit-parallel technique. The MWTO algorithm can also meet the global length constraint with a minor modification. Experimental results validate the correctness of the proposed algorithms, and show that they perform better than the existing pattern matching algorithms.

Key words Wildcard, pattern matching, bit-parallel, genetic sequence

Citation Qiang Ji-Peng, Xie Fei, Gao Jun, Hu Xue-Gang, Wu Xin-Dong. Pattern matching with arbitrary-length wildcards. *Acta Automatica Sinica*, 2014, **40**(11): 2499−2511

近几十年来, 带通配符的模式匹配得到了广泛的关注, 不仅具有理论研究价值, 而且在生物信息学[1−3]、序列模式挖掘[4−5]、问答系统[6] 和数据流挖掘[7−8] 等领域都有着实际应用价值. 例如, 在基因序列中寻找生物病毒出现的位置, 由于病毒并不是简单的进行复制, 而是相邻的两个字符之间可能会插入和删除一些较短的序列片段, 所以, 对生物病毒进行检索时, 需要带通配符的模式匹配算法, 如生物信息学分析工具 BLAST[9].

Fischer 等[10] 最早提出了带通配符的模式匹配算法, 但是要求字符之间通配符的数目是固定值. 在带通配符的模式匹配研究中, 主要分为三类研究成果: 1) 字符之间通配符的数目是固定值[10−12], 例如模式 $a\phi\phi c\phi t$, 其中 'ϕ' 表示可以匹配字母表中任意一个字符; 2) 字符之间可以是无穷长度的通配符[4−5, 13], 用 $[0,\infty]$ 表示, 例如模式 $ac[0,\infty]t$, 表示

收稿日期 2013-12-05 录用日期 2014-02-17
Manuscript received December 5, 2013; accepted February 17, 2014
国家自然科学基金 (61229301), 国家重点基础研究发展计划 (973 计划) (2013CB329604), 教育部长江学者和创新团队发展计划 (IRT13059), 中国博士后科学基金 (2013M541822) 资助
Supported by National Natural Science Foundation of China (61229301), National Basic Research Development Program of China (973 Program) (2013CB329604), the Program for Changjiang Scholars and Innovative Research Team in University (PCSIRT) of the Ministry of Education (IRT13059), and the Postdoctoral Science Foundation of China (2013M541822)
本文责任编委 王飞跃
Recommended by Associate Editor WANG Fei-Yue
1. 合肥工业大学计算机与信息学院 合肥 230009 中国 2. 合肥师范学院计算机科学与技术系 合肥 230601 中国 3. 佛蒙特大学计算机科学系 佛蒙特州 伯灵顿 05405 美国
1. School of Computer Science and Information Engineering, Hefei University of Technology, Hefei 230009, China 2. Department of Computer Science and Technology, Hefei Normal University, Hefei 230601, China 3. Department of Computer Science, University of Vermont, Burlington, VT 05405, USA

ac 和 t 之间可以匹配无穷多个字符; 3) 字符之间通配符是可变长度的通配符[14−17], 表示为 $[l,\ h]$ $(0\leq l\leq h)$, 其中 l 表示最小长度约束, h 表示最大长度约束, 例如模式 $ac[1,\ 2]t$, 表示 ac 和 t 之间字符的最少是 1 个, 最多是 2 个. 可以看出, 第 1) 类问题的模式是第 3) 类模式的特例, 满足 $l=h$ 的要求, 如模式 $a\phi\phi c\phi t$ 可以表示成 $a[2,\ 2]c[1,\ 1]t$.

本文研究更普遍的一个问题, 带任意长度通配符的模式匹配问题 (Pattern matching with arbitrary-length wildcards, PMAW). 该问题有如下三个特点: 模式中可以包含多个通配符; 每个通配符的数目可以是两个整数, 或者是从整数到无穷; 序列中任意位置的字符只能被使用一次. 下面对这些特点进行举例说明:

例 1. 给定序列 $S=s_0s_1s_2s_3s_4s_5=aacccc$, 模式 $P=p_0[l_0,\ h_0]p_1[l_1,\ h_1]p_2=a[1,\ 2]c[1,\ \infty]c$.

可以发现 $\langle 0,\ 2,\ 4\rangle$ 是一个合法的出现, 这是因为 $p_0=s_0=$ 'a', $p_1=s_2=$ 'c', $p_2=s_4=$'c', 且 $l_0\leq 2-0-1\leq h_0$ 和 $l_1\leq 4-2-1\leq h_1$. 同理, $\langle 0,3,5\rangle$ 和 $\langle 1,3,5\rangle$ 也是满足要求的两个合法出现. 在序列中, 只要有一个位置发生变化, 就可能是一个合法出现. 如果存在无穷长度通配符, 或者可变长度通配符的最大长度约束比较大的情况, 模式的出现可能是指数级别的, 所以, 如果要获得每次出现的匹配位置是非常不可行的. 为此, 本文在计算模式的出现时, 引入了 One-off 条件: 模式的任意两次出现不能共享序列中同一位置的字符[5, 14]. One-off 条件有着重要的实际应用意义, 可以避免序列中同一字符多次使用. 例如, 模式 a 的出现次数为 2, 但是, 不考虑 One-off 条件的情况下, 模式 $a[0,\ \infty]c$ 的出现次数为 8, 这样计算模式的出现次数显然不是合理的[18].

可见, 引入通配符虽然使问题变得更加复杂, 但却使模式的形式更加灵活, 更有利于实际应用的需要. 比如, 在生物信息学中, 启动子 $TATA$ 序列常常出现在 $CAATCT$ 序列之后, 中间间隔 $30\sim 50$ 个通配符[13−14]; 在序列模式挖掘中挖掘带通配符的序列模式, 常常在模式之间分别指定可变长度通配符约束[18−19] 和无穷长度通配符约束[4−5].

综上所述, 本文研究的问题是: 给定序列 S 和带多个任意长度通配符模式 P, 从序列 S 中获得模式 P 的出现次数, 且任意两次出现要满足 One-off 条件. 同时考虑通配符约束和 One-off 条件是一个非常复杂的问题, 如例 1, $\langle 0,\ 2,\ 4\rangle$, $\langle 1,\ 3,\ 5\rangle$ 和 $\langle 0,\ 3,\ 5\rangle$ 是满足两个条件的两组解, 寻找 One-off 条件下的最大一组出现集合仍然是一个开放的问题[14]. 本文的主要贡献在于:

1) 提出了带任意长度通配符的模式匹配问题 PMAW. 不但允许模式可以包含有可变长度的通配符, 而且允许通配符的数目从整数到无穷大. 同时, 任何两次出现都要满足 One-off 条件.

2) 给出了两种解决 PMAW 问题的方法, 都采用了位操作和布尔操作的位并行技术去寻找模式在序列中的出现位置. 第一种方法 MOTW 从序列的左向右搜索模式的出现, 先确定是否存在模式的出现, 然后根据出现确定窗口, 确定窗口后继续向右搜索. 第二种方法 MWTO 先选择窗口的开始位置, 然后寻找该窗口内是否有模式的出现, 如果有, 就搜索出现位置, 否则向右移动窗口继续搜索. 详细分析了两个算法的时间复杂度和空间复杂度. 通过 DNA 数据和人工生成数据进行实验, 实验结果验证了提出算法的正确性和有效性.

3) 引入全局长度约束, 即对一次出现的在序列中的跨度进行约束. MWTO 算法进行简单的改变, 就可以满足全局长度约束. 加入全局约束后, 也间接地对无穷长度通配符进行了约束, 所以一些相关的带可变长度通配符的匹配算法对模式进行转换后也能进行处理. 实验结果验证了 MWTO 算法比已存在的模式匹配算法具有更好的性能.

下面是本文的组织结构: 本文第 1 节介绍了相关工作; 第 2 节给出了问题的定义; 第 3 节详细地描述了 MOTW 算法和 MWTO 算法的设计过程及两个算法的时间复杂度和空间复杂度, 和改动 MWTO 算法满足全局长度约束; 第 4 节给出了真实生物数据和人工生成数据的实验结果, 并对结果加以分析; 第 5 节给出本文的结论.

1 相关工作

要解决带通配符的模式匹配问题, 最简单的方法是模式表示成正则表达式, 然后选择相应的正则表达式算法去检索[20−21]. 但是, 正则表达式并不能很好的处理带通配符的模式匹配问题, 并且会严重降低搜索的效率[11]. 为此, 针对带通配符的模式匹配问题, 很多学者专门设计了不同的算法. 带通配符的模式匹配算法最早由 Fischer 等[10] 在 1974 年提出, 他们的问题中通配符的数目是固定的. 后来, 一系列提高时间复杂度的算法被提出[22−23]. 为了放宽对模式中通配符的约束, Manber 等[24] 研究了带可变长度通配符的模式匹配, 但是他们研究的方法只能处理单个可变长度的通配符.

