软件学报 ISSN 1000-9825, CODEN RUXUEW
*Journal of Software*,2018,29(2):363−382 [doi: 10.13328/j.cnki.jos.005255]

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel: +86-10-62562563

# 一般间隙与 One-Off 条件的序列模式匹配*

刘慧婷[1,2], 刘志中[1,2], 黄厚柱[1,2], 吴信东[3,4]

[1](计算智能与信号处理教育部重点实验室(安徽大学),安徽 合肥 230039)

[2](安徽大学 计算机科学与技术学院,安徽 合肥 230601)

[3](合肥工业大学 计算机科学与信息工程学院,安徽 合肥 230009)

[4](School of Computing and Informatics, University of Louisiana at Lafayette, Lafayette 70503, USA)

通讯作者: 刘慧婷, E-mail: htliu@ahu.edu.cn

**摘 要**: 带有间隙约束的模式匹配问题是序列模式挖掘的关键问题之一.目前,大多数的研究都为非负间隙,对字符串中每个字符的出现顺序有着严格的要求.为了增加匹配的灵活性,并且考虑到在序列模式挖掘中采用 one-off 条件更加合理,研究一般间隙与 one-off 条件下的模式匹配问题.该问题为 NP-Hard 问题.为了有效地求解该问题,提出了 MSAING(maximum sequential pattern matching with one-off and general gaps condition)算法:首先,利用 Reverse 策略使模式与序列达到最佳的匹配状态;然后,使用线性表的结构使匹配过程中消耗的时间和空间大幅度地降低,同时,利用回溯机制提高匹配的成功率;最后,根据 inside_Checking 机制判断模式串是否会产生内部重复现象,以进一步提高算法的执行效率.理论证明了 MSAING 算法的完备性,实验结果验证了 MSAING 算法匹配结果的准确性以及在时间和空间方面的高效性.

**关键词**: 一般间隙;one-off 条件;模式匹配;线性表

**中图法分类号**: TP181

中文引用格式: 刘慧婷,刘志中,黄厚柱,吴信东.一般间隙与 one-off 条件的序列模式匹配.软件学报,2018,29(2):363−382. http://www.jos.org.cn/1000-9825/5255.htm

英文引用格式: Liu HT, Liu ZZ, Huang HZ, Wu XD. Sequential pattern matching with general gap and one-off condition. Ruan Jian Xue Bao/Journal of Software, 2018,29(2):363−382 (in Chinese). http://www.jos.org.cn/1000-9825/5255.htm

## Sequential Pattern Matching with General Gap and One-Off Condition

LIU Hui-Ting[1,2], LIU Zhi-Zhong[1,2], HUANG Hou-Zhu[1,2], WU Xin-Dong[3,4]

[1](Key Laboratory of Intelligent Computing and Signal Processing (Anhui University), Ministry of Education, Hefei 230039, China)

[2](School of Computer Science and Technology, Anhui University, Hefei 230601, China)

[3](School of Computer Science and Information Engineering, Hefei University of Technology, Hefei 230009, China)

[4](School of Computing and Informatics, University of Louisiana at Lafayette, Lafayette 70503, USA)

**Abstract**: Pattern matching with gap constraints is one of the key issues of sequential pattern mining. Recently, most research work focuses on pattern matching with non-negative gaps, but the rule strictly limits the order that each character appears in the sequence. In order to increase the flexibility of matching while taking into account that it is more reasonable to use one-off condition in sequential pattern mining, this paper studies the pattern matching problem under general gap and one-off condition, which is NP-hard. To tackle this issue, an algorithm, named MSAING, is proposed. Firstly, the algorithm processes the pattern and sequence using the Reverse strategy to

* 基金项目: 国家重点研发计划(2016YFB1000901); 国家自然科学基金(61202227)

Foundation item: National Key Research and Development Program of China (2016YFB1000901); National Natural Science Foundation of China (61202227)

收稿时间: 2016-09-28; 修改时间: 2016-11-23; 采用时间: 2017-01-10; jos 在线出版时间: 2017-01-22

CNKI 网络优先出版: 2017-01-22 16:42:06, http://www.cnki.net/kcms/detail/11.2560.TP.20170122.1642.001.html

get the maximum number of matching results. Secondly, it significantly reduces the time and space overhead with linear table structure in the matching process, and improves the matching rate using the backtracking method. Finally, to further improve the efficiency of the algorithm, it determines whether internal repetition exists in the pattern or not, according to the inside_Checking mechanism. Completeness of the MSAING algorithm is proved in theory. Experimental results verify the accuracy of the matching results of the MSAING algorithm and its validity in terms of the time and space complexity.

**Key words**: general gap; one-off condition; pattern matching; linear table

随着大数据时代的到来,模式匹配技术已被广泛应用到生物信息学[1]、空气质量监测[2]、数据流挖掘[3]以及信息检索与过滤等重要领域[4].在生物信息学方面,研究者利用模式匹配技术,借助 TATA 框,能够快速地从数以亿计的 DNA 序列中定位内含子的起始位置,通常会在 CAATCT 后 30 个~50 个字符出现,可以表示成···CAATCT[30,50]TATA···[5];在空气质量监测方面,通过空气质量传感器可以实时地监测一个城市周边的空气质量指数,地理位置相近的传感器检测到的空气质量指数变化往往具有相关性,利用模式匹配技术监测变化是否具有周期性的特点,可以帮助人们找到该城市空气污染的主要因素并采取措施防止污染扩散[2];在数据流挖掘方面,利用模式匹配技术能够找出模式在数据序列中出现的位置,有效地预测数据流的变化规律[3];在信息检索方面,可通过带间隙的模式匹配技术从网络上大量的文本信息中正确检索出用户所需要的信息[6].

Fischer 等人首次在模式匹配中提出了通配符的概念[7].通配符可以代表任意的字符,记为 Φ, 如 $a\Phi c\Phi\Phi g$. 随后,Manber 等人[5]将通配符的个数从固定值变为可变值,但没有限制字符个数的最大长度.Chen 等人[8]提出了独立通配符的概念,形如 $P=p_0[\min_0,\max_0]p_1\ldots p_j\ldots[\min_{m-2},\max_{m-2}]p_{m-1}$,其中,$\min_j$ 和 $\max_j$ 分别指 $p_j$ 和 $p_{j+1}$ 之间通配符个数的最小值和最大值,当各个间隙区间都相同时,称为周期间隙[9].Liu 等人[10]对带间隔约束与 one-off 条件的序列模式匹配问题进行了研究,通过 CluTree 结构,提高了匹配解的完备性.

上述研究讨论的均是非负间隙的模式匹配,即在进行匹配过程中,子模式 $p_j$ 一定出现在 $p_{j+1}$ 之前.由于序列的有序性,采用非负间隙的模式匹配技术无疑禁止了子模式串次序颠倒现象的发生.然而在实际生活中,这种次序颠倒的现象普遍存在,若采用非负间隙进行模式匹配,就会漏掉一些有价值的信息.例如,通过检测人体的某段 DNA 序列中含有 CAATCT 基因片段的数目,可以判断是否患有某种疾病.由于基因片段在转录的过程中发生变异,片段“AT”和“CT”出现了位置颠倒,同时,变异后的基因与原基因具有相同的致病效果,我们进行模式匹配时不仅要统计 DNA 序列中 CAATCT 出现的次数,而且还要统计变异后出现的次数.此时,如果利用非负间隙模式匹配进行统计,则会忽略变异情况;如果利用一般间隙模式匹配,则可以准确地进行统计.Fredriksson 等人[11,12]对间隙可以为负情况下(称为一般间隙)的模式匹配进行了研究,但未对序列中的每个位置能否被多次使用进行讨论.之后,柴欣等人[13]采用网树结构,在一般间隙模式匹配问题上进行研究,允许序列中的每个位置能够被多个出现使用;文献[14]中考虑了 one-off 条件,即模式的任意两次出现都不共享序列中的相同位置[13]. one-off 条件更加符合实际需要,如上述的生物学的例子,由于基因在转录过程中形成特定的 RNA,所以 DAN 序列中的位置最多只能被致病基因匹配 1 次.