目前, 研究者更致力于研究带多个通配符约束的模式匹配问题. 他们分别是带可变长度通配符的模式和无穷长度通配符约束的模式. 在可变长度通配符和无穷长度通配符的情况下, 只要一个子模式的出现位置发生变化, 就是该模式的一次出现, 这直接导致解的规模组合爆炸. 针对可变长度通配符的

情况下, 解的空间大小为 $O(nw^m)$, 这里 n 为序列的长度, w 为可变长度通配符的间距, m 为模式的长度. 对于无穷长度通配符的情况下, 解的空间大小为 $O(n^{m+1})$. 可以看出, 在这两种解的规模下, 要获得每次出现的匹配位置是不可行的. 为此, 目前的研究中, 主要有三方面解决方案: 只给出每次出现的结束位置或开始位置[8, 17, 25], 称之为宽松模式匹配; 只计算模式在序列中的出现次数[26] 和引入约束条件限制解的空间[4, 14], 这两种称之为严格模式匹配. 此外, 为了控制一次出现在序列中的跨度, 长度约束被引入到带通配符的模式匹配中[14, 26]. 长度约束不仅在模式匹配中得到应用, 而且在序列模式挖掘中也取得了广泛的应用[19, 27].

Navarro 等[25] 和 Bille 等[17] 解决了带可变长度约束的模式匹配问题. 他们采取了宽松模式匹配的方式, 这样解的空间大小就降低到 $O(n)$. Navarro 等[25] 采用了位并行的技术模拟搜索过程, 给出了 Gaps-BNDM 和 Gaps-shift-and 算法. 但是, 两个方法在处理连续两个可变长度约束中只有一个字符的情况下就不能很好的工作. Bille 等[17] 使用著名的 Aho-corasick (AC) 自动机[28] 去报到模式中每个子模式的出现, 根据当前子模式出现的位置和可变长度约束计算下一个子模式出现的有效范围. 通过对有效范围的删减达到了很好的时间效率和空间效率.

带通配符的模式匹配是带通配符的序列模式挖掘的核心与基础[4−5, 17]. 模式挖掘过程中, 模式挖据算法需要计算模式在序列中的出现次数, 出现次数超过指定阈值即为频繁模式. 为此, Min 等[26] 只计算模式在序列中的出现次数, 而不关心匹配位置. 他们采用了动态规划的思想, 通过保存子模式的出现次数, 获得模式的所有出现次数, 多项式时间内就可以完成搜索.

Chen 等[14] 发现序列中的每个字符被使用一次具有很重要的实际意义, 即 One-off 条件, 并提出了解决带可变长度通配符和 One-off 条件的 SAIL 算法. 引入了 One-off 条件后, 解的空间大小降低到 $O(n/m)$. SAIL 算法不仅能够获得模式的出现次数, 而且能够获得每次出现的匹配位置. 后来, Guo 等[29] 采用了位并行的方法去解决该问题, 实验结果显示, 当模式的长度和可变长度约束不大的情况下, 该方法具有更好的时间性能. 由于寻找 One-off 条件下的最大一组出现集合仍然是一个开放的问题, 武优西等[15] 在离线情况下, 对序列贪婪的搜索, 大部分情况下, 可以获得更多的出现解. 吴信东等[18] 把该方法运用到序列模式挖掘中, 相比其他挖掘算法可以挖到更多频繁模式.

Kucherov 等[13] 致力于解决带无穷长度通配符的模式匹配问题, 提出了基于有向非循环字图 (Directed acyclic word graph, DAWG)[30] 的方法. 给定多个模式, 他们是同时搜索多个模式, 验证是否有模式在序列中出现, 如果有, 就停止搜索. Zhang 等[31] 提出了基于快速傅立叶变化 (Fast Fourier transform, FFT) 的方法, 提高了解决该问题的时间复杂度.

为了能够挖掘出频繁模式, 序列模式挖掘中, 一般需要事先指定通配符长度约束范围, 这往往给用户带来的极大的不便. 所以, Ding 等[4] 和 Wu 等[5] 提出了解决无穷长度通配符的模式挖掘算法. 在挖掘算法中计算模式的出现次数时, Ding 等[4] 采用了 Non-overlap 条件和 Wu 等[5] 采用了 One-off 条件. Non-overlap 条件与我们定义的 One-off 条件非常相似, 但还存在着本质的区别. Non-overlap 条件只要求同一位置的子模式不同共享序列中相同位置, 不同位置的子模式仍然可以共享序列中同一个位置.

上述的方法, 都是分别解决可变长度通配符和无穷长度通配符的方法. Haapasalo 等[32] 提出了一种可以同时解决可变长度通配符和无穷长度通配符的方法. 他们的方法和 Bille 等[17] 非常类似, 采用了 AC 自动机模拟搜索的过程. 由于解空间的指数级, 他们也采用了只计算模式在序列中的结束位置, 即宽松模式匹配. 但是, 该方法有非常不合理的地方, 如模式 $P = a[0, 2]c[0, \infty]c$ 和序列 $S = s_0s_1s_2s_3s_4s_5 = accccc$, 当找到模式 $p_1 = c$ 的出现以后, 序列再搜索到字符 c 都是模式 P 的出现, 因为 p_2 和 p_1 满足无穷大的约束. 采用 Haapasalo 等的方法, 可以得到模式 P 在 S 中出现 4 次, 出现位置分别为 {2, 3, 4, 5}. 从上面的结果可以看出, 当模式中引入了无穷长度, 还只计算模式在文本中的结束位置是非常不合理的.

表 1 给出了上述相关模式匹配算法的对比. 通过表 1 可以看出, 本文的研究与 Chen 等[14] 和 Guo 等[29] 的研究工作相近. 区别在于, 他们只研究带可变长度通配符的模式匹配算法, 而本文的通配符约束, 既可以是可变通配符, 也可以是无穷通配符, 更具有一般性. 本文的研究不但求解难度更大, 而且更具有实际意义. 在可变长度通配符情况下, 可以根据通配符的最大长度约束确定模式出现的最大跨度, 从而能在最大跨度内搜索模式的一次出现和确定出现的匹配位置. 而在无穷通配符约束情况下, 很难确定一次出现的最大跨度, 所以确定一次出现的匹配位置非常困难, 极端情况下, 甚至会一次出现包含了序列的第一个字符和最后一个字符. 因此该问题的难度在于确定模式的所有出现及每次出现的匹配位置. 而在应用方面, 模式中有些通配符约束可以事先计算得到, 而有些通配符约束, 用户很难提供合适的

表 1 带通配符约束的模式匹配算法对比

Table 1 Comparison of pattern matching with wildcards

算法	固定数目	可变长度	无穷长度	约束条件	匹配类型	长度约束
Fischer 等[10]	√	-	-	-	严格匹配	无
Navarro 等[25]; Bille 等[17]	√	√	-	-	宽松匹配	-
Min 等[26]	√	√	-	-	严格匹配	有
Chen 等[14]; 武等[15−16]	√	√	-	One-off 条件	严格匹配	有
Kucherov 等[13]; Zhang 等[31]	-	-	√	-	宽松匹配	-
Ding 等[4]	-	-	√	Non-overlap 条件	严格匹配	无
Wu 等[5]	-	-	√	One-off 条件	严格匹配	有
Haapasalo 等[32]	√	√	√	-	宽松匹配	-
本文	√	√	√	One-off 条件	严格匹配	有

* 宽松匹配通常不考虑 “One-off 条件” 和长度约束, 这里 ‘-’ 表示不考虑.

约束值, 因而采用任意长度通配符约束将有助于提高用户的灵活性和发现更多有价值的模式.

2 问题定义

定义 1. 序列 $S = s_0 s_1 \cdots s_{n-1}$, 这里 n 为序列 S 的长度, $s_0 \in \sum(0 \le i \le n-1)$, 这里 $\sum$ 表示字母表, 字母表包括序列 S 中所有不同的字母, 用 $|\sum|$ 表示字母表大小. 例如, DNA 序列的字母表 $\sum = \{a, c, t, g\}$, $|\sum| = 4$; 蛋白质的字母表 $|\sum| = 20$.

定义 2. 模式 $P = p_0[l_0, h_0]p_1 \cdots [l_{j-1}, h_{j-1}]p_j \cdots [l_{m-2}, h_{m-2}]p_{m-1}$, 这里 m 表示模式 P 的长度, $p_i \in \sum$ 表示单个字符. $[l_{j-1}, h_{j-1}]$ 表示 p_{j-1} 和 p_j 之间的局部通配符约束, l_{j-1} 表示最小间隔约束, h_{j-1} 表示最大间隔约束, 这里 l_{j-1} 和 h_{j-1} 都是整数且满足 $l_{j-1} \le h_{j-1}$, 或者 l_{j-1} 是整数和 $h_{j-1} = \infty$. 当 $l_{j-1} = h_{j-1}$, 表示固定数目的通配符; $l_{j-1} = 0$ 和 $h_{j-1} = \infty$, 表示无穷长度的通配符. 如果模式中两个字符中间没有通配符, 可以用 [0, 0] 表示.

定义 3. 用 L 表示模式的可计算长度, 计算如式 (1), 用 $\min L$ 表示模式 P 的最小长度, 计算如式 (2).

$$L = m + \sum_{j=0}^{m-2} h_j(h_j \neq \infty) + \sum_{j=0}^{m-2} l_j(h_j = \infty) \quad (1)$$

$$\min L = m + \sum_{j=0}^{m-2} l_j \quad (2)$$

例 2. 给定模式 $P = a[0, 2]c[1, \infty]c$. 根据模式 P, 可以得到, $m = 3$, $l_0 = 0$, $h_0 = 2$, $l_1 = 1$, $h_1 = \infty$, $L = 3 + 2 + 1 = 6$, $\min L = 3 + 0 + 1 = 4$.

定义 4. 如果存在一个位置序列 $occ = (i_0, i_1, \cdots, i_{m-1})$, 这里 $0 \le i_0 < i_1 < \cdots < i_{m-1} \le n-1$, 满足以下条件:

1) $s_{i_j} = p_j$, 其中, $0 \le j \le m-1$;

2) $l_{j-1} \le i_j - i_{j-1} - 1 \le h_{j-1}$, 其中, $1 \le j \le m-1$, 则称 occ 为模式 P 在序列 S 中的一次出现.

定义 5. 给定模式 P 在序列 S 中的两次出现 $occ1 = (i_0, i_1, \cdots, i_p, \cdots, i_{m-1})$ 和 $occ2 = (j_0, j_1, \cdots, j_q, \cdots, j_{m-1})$. 如果 $i_p \neq j_q$, 对任意的 $0 \le i_p, j_q \le m-1$, 则称 $occ1$ 和 $occ2$ 满足 One-off 条件.

例 3. 如例 1, <0, 2, 4> 和 <1, 3, 5> 两次出现满足 One-off 条件, <0, 2, 4> 和 <0, 3, 5> 不满足 One-off 条件, 因为共享了位置 0.

定义 6. 给定模式 P 和序列 S, 设 max_occ 为模式 P 在序列 S 中的出现集合, 这里 max_occ 中任意两个出现都满足 One-off 条件, 用 $|max\_occ|$ 表示 max_occ 的出现数目. 如果不存在其他满足 One-off 条件的出现集合 max_occ', 满足 $|max\_occ'| > |max\_occ|$, 我们称 max_occ 为模式 P 在序列 S 中的最大出现集合.

例 4. 如例 1, <0, 2, 4> 和 <1, 3, 5> 为 P 在 S 中的最大出现集合, 出现数目为 2. <0, 3, 5>虽然也是 P 在 S 中的出现集合, 但是出现数目仅为 1, 不是最大出现集合.

给定序列 S 和带任意通配符的模式 P, 带有通配符的模式匹配算法就是从 S 中寻找 P 的出现次数及每次出现在 S 中的匹配位置.

在一些应用中, 用户也许需要控制一次出现在序列中的最大跨度和最小跨度, 即全局长度约束[6, 29].

定义 7. 给定长度为 n 的序列 S, 模式 $P = p_0[l_0, h_0]p_1 \cdots [l_{j-1}, h_{j-1}]p_j \cdots [l_{m-2}, h_{m-2}]p_{m-1}$, 和全局长度约束 $[G_N, G_M]$, 这里 G_N 表示全局最小长度约束和 G_M 表示全局最大长度约束. 位置序列

$occ=(i_0, i_1, \cdots, i_{m-1})$ 不仅要满足定义 4 的两个条件, 还要满足下面条件:

$G_N \le i_{m-1}-i_0+1 \le G_M$

我们称 occ 为模式 P 在序列 S 中的一次出现.

例 5. 模式 $P=a[0, 2]c[1, \infty]c$, $S=atcggcttc$, $G_N=5$ 和 $G_M=8$. 可以发现索引 <0, 2, 5> 是模式 P 在 S 中的一次出现, 满足全局约束, 因为 $5\le 5-0+1\le 8$, 而索引 $<0, 2, 8>$ 并不是一次出现, 因为 $8-0+1>8$, 不满足全局最大约束.

但引入全局长度约束后, 计算模式 P 在序列 S 中的出现集合, 还需要出现集合中每次出现都满足全局长度约束.

3　算法设计与分析

3.1　带通配符的位并行操作

3.1.1　非确定有限自动机 (NFA)

MOTW 算法和 MWTO 算法都是采用位并行技术实现搜索的过程. 位并行技术是利用了计算机机器字位运算的内在并行性, 把多个位装入一个机器字 w 中, 达到一次运算就可以更新 w 位, 这里 w 就是机器字的位数. 目前的计算机系统结构中, w 为 64 或者 128 的比较多, 因此实际运算中效果还是比较明显的.

MOTW 算法和 MWTO 算法都可以看作是使用一个非确定有限自动机 (Nondeterministic finite state automaton, NFA) 对扫描序列的过程进行模拟. NFA 是根据模式进行构建的, 如图 1 给出了模式 $a[0, 2]c[1, \infty]c$ 对应的一个 NFA. 我们是用位并行的技术模拟 NFA, 给定一个带通配符的可变模式, 模式就可以用长度为 L 的位掩码 $D=d_L\cdots d_1$ 进行表示, 这里位掩码位于长度为 w 的机器字中, L 为模式的可计算长度. 当扫描序列过程中, 每读入一个字符, NFA 上的活动状态将被改变.

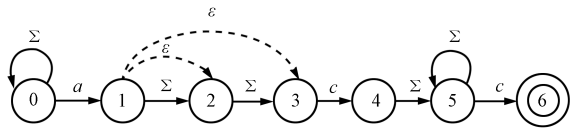

图 1　识别模式 "$a[0, 2]c[1, \infty]c$" 所有前缀的非确定自动机

Fig. 1　A nondeterministic automaton to recognize all the prefix factors of the pattern $a[0, 2]c[1, \infty]c$

当状态机的当前状态为一个终止态时 (如图 1 的状态 6), 就表示识别出模式, 即 $d_L=1$. 图 1 中的符号 Σ 表示字母表, 即可以匹配任意字符; ε 转移表示空转移, 不需要任何输入字符就可以跳转. 在字符 'a' 和 'c' 之间增加了两个 ε 转移, 使得当 'a' 被识别后能跳过随后的一个或两个状态, 从而对应了可变长度通配符 [0, 2].

3.1.2　位掩码

为了描述 NFA 上的状态和状态的转移, 需要定义以下几个位掩码: B 表、I、F 和 U, 每个位掩码需要 L 位表示. B 表存储的模式 P 上的不同字符和通配符的位掩码, B 表的大小为 $|\Sigma_P|+1$, 这里 Σ_P 就是模式 P 包含的不同字符的个数. 位掩码 I 和 F 分别用来表示可变长度通配符的初始状态和结束状态, 每一个可变长度通配符的初始状态或者结束状态都对应一个可变长度通配符 $[l, h]$ $(h\ne\infty)$, 如图 1 的状态 1 表示初始状态, 状态 3 表示结束状态. 位掩码 U 用来表示无穷通配符的起始位, 如图 1 的状态 5.

对于模式 P 上的每个字符 σ, $B[\sigma]$ 的位掩码在 NFA 上标注为 σ 或者 Σ 的位为 1, 如图 1, $B[c]=1^50$. 这里, 对于位的重复出现, 可以使用指数形式来表示, 例如, $0^31=0001$. 通配符 ϕ 的 $B[\phi]$ 在 NFA 上标注为 Σ 的位为 1, 序列中的字符被使用过, 就用 ϕ 表示, 可以保证该位置不会再次被使用, 如图 1, $\mathrm{B}[\phi]=010110$. 位掩码 I 和 F 分别在可变长度通配符的初始状态和结束状态的位置为 1, 如图 1, $I=0^410$, $F=0^310^2$. 位掩码 U 在无穷通配符的起始位为 1, 如图 1, $U=010^4$. 根据模式 P, 预处理模式 P, 得到位掩码的过程如算法 1 所示.