综上所示,利用一般间隙与 one-off 条件的模式匹配进行序列挖掘,能够发现更多有价值的信息,更符合实际应用的需要.由于序列模式匹配是序列挖掘的关键步骤,本文对基于一般间隙与 one-off 条件的序列模式匹配(sequential pattern matching with general gaps and one-off condition,简称 SPMGOO)进行研究.该问题有如下 5 个特点:是 NP-Hard 问题;是一种精确的模式匹配;是一种严格的模式匹配;是基于 one-off 条件的模式匹配;模式串中的间隔可以为负值.

一般间隙模式匹配通过并、或运算可以分解成非负间隙模式匹配[15].而相对于非负间隙约束,一般间隙约束更具有灵活性,因为非负间隙约束都需要在前一事件发生在后一事件之前的理想情况下进行模式匹配.然而在实际生活中,事件发生的顺序是可以改变的,如上述对 DNA 中致病基因的分析.同时,one-off 条件也更加符合实际需要,如典型的序列模式挖掘的例子,购买了手机,一段时间后很可能购买手机配件,则每一次购买手机与手机配件应该只作为 1 次出现统计.因此,利用一般间隙的模式匹配进行序列挖掘能够发现更多有价值的信息,具有更大的实际应用价值.本文的主要贡献如下.

(1) 提出了一般间隙与 one-off 条件的序列模式匹配问题 SPMGOO.

在具有间隙约束的模式串中,允许子模式串之间的间隙为负值;同时加入了 one-off 条件,允许序列串中任意位置的字符最多使用 1 次的精确的严格模式匹配.之后,通过理论证明了 SPMGOO 问题为 NP-Hard 问题.

(2) 首次使用线性表解决 SPMGOO 问题,并且在模式匹配的过程中首次提出对模式串结构以及符号集$\Sigma$中各个元素在序列串中的频度进行分析,判断是否需要转置操作,使模式与序列达到最佳匹配状态.

(3) 提出了基于一般间隙与 one-off 条件的最大数目的序列模式匹配算法 MSAING.

MSAING 算法首先采用 Reverse 策略判断是否需要转置操作;然后,利用线性表的结构进行模式匹配,具体分为 Locate 阶段、Forward 阶段、Backward 阶段,使 MSAING 算法在模式匹配过程中消耗的时间和内存大为减少,同时,在 Backward 阶段使用回溯机制,使匹配的成功率大幅度提高;最后,提出了 inside_Checking 机制判断模式串是否会产生内部重复现象,以及如果产生内部重复会在模式串的哪个位置产生,从而有效地提高了 MSAING 算法的运行效率.经过理论分析得出,MSAING 算法的时间复杂度为 $O(MSAING)=O(n+kmg(l+m))$,空间复杂度为 $O(lm)$.其中,$n$ 为序列串的长度,$k$ 表示字符 $p_{m-1}$ 在序列串 $S$ 中出现的频度,$m$ 为模式串 $P$ 的长度,$l=\sum_{i=0}^{m-2}\max_i-\min_i+m$ 为建立搜索表的长度,$g=\max\{\max_i-\min_i\}$表示最大的间隙长度.

(4) 从理论上证明了 MSAING 算法的正确性和完备性,并对比测试了 MSAING 算法的性能.

SPMGOO 问题为 NP-Hard 问题,本文首先从理论上证明了 MSAING 算法比目前已有算法具有更好的完备性,对于不含重复的模式能够取得完备解;其次,本文在真实的生物数据集以及文本上,与 DCNP 等多种相关的改进算法进行了对比实验,通过实验结果验证了 MSAING 算法具有较高的准确性和较低的时空复杂度,并对实验结果及其意义进行了分析.