算法 1. Preprocess(P)

输入. 模式 P=$p_0[l_0, h_0]p_1\cdots[l_{m-2}, h_{m-2}]p_{m-1}$

输出. $B[\,]$, I, F, U

```
1  L ← 模式 P 的可计算长度，公式 (1)
2  for σ ∈ Σ_P∪φ do B[σ]←0^L
3  I ←0^L, F ←0^L, U ←0^L
4  i ←0
5  for j ∈ 0···m−2 do
6    for σ ∈ Σ_P∪φ do B[σ]← B[σ]|0^{L−i−1}10^i
7    i ← i+1
8    if h_j ≠ ∞ then
9      I ← I |0^{L−i}10^{i−1}
10     F ← F |0^{L−(i+h_j−l_j)}10^{i−h_j+l_j−1}
11       for σ ∈ Σ_P∪φ do B[σ]← B[σ]|0^{L−i−h_j}1^{h_j}0^i
12     i ← i+h_j
13   else /*h_j = ∞ */
14     U ← U |0^{L−i−l_j}10^{i+l_j+−1}
15       for σ ∈ Σ_P∪φ do B[σ]← B[σ]|0^{L−i−l_j}1^{l_j}0^i
16     i ← i+l_j
17 for σ ∈ Σ_P∪φ do B[σ]← B[σ]|0^{L−i−1}10^i
```

3.1.3 位操作

我们用位操作搜索模式是否出现在序列中, 使用位掩码 D 保存 NFA 上的状态. 当读入序列下一个字符为 σ, MOTW 算法和 MWTO 算法是采用式 (3) 和式 (4) 来模拟 NFA 上所有的状态转移.

$$D \leftarrow ((D \ll 1)\&B[\sigma])|(D\&U) \tag{3}$$

$$D \leftarrow D|((F-(D\&I)\wedge F) \tag{4}$$

式 (3) 用来关联模式 P 中每个字符和通配符的状态, 和处理整数到无穷通配符约束的状态转移. 主要分成两个部分: 1) "$D \ll 1$" 将 D 的所有位右移一位并以零填充右边移入的位, 再 "$\&B[\sigma]$" 用来实现当前的活动状态是否与读入的字符匹配; 2) 通过 $\&U$ 验证 D 中的起始状态是否是活动的状态. 如果是活动的状态, 通过与原来的结果进行或运算 "|", 就可以继续保存到 D 中, 从而可以满足匹配无穷通配符的要求.

式 (4) 实现 ε- 转移. 基本原理如下: $D\&I$ 使得活动的可变长度通配符的初始状态与其他状态相隔离, 然后把它从 F 中减去. 这样对于每个可变长度通配符 $[l_i, h_i]$ 的初始状态 j 都将得到两种可能的结果. 1) 如果 j 是活动的, 那么将使得从状态 j 到状态 $(j+h_i-l_i-1)$ 相对应的比特位均置为 1, 再通过 "$\wedge F$" 运算将状态 $(j+h_i-l_i)$ 对应的比特位置为 1, 并消除其他不需要的 1; 2) 如果 i 是不活动的, 那么仅仅将状态 $(i+h_i-l_i)$ 对应的比特位置为 1, 再通过 "$\wedge F$" 运算将状态 $(i+h_i-l_i)$ 对应的比特位置 0; 最后, 通过 "D |" 把 D 中原因已激活的状态继续保留到 D 中. 这样成功把 ε- 转移上的状态置为 1.

3.2 模式匹配算法简介

这一节将分别介绍 MOTW 算法和 MWTO 算法.

3.2.1 MOTW 算法

算法 2 给出了 MOTW 算法的执行步骤. 首先, MOTW 对模式 P 进行预处理, 获得 B 表、I、F 和 U, 如算法 1. 然后, 从左向右扫描序列 S, 使用位掩码 $D = d_L \cdots d_1$ 保存目前的状态. 在第 4 行, "$|0^{L-1}1$" 保证 d_1 一直等于 1, 从而使匹配可以从序列的任何位置开始进行匹配. 第 6 行, 是把每次搜索的状态保存到一个动态数组 *dVector* 中, 为后面的输出匹配位置使用. 如果 $d_L = 1$, 即 $(D\&10^{L-1}) \neq 0$, 表示成功识别模式 P 的一次出现, 这时, 会调用输出位置函数 (Occurrence_output) 去确定这次出现对应的序列的位置, 然后, D 赋值为 0 和清空 *dVector* 中的数据. Occurrence_output 函数根据 *dVector* 出现的结束位置, 确定模式中每个字符的位置, 并对确定的匹配位置都赋值为通配符, 防止下次被再次使用, 最后返回出现的开始位置到 *begin* 中. 根据位置 *begin*, MOTW 算法从下一个位置继续对序列进行循环扫描, 直到扫描结束.

算法 2. MOTW (Method of occurrence then window)

输入. 模式 P, 序列 S

输出. 出现

```
1  (B[ ], I, F, U)←Preprocess(P);
2  begin ← 0;
3  for i ←0; i < S.length; i ++ do;
4    D ←(((D ≪1)|0^{L-1}1)&B[s_i]) | (D&U);
5    D ← D |((F − (D&I))∧F);
6    dVector.add(D);
7    if ((D&10^{L-1}) ≠ 0 then
8      begin ← Occurrence_output(dVector,begin,i);
9      i ← begin;
10     begin++;
11     D ←0;
12     dVector.clear();
```

3.2.2 MWTO 算法

算法 3 给出了 MWTO 算法的执行步骤. MWTO 算法从左向右扫描序列中, 变量 i 记录窗口的开始位置, 由于模式出现的最小跨度是 min$L$, 所以要满足 $i < S$.length−min$L$+1. 位掩码 D 上来被初始化为 $0^{L-1}1$, 表示只能从位置 i 开始进行匹配. 然后, 从窗口的位置 i 循环扫描序列, 用变量 j 记录了当前所在的序列位置. 在第 6 行, 对式 (3) 进行了改变, 如式 (5) 所示. $D\&B[\mathrm{s}_i]$ 中 D 没有右移一位, 主要是因为 D 初始化为 $0^{L-1}1$, 如果右移, 就不能把位置 i 作为匹配的开始位置, 从而, 选择了在第 13 行对 D 进行右移. 同时, 由于在第 13 行对 D 进行右移, 为了获取右移前任意通配符的起始位是否为 1, 必须对 D 进行左移, 即 "$(D \gg 1)\&U$" 操作.

$$D \leftarrow (D\&B[s_i])|((D \gg 1)\&U) \tag{5}$$

窗口开始位置确定以后, 扫描过程中满足下面两种情况跳出窗口: 1) $(D\&10^{L-1}) \neq 0$ 表示当前的位置 j 处, 有一次出现, 调用输出函数 Occurrence_output; 2) $D = 0$ 表示没有活动状态. 然后, 会重新选择位置 i+1 作为窗口的开始位置, 循环扫描.

MWTO 算法与 MOTW 算法的不同之处是, MOTW 是一直扫描序列, 知道发现了模式的一次出现, 再确定出现的开始位置. 而 MWTO 是把序列中的每个位置都当作出现的开始位置进行扫描, 判断是否有以该位置为开始位置的出现.

算法 3. MWTO (Method of window then occurrence)

输入. 模式 P, 序列 S
输出. 出现

```
(B[ ],I,F,U)←Preprocess(P);
for i ←0; i < S.length−minL+1; i++ do
  j ← i
  D ←0^{L−1}1
  while j < S.length do
    D ←(D&B[s_i]) | ((D ≫1)&U);
    D ← D |((F − (D&I))∧F);
    dVector.add(D);
    if ((D&10L − 1) ≠ 0 then
      Occurrence_output(dVector,i,j);
      dVector.clear();
      break;
    if D = 0 then break;
    D ← D ≪1;
    ++j;
  if j = n then break;
```

3.2.3 输出位置

该阶段输出每次出现在序列中的位置, 当有多个可选位置输出的时候, 选择最左面的一个匹配位置, 即最左最优的思想. 首先, 在预处理阶段增加一个长度为 m 的数组 *occ*, 这里 m 为模式的长度. $occ[i]$ 用来标注 p_i 在模式 P 中的位置, 位掩码长度为 L. 如模式 $P = a[0,\ 2]c[1,\ \infty]c$, $occ[0] = 10^5$, $occ[1] = 0^3 10^2$, $occ[2] = 0^5 1$.

匹配位置的输出, 是从右向左输出. 首先, p_{m-1} 的匹配位置很容易确定, 即出现的结束位置. 每当发现了模式的匹配位置, 都对序列字符赋值为通配符 ϕ, 这样可以防止序列的该位置被重复使用, 即满足了 One-off 条件. 然后, 要想寻找到 $p_i(0 \le i \le m-2)$ 的出现, 根据 *dVector* 中的 D 值, 通过 $(dVector[minPrePos - begin]\&occ[i]) = 0$ 找到满足匹配要求和通配符约束的位置, 然后输出, 并赋值为 ϕ. 用到两个变量 *minPos* 和 *minPrePos*, 当寻找 p_i 的出现时, *minPos* 是用来记录 p_{i+1} 的匹配位置, 从而可以计算 p_i 的匹配区间. 对于 $h_i = \infty$ 的情况, p_i 的匹配区间必须在 '$minPrePos - l_i - 1$' 之前; 对于 $h_i \ne \infty$ 的情况, p_i 的匹配区间必须在 $[minPos - h_i - 1,\ minPrePos - l_i - 1]$ 之内. *minPrePos* 用来记录满足要求的匹配位置, 直到获取最左的匹配位置为止. 最后, 返回位置 *minPos*, 即为 p_0 的匹配位置, 也是这次出现在序列的开始位置. 伪代码如算法 4 所示.

算法 4. Occurrence_output
输入. *dVector*, *begin*, *end*, *occ*
输出. 匹配位置

```
output end;/* 匹配位置 */
s_end ← φ;
minPos ← end;
minPrePos ←0;
for i ← m−2; i ≥0; i– do
  if h_i = ∞ then
    minPrePos ← minPos − l_i−1;
    while (dVector[minPrePos − begin]&occ[i]) ≠ 0 do
      minPrePos−−;
    minPos ← minPrePos+1;
    output minPos; /* 匹配位置 */
    s_minPos ← φ ;
  else /*h_i ≠ ∞ */
    for k ← l_i; k ≤ h_i; k++ do
      if (dVector[minPrePos − k − 1 − begin]&occ[i])≠0
        minPrePos ← minPos − k − 1;
    minPos ← minPrePos;
    output minPos; /* 匹配位置 */
    s_minPos ← φ ;
return minPos;
```

3.3 模式匹配算法复杂度

这一节, 分析 MOTW 算法和 MWTO 算法的空间复杂度和时间复杂度.

3.3.1 空间复杂度分析

两个算法有相同的空间复杂度.

预处理过程, 需要 $O(|\sum_P|+4+m)$ 空间存储 B 表、I、F、U 和 *occ*. 对于每个位掩码需要 $\lceil L/w \rceil$ 个机器字长存储, 这里 w 为机器字长的位数. 所以, 预处理阶段的空间复杂度为 $O((|\sum_P|+4+m)\lceil L/w \rceil)$.

搜索阶段需要动态数组 *dVector* 保存每一个执行的状态. *dVector* 占用的空间为 $O(g\lceil L/w \rceil)$, 这里 g 为模式的一次出现在序列中最大跨度. 极端情况下, 在序列中没有模式的出现, 或者出现的结束位置在序列的结束位置时, *dVector* 占用的空间为 $O(n\lceil L/w \rceil)$.

综上所述, 一般情况下, MOTW 算法和 MWTO 算法的空间复杂度为 $O((|\sum_P|+4+m+g)\lceil L/w \rceil)$. 极端情况下, 两个算法的空间复杂度为 $O((|\sum_P|+4+m+n)\lceil L/w \rceil)$.

3.3.2 时间复杂度分析

预处理阶段需要对模式 P 进行预处理, 获得 B 表、I、F、U 和 *occ*, 需要的时间为 $O(L|\sum_P|)$.

MOTW 算法的时间复杂度:

搜索阶段需要的时间为 $O(n+\alpha g)$, 这里 α 为

MOTW 算法计算的模式在序列中的出现次数, 输出阶段需要的时间为 O(αg), 所以, MOTW 算法总的时间复杂度为 O(n+($L|\sum_P|$+2αg)$\lceil L/w\rceil$).

MWTO 算法的时间复杂度:

搜索阶段需要的时间为 O(n+fg), 这里 f 为模式首字母 p_0 在序列 S 中的频率, 输出阶段需要的时间为 O(αg), MWTO 算法总的时间复杂度为 O(n+($L|\sum_P|$+fg+αg)$\lceil L/w\rceil$).

可以看出, MOTW 算法相对 MWTO 算法有着更好的时间复杂度. 因为 $\alpha \le f$, 只有当 p_0 在模式出现一次, 而且 p_0 的每一次都是一次出现, 才满足 $\alpha = f$.

下面是计算极端情况下, MOTW 算法和 MWTO 算法的时间复杂度.

极端情况下, 序列中的每个字符都被模式的出现使用, 而且每次出现尽量有最大的跨度. 在这种极端的情况下, 两个算法匹配的次数最多, 消耗的时间也最多. 这种情况下, 两个算法有着相同的复杂度. 下面是计算过程:

要取得最坏的时间复杂度, 就是使 αg 取最大值. 由于序列中有 α 次出现, 所以 $g \le n-\alpha$, 即 $\alpha g \le \alpha\ (n-\alpha)$. 因为 One-off 条件, 序列中每个位置最多只能使用一次, α 满足下面条件:

$$\alpha \le n/m \tag{6}$$

要使 $\alpha\ (n-\alpha)$ 取得最大值, 就是 α 和 $(n-\alpha)$ 的值越接近. 由公式 (6) 可知, $\alpha \le (n-\alpha)$, 这里只考虑 $m > 1$ 的情况, 因为 $m = 1$ 时, 模式中不存在通配符.

$$\alpha g \le \alpha(n-\alpha) \le \frac{n}{m}(n-\frac{n}{m}) = \frac{n^2(m-1)}{m^2} \approx \frac{n^2}{m} \tag{7}$$

所以, 可以得到 αg 的最大值 $\max(\alpha g)$:

$$\max(\alpha g) = \frac{n^2}{m} \tag{8}$$

根据式 (8) 可以得到, MOTW 算法和 MWTO 算法的最坏时间复杂度为 O(n+($L\times|\sum_P|$+2$\frac{n^2}{m}$)$\lceil L/w\rceil$).

3.4 全局长度约束

MWTO 算法做一些简单的改动就可以满足全局长度约束.

针对 MWTO 算法, 仅仅算法 3 需要改动. 在第 5 行之前, 定义一个变量 *len*, 记录下面 while 循环执行的次数, 初始化了 $len = 0$. 然后, 对第 9 行进行改变, 验证循环扫描的序列字符数是否满足全局约束. 改变如下,

if(((D&10^{L-1}) $\neq$ 0)&&($len \ge G_N$)&&($len \le G_M$))

对第 12 行也进行改变, 如果循环中扫描的序列数大于最大全局约束 G_M, 也可以跳出循环. 改变如下,

if (($D = 0$)||($len \le G_M$))

最后, 在第 15 行之前, 对 *len* 进行加 1 操作, 从而记录循环中扫描的序列字符数.

指定全局约束后, 一次出现的最大跨度为 G_M. 所以, 引入全局长度约束后, MWTO 算法的空间复杂度为 ((| $\sum_P$ |+4+m+G_M)$\lceil L/w\rceil$), 和时间复杂度为 O(n+(L | $\sum_P$ |+fG_M+αG_M)$\lceil L/w\rceil$).

4 实验

这一节, 通过人工数据和真实生物数据的进行实验, 验证算法 MOTW 和算法 MWTO 在以下三方面的性能. 1) 提出的启发式算法 MOTW 和算法 MWTO 是否接近于最优解; 2) 算法的时间性能; 3) 选择类似的算法 SAIL[14] 和 BPBM[29], 验证提出的算法在全局约束情况下的时间性能.