本文第 2 节对相关工作进行总结.第 3 节给出 SPMGOO 问题的定义和相关理论分析.第 4 节介绍 MSAING 算法设计的过程,并分析算法的时间和空间复杂度以及算法的正确性与完备性.第 5 节通过大量对比实验验证 MSAING 算法的性能.第 6 节给出结论.

## 1 相关工作

目前,对于带有间隙约束的模式匹配问题的研究可以分为以下 6 个方面:是周期模式匹配还是具有多个可变间隙的模式匹配;是精确匹配还是近似匹配;是严格模式匹配还是宽松模式匹配;一般间隙与否;是否具有 one-off 条件;是否具有长度约束[13].

Fischer 等人在 1974 年首次在模式匹配中提出了通配符的概念[7].在研究模式匹配过程中,根据通配符的不同形成了 5 类研究成果:最先开始研究的是连续的模式匹配问题[7],即子模式串之间不含通配符;子模式串之间通配符的个数是非负且固定的[16];子模式串之间通配符的个数范围为 0~∞[17],用[0,∞]表示,如 $ac[0,\infty]t$ 表示 $ac$ 和 $t$ 之间可以有无穷多个通配符;子模式串之间通配符的个数范围是上下限可变的非负数[8],其中的特例是,当每个间隙的变化范围相同时,称为带周期间隙的模式匹配[18],如 $a[1,2]c[1,2]t$;若子模式串之间的通配符的变化范围可以为负数时称为一般间隙模式[15],如例 1 中的模式串为一般间隙模式串.对于带通配符的模式匹配的研究发展过程如图 1 所示.

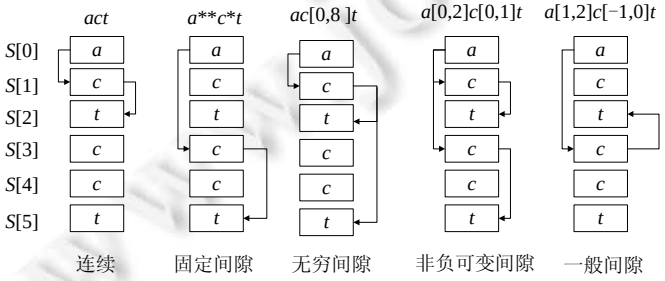



Fig.1 Development process of pattern matching with wildcard

图 1 带通配符的模式匹配的发展过程

近几年来,随着带一般间隙模式匹配问题研究的深入,人们逐渐意识到一般间隙模式匹配问题的重要性.Myers[19]最早对一般间隙模式匹配问题进行了研究,接着,Fredriksson 等人[11,12]分别在 2006 年和 2008 年对一般间隙模式匹配问题进行研究,并应用到蛋白质序列匹配和音乐信息检索等领域.武等人在网树结构的基础上做了大量模式匹配方面的研究,其中,在文献[20]中通过将 SGSP 算法和 SRMP 算法相结合提出了 SBO 算法,并在对一般间隙及 one-off 条件的严格模式匹配问题进行研究的基础上[13]提出了 DCNP 算法.但是,基于网树结构的算法在模式匹配过程中时间和空间开销较大,而且算法仅考虑到局部最优而忽略了模式以及序列本身所具有的特性.

在文献[8]中,Chen 等人提出了 SAIL 算法.该算法在 one-off 条件约束下进行模式匹配,利用 one-off 条件过滤掉大量冗余信息,提高了匹配的效率.但是算法只适合在线的模式匹配,在离线的模式匹配情况下,解的完备性较差.Lam 等人[21]利用 one-off 条件对压缩序列模式挖掘问题进行了研究.表 1 对比了几种具有间隙的模式匹配算法.