实验过程中选择的对比算法有: 1) SAIL 和 BPBM: 可以处理带可变长度通配符和 One-off 条件的模式匹配算法; 2) SAIL_infinity: 我们对 SAIL 算法进行适当的修改, 使其可以处理带任意长度通配符的模式. 实验过程中使用的模式都是根据字母表的大小和模式的长度等变化, 随机生成的模式. 算法 MOTW 和算法 MWTO 的源代码已全部公开. 实验运行的软、硬件环境为 Intel(R) Core(TM) i5-3470 S 处理器、主频为 2.9 GHZ、内存为 8.0 GB、Windows7 操作系统的计算机.

4.1 完备率

我们设计了一个序列生成器, 可以产生指定出现次数的模式的序列. 指定模式 P 和 P 的出现次数为 *max_occ*, 序列生成器产出一个序列 S, 满足模式 P 在 S 中的最大出现为 *max_occ*. 然后, 通过我们的算法在 S 中搜索模式 P 的出现次数 *occ*, 就可以计算提出的算法与最优解的差距, 评估函数如公式 (9), 称为完备率 (*rate*).

$$rate = \frac{occ}{max\_occ} \tag{9}$$

本次做了三组实验: 验证模式长度变化对完备率的影响; 验证字母表大小变化对完备率的影响; 验证无穷通配符约束数目占总间隔约束数目的百分比对完备率的影响. 第一组中, 选择的字母表大小为 4, 序列长度为 100 000, $max\_occ = 100$, 模式的长度取值分别为 2、5、$\cdots$、29. 第二组中, 由于 DNA 的字母表为 4, 蛋白质的字母表为 20, 选择了字母表取值

表 2 模式长度对完备率的影响

Table 2 The complete rate under the length of pattern

模式长度	2	5	8	11	14	17	20	23	26	29
MOTW/MWTO/SAIL_infinity	1	0.973	0.935	0.893	0.865	0.793	0.763	0.724	0.675	0.667

表 3 模式长度和字符表大小对完备率的影响

Table 3 The complete rate under the size of the alphabet

字母表大小	2	4	6	8	10	12	14	16	18	20
MOTW/MWTO/SAIL_infinity	0.913	0.94	0.969	0.961	0.966	0.978	0.974	0.991	0.979	0.988

表 4 无穷通配符比例对完备率的影响

Table 4 The complete rate under the proportion of infinity wildcard

无穷通配符比例	0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
MOTW/MWTO/SAIL_infinity	0.922	0.856	0.862	0.851	0.851	0.857	0.841	0.856	0.855	0.858

分别为 2、4、6、$\cdots$、20, 序列长度为 10 000, max_occ = 100, m = 5. 第三组中, 选择的字母表大小为 4, 序列长度为 10 000, max_occ = 100, 模式的长度为 11, 总共通配符约束为 10, 无穷通配符所占比例分别取 0.1、0.2、$\cdots$、0.9. 实验结果如表 2 $\sim$ 4 所示, 可以得到如下结论:

1)SAIL_infinity、MOTW 和 MWTO 三个算法具有一样的完备率, 验证了 MOTW 算法和 MWTO 算法的求解问题的正确性. 主要因为它们在选择匹配位置时, 都是采取同样的启发式思想;

2) 在模式长度比较逐渐增加的过程中, 完备率逐渐降低, 主要因为模式长度增加, 模式之间的共享位置现象比较严重, 而算法都是选择最左的位置, 导致丢解;

3) 字母表大小逐渐增加, 完备率逐渐升高. 当字母比较小的时候, 模式中相同字符比较多, 不同出现之间匹配位置共享序列的相同位置比较严重, 所以丢解相对较严重一点;

4) 无穷通配符占总间隔数目的比例对完备率有影响. 当不包含无穷通配符和全部是无穷通配符约束时, 完备率最高, 主要因为当没有无穷通配符, 可选位置都在有限的范围内, 冲突相对较小一点. 当所有的都是无穷通配符, 所有出现的出现位置都可以共享, 该次出现的位置被其他出现占用影响不大, 能够选择其他出现的出现位置概率比较高, 所以完备率较高. 当包含部分无穷通配符时, 有的位置在有限范围, 有的在整个序列, 如果一次出现的下一个是灵活通配符约束的位置被其他出现选中, 能够选择其他出现的出现位置的概率比较低, 从而会造成更多的丢解.

4.2 模式匹配算法的时间性能

这一节, 主要对比了 SAIL_infinity、MOTW 和 MWTO 三个算法的时间性能. 考虑了模式长度和字母表大小两个因素进行实验. 实验序列是根据字母表 Σ = 4、8、16、24 随机生成的序列, 长度均为 10 000 字符. 实验过程中, 变化模式的长度 m = 2、5、$\cdots$、32, 求上面几个算法运行的时间, 这里每组不同长度的模式根据序列的字符表随机生成 1 000 个, 实验结果是求平均值.

实验结果如图 2 所示, 可以看出 SAIL_infinity 算法在搜索中花费的时间远远大于 MOTW 算法和 MWTO 算法. 当模式长度增加 14 以后, SAIL_infinity 算法花费的时间太长, 就没有在图中标注出来. 可以看出, 随着模式长度的增加, 运行时间大幅度增加, 主要因为 SAIL_infinity 算法的时间性能与模式的长度和最大通配符约束都成正比关系. 在任意长通配符情况下, 窗口和最大通配符约束很难确定. 而 MOTW 算法和 MWTO 算法不随着字母表和模式长度的变化而大幅度变化, 具有良好的稳定性.

由于 MOTW 算法和 MWTO 算法在性能上远远优于 SAIL_infinity 算法, 图 2 并不能很好地显示出 MOTW 算法和 MWTO 算法的具体情况. 为此, 针对模式长度和字母表大小, 单独对比了 MOTW 算法和 MWTO 算法. 序列的长度 n = 10 000, 固定字母表的大小, 验证模式变化对运行时间的影响, 和固定模式长度的大小, 验证字母表大小变化对运行时间的影响. 实验结果如表 5 所示. 从实验结果, 可以得到如下发现:

1) 两个算法都随着模式的增加或字母表的增加, 运行的时间先增加后降低. 根据时间复杂度的分析, 影响时间性能的主要因素是 O(αg), 这里 α 为模式在序列中的出现次数和 g 为模式的一次出现在序列中的平均跨度. 模式长度的增加 或者 字母表 的增

表 5 MOTW 与 MWTO 运行时间比较
Table 5 Running time comparisons of MOTW and MWTO

(a) $\Sigma = 4$ ms

m	2	5	8	11	14	17	20	23	26	29	32	35
MOTW	9	48	35	39	31	32	43	48	45	46	44	45
MWTO	10	50	37	42	34	34	45	52	50	49	48	49

(b) $\Sigma = 20$ ms

m	2	5	8	11	14	17	20	23	26	29	32	35
MOTW	3	9	4	4	2	4	3	4	3	3	2	1
MWTO	4	9	7	5	5	3	4	4	4	2	2	3

(c) $m = 5$ ms

Σ	2	4	6	8	10	12	14	16	18	20	22	24
MOTW	81	76	91	52	49	39	26	25	22	17	17	12
MWTO	69	73	89	52	50	39	26	25	22	18	17	13

(d) $m = 10$ ms

Σ	2	4	6	8	10	12	14	16	18	20	22	24
MOTW	83	98	72	47	30	28	20	14	11	11	9	6
MWTO	78	101	79	51	33	31	20	19	15	11	10	8

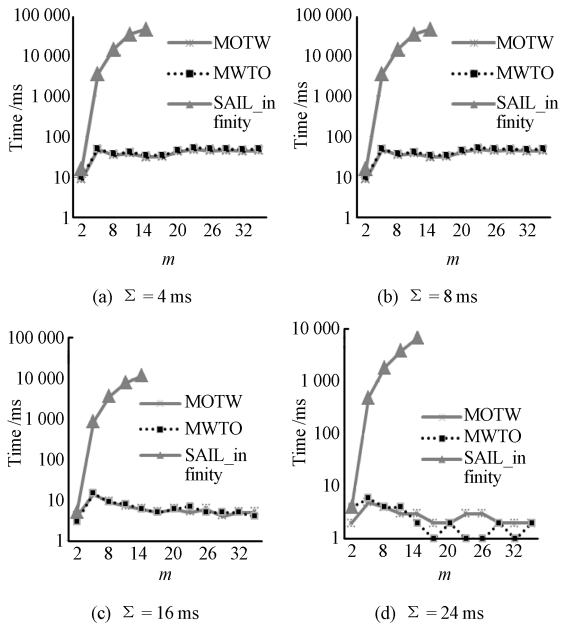

图 2 MOTW、MWTO 和 SAIL_infinity 的时间比较
Fig. 2 Running time comparisons of MOTW, MWTO and SAIL_infinity

加会导致 α 降低和 g 增加. 当模式长度比较小或者字母比较小的时候, g 占主导地位, 反之, 当模式长度比较大时子字母表比较大的时候, α 占主导地位.

2) MOTW 算法整体上要优越于 MWTO 算法. 表 5 总共 48 组实验, MOTW 算法优越于 MWTO 算法的有 31 次, 两者相等的 11 次, 比 MWTO 算法差的有 6 次. MOTW 算法比较差的情况, 都集中在字母表比较小的情况, 主要因为该情况下, 模式的出现次数比较多, 造成 MOTW 确定窗口的次数比较多. 总体上, 在处理没有长度约束的模式, 还是选择 MOTW 算法.

4.3 全局长度约束

这一节, 考虑引入全局长度约束后对算法性能的影响. 采用的两个真实的生物数据集是 AX 829 174 和 AB 038 490, 可以从美国国家计算信息中心的网站上进行下载[33]. 实验过程中, 考虑了全局长度约束的长度和模式的长度对算法时间性能的影响.

PMAW 问题引入了全局约束后, 如果对模式进行转变, 就可以采用带可变长度通配符的算法进行搜索. 引入了全局长度约束后, 对于每一个 $h_i = \infty$ 的约束都可以采用下式进行转变:

$$h_i = \infty \Leftrightarrow h_i = G_M - \min L + l_i \qquad (10)$$

即, 模式转变的方法就是把模式中 $h_i = \infty$ 都转化成 $h_i = G_M - \min L + l_i$. 改后的模式和原来的模式之所以是等价的, 是因为当引入全局约束后, 每个无穷长度通配符都不是无穷的, 而是要满足全局约束, 即一个无穷长度通配符的最大值情况就是其他通配符都取最小值的情况, 即式 (10). 转变后的模式就没有无穷长度通配符, 所以可以采用带可变长度通配符的算法进行求解. 例如, $P = a[0, 2]c[1, \infty]c$, G_N

$= 5$ 和 $G_M = 8$, 可知 $\min L = 4$, $P = a[0, 2]c[1, \infty]c = P = a[0, 2]c[1, 5]c$. 实验中选用了两个最好的能够处理可变长度通配符的算法 SAIL 和 BPBM 作为对比算法.

第一组实验, 选择数据集 AX829174, 固定模式的长度和可变通配符约束的最大长度, 变化全局约束, 实验结果如图 3 所示. 第二组实验, 选择数据集 AB038490, 固定全局长度约束和可变通配符约束的最大长度, 变化模式的长度, 实验结果如图 4 所示.

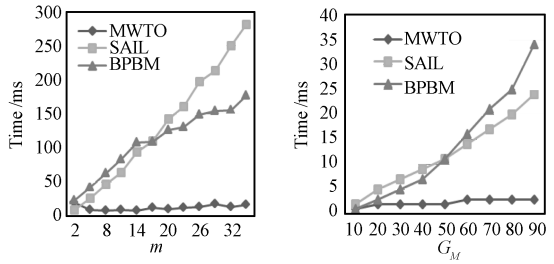

图 3　数据集 AX829174 上 MWTO、SAIL 和 BPBM 的时间对比

Fig. 3　Running time comparisons of MWTO, SAIL and BPBM under the dataset AX829174

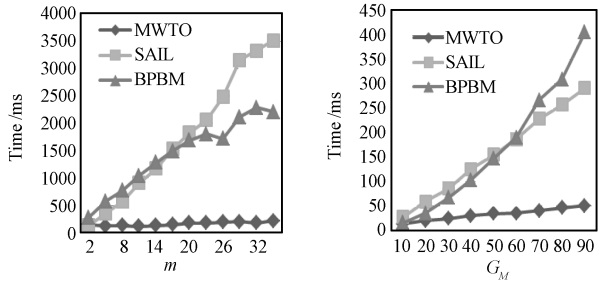

图 4　数据集 AB038490 上 MWTO、SAIL 和 BPBM 的时间对比

Fig. 4　Running time comparisons of MWTO, SAIL and BPBM under the dataset AB038490

随着全局长度约束 G_M 的增加或者模式长度 m 的增加, MWTO 算法要优越于 SAIL 算法和 BPBM 算法. 尽管采用了式 (10) 对模式进行转换, 使 SAIL 算法和 BPBM 算法能够处理任意长度的通配符的模式, 但是模式的总长度会发生变化. 转换后, 模式的总长度会受无穷通配符数目的多少和全局长度约束的大小而影响. 当全局长度约束增加, $G_M - \min L + l_i$ 肯定会增加, 即模式的总长度会大幅度增加, 而 BPBM 的位掩码是用模式的总长度进行表示的, 所以 BPBM 需要更多机器字长进行表示位掩码. 同样, 对于模式长度变化也是一样, 模式长度增加, 通配符约束也增加, 无穷长通配符的数目也可能增加. 所以, 尽管全局约束固定不变, 模式的总长度会随着模式的长度增加而增加, 导致使用更多的机器字长.

随着全局长度约束 G_M 增加或者模式长度 m 增加, MWTO 算法花费的时间趋于稳定. 影响 MWTO 算法最主要的因素是模式可计算长度占机器字长数目, 即 $\lceil L/w \rceil$. L 并不受全局长度约束的影响. 目前, 机器 $w = 64$ 或者 128 居多, 模式长度的增加对 $\lceil L/w \rceil$ 的影响不大. 所以, MWTO 算法比较稳定.

5　结论

本文提出了带有任意长度通配符的模式匹配问题 PMAW, 这里模式中可以包含多个通配符约束, 而且每个通配符约束既可以是两个整数, 也可以是整数到无穷. 该问题给用户带来了极大的方便, 无论模式中字符之间的通配符约束可以确定, 还是很难确定, 它都能够解决. 该问题不仅要求能够获得模式在序列中的出现次数, 而且能获得每次出现的匹配位置, 这里要求模式的任意两个出现都不能共享序列中的同一位置. 本文提出了解决该问题的 MOTW 算法和 MWTO 算法, 都采用了位并行的技术去模拟非确定有限自动机 (NFA), 并详细地分析了两个算法的时间复杂度和空间复杂度. 通过大量真实的生物数据和人工数据进行实验, 实验验证了 MOTW 算法和 MWTO 算法的正确性, 比相关的匹配算法具有更好的性能. 处理没有长度约束的情况下, 选择 MOTW 算法更合适. 当有长度约束时, 选择 MWTO 算法.

怎样获得 PMAW 问题的完备解仍然是一个开放的问题, 将作为我们进一步的研究方向. 目前, 并没有多项式的算法能够得到模式的最大出现集合, 所以, 我们猜测该问题有可能是一个 NP-hard 问题[6]. 下一步着手证明 PMAW 问题是否是 NP-Hard 问题, 如果不是, 打算提出一个多项式的算法对其进行解决.

References

1 Pisanti N, Crochemore M, Grossi R, Sagot M F. Bases of motifs for generating repeated patterns with wildcards. *IEEE/ACM Transactions on Computational Biology Bioinformatics*, 2005, **2**(1): 40−50

2 Yue Feng, Sun Liang, Wang Kuan-Quan, Wang Yong-Ji, Zuo Wang-Meng. State-of-the-art of cluster analysis of gene expression data. *Acta Automatica Sinica*, 2008, **34**(2): 113−120
(岳峰, 孙亮, 王宽全, 王永吉, 左旺孟. 基因表达数据的聚类分析研究进展. 自动化学报, 2008, **34**(2): 113−120)

3 Rahman M S, Iliopoulos C S, Lee I, Mohamed M, Smyth W F. Finding patterns with variable length gaps or don't cares. *Computing and Combinatorics*. Berlin, Heidelberg: Springer, 2006. 146−155

4 Ding B L, Lo D, Han J W, Khoo S C. Efficient mining of closed repetitive gapped subsequences from a sequence database. In: Proceedings of the 25th International Conference on Data Engineering. Shanghai, China: IEEE, 2009. 1024−1035

5 Wu X D, Zhu X Q, He Y, Arslan A N. PMBC: Pattern mining from biological sequences with wildcard constraints. *Computers in Biology and Medicine*, 2013, **43**(5): 481−492

6 Wang Bao-Xun, Liu Bing-Quan, Sun Cheng-Jie, Wang Xiao-Long, Sun Lin. Thread segmentation based answer detection in Chinese online forums. *Acta Automatica Sinica*, 2009, **39**(1): 11−20
(王宝勋, 刘秉权, 孙承杰, 王晓龙, 孙林. 基于论坛话题段落划分的答案识别. 自动化学报, 2013, **39**(1): 11−20)