**Table 1** Comparison of pattern matching with gap constraints

**表 1** 几种具有间隙约束的模式匹配对比

| 相关工作 | 严格/宽松 | 间隙类型 | 间隙个数 | 匹配类型 | one-off | 长度约束 |
|---|---|---|---|---|---|---|
| Manbe 等人[5] | 严格 | 非负间隙 | 一个可变 | 精确 | 否 | 无 |
| Bille 等人[6] | 宽松 | 非负间隙 | 多个可变 | 精确 | – | – |
| Fredriksso 等人[11] | 宽松 | 一般间隙 | 多个可变 | $\delta$近似 | 否 | 无 |
| Chen 等人[8] | 严格 | 非负间隙 | 多个可变 | 精确 | 是 | 有 |
| He 等人[22] | 严格 | 非负间隙 | 多个可变 | Hamming 近似 | 是 | 有 |
| Ding 等人[23] | 严格 | 非负间隙 | 多个可变 | 精确 | 否 | 有 |
| 武等人[13] | 严格 | 一般间隙 | 多个可变 | 精确 | 是 | 无 |
| 本文 | 严格 | 一般间隙 | 多个可变 | 精确 | 是 | 无 |

注:宽松模式匹配下通常不考虑 one-off 条件和长度约束问题,本文以“–”表示不予考虑

从表 1 可以看出,本文与文献[8,13]的研究工作比较接近,但本文与文献[8]主要区别在间隙的类型不相同.文献[8]为非负间隙与 one-off 条件下的在线模式匹配,因为在非负间隙的条件下,模式子串 $p_j$ 对应序列串中字符的位置一定在 $p_{j+1}$ 之前,因此在匹配出现解的过程中,只需要从前向后单向扫描即可.本文研究的是间隙可以为负的一般间隙的模式匹配,而在一般间隙的条件下,模式子串 $p_j$ 对应在序列串中字符的位置可能在 $p_{j+1}$ 之后,从而在一个出现中可能有内部重复的现象.因此,本文提出的 MSAING 算法不仅需要考虑如何避免出现与出现之间的位置重复,还要考虑如何避免出现的内部重复,求解难度更大,也更具有实际意义.

本文与文献[13]的区别在于:文献[13]一次性地在整个序列串上建立网树结构,当序列串或模式串较长时,将消耗大量的时间和内存空间;文献[13]中提出的 DCNP 算法属于贪心算法,容易陷入局部最优,从而无法获得全局最优解.本文提出利用线性表解决 SPMGOO 问题,通过线性表对序列分段存储、分段匹配,减少了时间和空间的消耗;本文所提出的 MSAING 算法通过对模式串、序列串的整体形式分析,使模式匹配的解尽可能地达到全局最优,并且利用回溯的方法提高匹配的成功率,采用检测机制提高执行效率,所以与 DCNP 相比效率更高,更加接近匹配的完备解.

## 2 SPMGOO 问题

### 2.1 问题定义

**定义 1**. 序列串 $S=\{(s_0s_1\ldots s_i\ldots s_{n-1})|s_i\in\Sigma,1\leqslant i\leqslant n-1\}$,其中,$n$ 表示序列串 $S$ 的长度.$s_i\in\Sigma$代表符号集,应用场景不同,$\Sigma$代表的符号也不相同,如,DNA 序列中的符号集合$\Sigma$是由$\{A,C,G,T\}$构成的;而在音乐信息检索方面,$\Sigma$由音符构成.

**定义 2**. 具有间隙约束的模式 $P$ 可表示成

$$P=\{(p_0[\min_0,\max_0]p_1\ldots[\min_{j-1},\max_{j-1}]p_j\ldots[\min_{m-2},\max_{m-2}]p_{m-1})|p_j\in\Sigma\},$$

其中,$m$ 表示模式串的长度;$\min_j,\max_j$ 为给定的整数值,分别表示 $p_j$ 和 $p_{j+1}$ 之间间隙的最小值和最大值,且满足

$\min_j \leqslant \max_j$.若模式串 $P$ 中的间隙满足$\{\exists \min_i<0 | 0\leqslant i\leqslant m-1\}$的条件,则模式串 $P$ 称为一般间隙模式串;否则,模式串 $P$ 称为非负间隙的模式串.