7 Pan Yun-He, Wang Jin-Long, Xu Cong-Fu. State-of-the-art on frequent pattern mining in data streams. *Acta Automatica Sinica*, 2006, **32**(4): 594−602
(潘云鹤, 王金龙, 徐从富. 数据流频繁模式挖掘研究进展. 自动化学报, 2006, **32**(4): 594−602)

8 Tanbeer S K, Ahmed C F, Jeong B S, Lee Y K. Efficient frequent pattern mining over data streams. In: Proceedings of the 17th ACM Conference on Information and Knowledge Management. California, USA: ACM, 2008. 1447−1448

9 Altschul S F, Gish W, Miller W, Myers E W, Lipman D J. Basic local alignment search tool. *Journal of Molecular Biology*, 1990, **215**(3): 403−410

10 Fischer M J, Paterson M S. String-matching and other products. Proceeding of the 7th SIAM AMS Complexity of Computation. Cambridge, USA, 1974. 113−125

11 Clifford P, Clifford R. Simple deterministic wildcard matching. *Knowledge and Information Systems*, 2007, **101**(2): 53−54

12 Cole R, Gottlieb L A, Lewenstein M. Dictionary matching and indexing with errors and don′t cares. In: Proceedings of the 36th Annual ACM Symposium on Theory of Computing. New York, USA: ACM, 2004. 91−100

13 Kucherov G, Rusinowitch M. Matching a set of strings with variable length don′t cares. *Theoretical Computer Science*, 1997, **78**(1−2): 129−154

14 Chen G, Wu X D, Zhu X Q, Arslan A N, He Y. Efficient string matching with wildcards and length constraints. *Knowledge and Information Systems*, 2006, **10**(4): 399−419

15 Wu You-Xi, Liu Ya-Wei, Guo Lei, Wu Xin-Dong. Subnettrees for strict pattern matching with general gaps and length constraints. *Journal of Software*, 2013, **24**(5): 915−932
(武优西, 刘亚伟, 郭磊, 吴信东. 子网树求解一般间隙和长度约束严格模式匹配. 软件学报, 2013, **24**(5): 915−932)

16 Wu You-Xi, Wu Xin-Dong, Jiang He, Min Fan. A heuristic algorithm for MPMGOOC. *Chinese Journal of Computers*, 2011, **34**(8): 1452−1462
(武优西, 吴信东, 江贺, 闵帆. 一种求解 MPMGOOC 问题的启发式算法. 计算机学报, 2011, **34**(8): 1452−1462)

17 Bille P, Görtz I L, Vildhój H W, Wind D K. String matching with variable length gaps. *Theoretical Computer Science*, 2012, **443**(1): 25−34

18 Wu Xin-Dong, Xie Fei, Huang Yong-Ming, Hu Xue-Gang, Gao Jun. Mining sequential patterns with wildcards and the one-off condition. *Journal of Software*, 2013, **24**(8): 1804−1815
(吴信东, 谢飞, 黄咏明, 胡学钢, 高隽. 带通配符和 One-Off 条件的序列模式挖掘. 软件学报, 2013, **24**(8): 1804−1815)

19 Ji X N, Bailey J, Dong G Z. Mining minimal distinguishing subsequence patterns with gap constraints. *Knowledge and Information Systems*, 2007, **11**(3): 259−286

20 Bille P, Thorup M. *Languages and Programming*. Berlin: Springer Heidelberg, 2009. 171−182

21 Navarro G, Raffinot M. New techniques for regular expression searching. *Algorithmica*, 2005, **41**(2): 89−116

22 Indyk P. Faster algorithms for string matching problems: matching the convolution bound. In: Proceedings of the 39th Symposium on Foundations of Computer Science. Washington, DC, USA: IEEE, 1998. 166−173

23 Kalai A. Efficient pattern-matching with don′t cares. In: Proceedings of the 13th ACM-SIAM Symposium on Discrete Algorithms. Philadelphia, PA, USA: ACM, 2002. 655−656

24 Manber U, Baeza-Yates R. An algorithm for string matching with a sequence of don′t cares. *Information Processing Letters*, 1991, **37**(3): 133−136

25 Navarro G, Raffinot M. Fast and simple character classes and bounded gaps pattern matching, with applications to protein searching. *Journal of Computational Biology*, 2003, **10**(6): 903−923

26 Min F, Wu X D, Lu Z Y. Pattern matching with independent wildcard gaps. In: Proceedings of the 8th IEEE International Conference on Dependable, Autonomic and Secure Computing. Chengdu, China: IEEE, 2009. 194−199

27 Ferreira P G, Azevedo P J. Protein sequence pattern mining with constraints. In: Proceedings of the 9th European Conference on Principles and Practice of Knowledge Discovery in Databases. Berlin Heidelberg: Springer, 2005. 96−107

28 Aho A V, Corasick M J. Efficient string matching: an aid to bibliographic search. *Communications of the ACM*, 1975, **18**(6): 333−340

29 Guo D, Hong X L, Hu X G, Gao J, Liu Y L, Wu G Q, Wu X D. A bit-parallel algorithm for sequential pattern matching with wildcards. *Cybernetics and Systems*, 2011, **42**(6): 382−401

30 Blumer A, Blumer J, Haussler D, Ehrenfeucht A, Chen M T, Seiferas J. The smallest automation recognizing the subwords of a text. *Theoretical Computer Science*, 1985, **40**(1): 31−55

31 Zhang M, Zhang Y, Hu L. A faster algorithm for matching a set of patterns with variable length don′t cares. *Information Processing Letters*, 2010, **110**(6): 216−220

32 Haapasalo T, Silvasti P, Sippu S, Soisalon-Soininen E. Online dictionary matching with variable-length gaps. In: Proceedings of the 10th International Symposium, SEA Kolimpari, Chania, Crete, Greece. Berlin Heidelberg: Springer, 2011. 76−87

33 NCBI National Center for Biotechnology Information [Online], available: http://www.ncbi.nlm.nih.gov, November 20, 2013

强继朋 合肥工业大学计算机与信息学院博士研究生. 主要研究方向为模式匹配与挖掘.
E-mail: qjp2100@163.com
(**QIANG Ji-Peng** Ph. D. candidate at the School of Computer Science and Information Engineering, Hefei University of Technology. His research interest covers pattern matching and mining.)

谢 飞 合肥师范学院计算机科学与技术系副教授. 分别于 2007 年和 2011 年获得合肥工业大学硕士和博士学位. 主要研究方向为数据挖掘.
E-mail: xiefei9815057@sina.com
(**XIE Fei** Associate professor in the Department of Computer Science and Technology, Hefei Normal University. He received his master and Ph. D. degrees from Hefei University of Technology in 2007 and 2011. His main research interest is data mining.)

高 隽 合肥工业大学计算机与信息学院教授. 1999 年获中国科学技术大学博士学位. 主要研究方向为图像处理, 模式识别和神经网络理论及应用.
E-mail: gaojun@hfut.edu.cn
(**GAO Jun** Professor at the School of Computer Science and Information Engineering, Hefei University of Technology. He received his Ph. D. degree from University of Science and Technology of China. His research interest covers image processing, pattern recognition and neural network theory and applications.)

胡学钢 合肥工业大学计算机与信息学院教授. 2000 年获合肥工业大学博士学位. 主要研究方向为机器学习、人工智能和数据挖掘.
E-mail: jsjxhuxg@hfut.edu.cn
(**HU Xue-Gang** Professor at the School of Computer Science and Information Engineering, Hefei University of Technology. He received his Ph. D. degree from Hefei University of Technology. His research interest covers machine learning, artificial intelligence and data mining.)

吴信东 合肥工业大学计算机与信息学院长江学者. 1993 年获英国爱丁堡大学人工智能博士学位. 美国佛蒙特大学计算机科学系统教授, IEEE Fellow 和 AAAS Fellow. 主要研究领域为数据挖掘, 专家系统和万维网信息处理. 本文通信作者. E-mail: xwu@hfut.edu.cn
(**WU Xin-Dong** Yangtze river scholar at the School of Computer Science and Information Engineering, Hefei University of Technology. He received his Ph. D. degree in artificial intelligence from University of Edinburgh, United Kingdom in 1993. He is a professor in computer science at University of Vermont, USA, and a fellow of the IEEE and AAAS. His research interest covers data mining, knowledge-based systems and web information exploration. Corresponding author of this paper.)