**定义 3**. 给定模式串 $P=p_0[\min_0,\max_0]p_1\ldots p_j\ldots[\min_{m-2},\max_{m-2}]p_{m-1}$,如果 $p_i=p_j$,其中,$0\leqslant i.j\leqslant m-1$ 且 $i\neq j$,则模式串 $P$ 称为带重复字符的模式串,简写为 R 模式(pattern with recurring characters),如 $a[0,1]c[0,2]c[0,1]t$;否则称为不含重复字符的模式串,简写为 NR 模式.若 $p_0=p_1=\ldots=p_j$ 成立,则模式串 $P$ 称为首部重复的模式串,简写为 RH 模式,如 $a[0,1]a[0,2]c$;同理,如果 $p_i=p_{i+1}=\ldots=p_{m-1}$,则模式串 $P$ 称为尾部重复的模式串,简写为 RT 模式,如 $a[0,1]c[0,2]c$.

**定义 4**. 若一个序列 $S$ 中的位置索引 $occ=\langle o_0,o_1,\ldots,o_j,\ldots,o_{m-1}\rangle$,其中,$0\leqslant j\leqslant m-1, 0\leqslant o_j\leqslant n-1$ 满足以下的条件:

$$S_{o_j}=p_j \tag{1}$$

$$o_{j-1}\neq o_j \tag{2}$$

$$\begin{cases}\min_{j-1}\leqslant o_j-o_{j-1}-1\leqslant \max_{j-1}, & \text{if } o_{j-1}<o_j\\ \min_{j-1}\leqslant o_j-o_{j-1}\leqslant \max_{j-1}, & \text{if } o_{j-1}>o_j\end{cases} \tag{3}$$

则称 $occ$ 是模式串 $P$ 在序列串 $S$ 中满足间隙约束的一个出现.

**定义 5**. 给定一个出现 $occ=\langle o_0,\ldots,o_j,\ldots,o_{m-1}\rangle$,若 $o_k=o_q(k\neq q,0\leqslant k,q\leqslant m-1)$的情况存在,则称为有内部重复的出现;否则,称为无内部重复的出现.

**定义 6**. 给定两个无内部重复的出现 $occ_i=(o_0^i,\ldots,o_k^i,o_{m-1}^i)$ 和 $occ_j=(o_0^j,\ldots,o_q^j,o_{m-1}^j)$, 如果满足 $\{\forall k,q,o_k^i\neq o_q^j, 0\leqslant k,q\leqslant m-1\}$,则称 $occ_i$ 和 $occ_j$ 满足 one-off 条件;否则,称 $occ_i$ 和 $occ_j$ 不满足 one-off 条件.

**定义 7**. 令集合 $OCC_{SPMGOO}$ 表示模式串 $P$ 在序列串 $S$ 中的满足一般间隙与 one-off 条件的出现,其大小用 $|OCC_{SPMGOO}|$表示,本文即计算具有 one-off 条件的一般间隙的最大模式匹配数目,即求$|OCC_{\max}|$的最大值.

### 2.2 理论分析

一般间隙约束的模式匹配存在内部重复的可能,但并不是所有的模式匹配都会出现内部重复.下面分析在哪些情况下可能出现内部重复.

(1) 非负间隙的模式串不会有内部重复的出现,因为模式串中的字符 $p_j$ 一定在 $p_{j-1}$ 的后面出现,如 $a[0,1]c[0,3]a$.

(2) NR 模式不会有内部重复的出现,因为模式串中的每个字符不可能在序列串中占用相同的位置,如 $P_t=a[0,2]c[-1,1]t$.

(3) 只有在一般间隙约束下的 R 模式才可能有内部重复的出现,即 $p_j=p_i(0<j,i\leqslant m-1)$,并且 $p_j$ 可能出现在 $p_i$ 之后,如 $P=a[0,1]c[-1,2]c[-1,2]c$.

为了避免有内部重复的出现以及提高算法的执行效率,针对可能出现内部重复的情况,本文提出了内部重复检测机制.引理 1 说明了需要进行内部重复检测的条件.

**引理 1**. 在一般间隙模式匹配中,当模式串 $P$ 的长度大于等于 2 时,如果 $p_j=p_{j+k+1}$,并且满足 $\sum_{t=0}^{k}\min_{j+t}+pos_{\min}\leqslant 0\leqslant\sum_{t=0}^{k}\max_{j+t}+pos_{\max}$, 其中,$0\leqslant j\leqslant m-2, 0\leqslant k\leqslant m-2, pos_{\min}$ 和 $pos_{\max}$ 分别表示由 $p_j$ 到 $p_{j+k+1}$ 内最小间隙约束值 $\min_{j+t}\geqslant 0$ 的个数以及 $\max_{j+t}\geqslant 0$ 的个数,则需要进行内部重复检测.

证明:给定模式串 $P=p_0[\min_0,\max_0]p_1\ldots[\min_{j-1},\max_{j-1}]p_j\ldots[\min_{m-2},\max_{m-2}]p_{m-1}$,$p_j$ 出现的位置设为 $h$,$p_j$ 到 $p_{j+k+1}$ 的间隙约束为$[\min_j,\max_j]\ldots[\min_{j+k},\max_{j+k}]$,所以 $p_{j+k+1}$ 出现的最小位置为 $\sum_{t=0}^{k}\min_{j+t}+pos_{\min}+h$,最大位置为 $\sum_{t=0}^{k}\max_{j+t}+pos_{\max}+h$. 当 $\sum_{t=0}^{k}\min_{j+1}+pos_{\min}>0$ 时,$p_{j+k+1}$ 只能出现在 $p_j$ 之后;当 $\sum_{t=0}^{k}\max_{j+t}+pos_{\max}+h$ 时,$p_{j+k+1}$ 只能出现在 $p_j$ 之前,均不会有重复的可能;只有当 $\sum_{t=0}^{k}\min_{j+t}+pos_{\min}\leqslant 0\leqslant\sum_{t=0}^{k}\max_{j+t}+pos_{\max}$ 时,$p_j$ 与 $p_{j+k+1}$ 匹配的位置才有重复的可能.因此,当 $p_j=p_{j+k+1}$ 且满足 $\sum_{t=0}^{k}\min_{j+t}+pos_{\min}\leqslant 0\leqslant\sum_{t=0}^{k}\max_{j+t}+pos_{\max}$ 时,在 $p_j$ 处可能会有内部重复的出现,在这种情况下,需要进行内部检测.

**定理 1**. 一般间隙与 one-off 条件的模式匹配的判定问题的计算复杂性为 NP-Complete 问题.

证明:文献[13]给出了具体的计算复杂性的证明,此处从略. □

由于一般间隙与 one-off 条件的模式匹配的判定问题的计算复杂性为 NP-Complete 问题,所以一般间隙与 one-off 条件的序列模式匹配的优化问题为 NP-Hard 问题.如何匹配出最优的完备性的解集以及大幅度地降低时空消耗,是本文的核心任务.同时,在一般间隙模式匹配问题中如何有效地避免出现与出现之间的重复,判断内部重复可能产生的范围并有效地消除范围被放大的影响,这就是本文处理 SPMGOO 问题中比较复杂的环节.下面将通过介绍 MSAING 算法给出这些问题的解决方案.

## 3 MSAING 算法分析及设计

### 3.1 SAING算法

Chen 等人[8]提出了 SAIL 算法在非负间隙约束下解决在线模式匹配问题,在 NR 模式的情况下取得了完备的解,而 R 模式的情况下匹配结果较差.因此,本文在此基础上提出了 SAING 算法,在一般间隙与 one-off 条件下,使 R 模式匹配取得完备解,NR 模式匹配取得最优解.

SAING 算法包括 3 个部分:Locate phase,Forward phase,Backward phase[8].Locate 阶段查找序列串中位置 $i$,满足$\{i|s_i=p_{m-1}\}$,即模式串的尾字符 $p_{m-1}$ 在序列串中匹配的位置;Forward phase 判断能否建立搜索表;如果搜索表建立成功,则 Backward phase 找出一个最优出现.由于 Locate phase 比较简单,所以下面将分别从算法 1:Forward phase 以及算法 3:Backward phase 具体地介绍 SAING 算法.其中,Backward phase 根据一般间隙的特性需要考虑内部重复出现[13]、回溯机制提高解的完备性.

#### 3.1.1 Forward phase 算法的设计

Forward phase 主要是判断能否建立搜索表,算法返回值是布尔型.在第 1 行中,对能否成功建立搜索表的标记位进行初始化,初值为 false;第 2 行计算模式串匹配的范围,设定可达到的最大位置为 *limit*,最小位置为 *start*;第 3 行初始化一个长度为 *limit*−*start*+1 的搜索表 table,如图 2 所示.通过对搜索表进行遍历(第 4 行、第 5 行),首先找出 $p_0$ 可能出现的位置,即如果 $s_j$ 未被使用,并且与 $p_0$ 相同,则对位置 $j$ 标记(第 6 行、第 7 行);第 8 行~第 14 行找出 $p_1$~$p_{m-1}$ 的所有出现:如果 $s_j$ 未被使用,与 $p_i$ 相同,并且搜索表 table 中在$[\min_{i-1},\max_{i-1}]$范围内 $p_{i-1}$ 已被标记,则表明在间隙约束范围内有 $p_i$ 的出现,对 $p_i$ 也进行标记;只有当 $p_{m-1}$ 在 *end* 处被标记时,才完成搜索表的建立,否则无法建立搜索表(第 17 行~第 19 行).

**算法 1**. Forward phase.

输入:标志数组 *used*,$p_{m-1}$ 在序列中的位置 *end*.

输出:*flage_table*.

```
1.  falg_table=false;
2.  计算模式串匹配的范围为[start,limit];
3.  creat_table();   //创建一个长度为 len=limit−start+1 的线性表
4.  for i=0 to m−1
5.      for j=start to limit
6.          if (!used[j] and s_j=p_i and i=0) then
7.              table[i][j−start]=1;
8.          else if (!used[j] and s_j=p_i) then
9.              for k=min_{i−1} to max_{i−1}
10.                 if (table[i−1][j−k−start]=1) then
11.                     table[i][j−start]=1
12.                 end if
13.             end for
14.         end if
```

15. end for
16. end for
17. if (*table*[*m*−1][*end*−*start*]=1)
18. *flag_table*=true;
19. end if
20. return *flag_table*

文献[8]提出了使用线性表解决非负间隙模式匹配问题,并且极大减少了空间的消耗.然而,由于一般间隙模式匹配相对于传统的非负间隙问题更加复杂,因此本文重新设计了线性表的长度计算,并首次提出了利用线性表解决 SPMGOO 问题,建立的搜索表如图 2 所示.该搜索表的行数为模式串的长度,列数为模式串在序列串中可能匹配的范围的大小,即 *limit*−*start*+1,不大于序列串的长度 *n*,从而有效地节约了内存空间.

| 0 1 … *start* … | *end* … | *limit* … *n*−1 |
|---|---|---|
| *table*[0][0]… | *table*[0][*end*−*start*]… | *table*[0][*limit*−*start*]… |
| *table*[1][0]… | *table*[1][*end*−*start*]… | *table*[1][*limit*−*start*]… |
| … | … | … |
| *table*[*j*][0]… | *table*[*j*][*end*−*start*]… | *table*[*j*][*limit*−*start*]… |
| … | … | … |
| *table*[*m*−1][0]… | *table*[*m*−1][*end*−*start*]… | *table*[*m*−1][*limit*−*start*]… |

Fig.2 Search table

图 2 搜索表

3.1.2 内部重复出现检测机制

通过引理 1 的分析证明可知,一般间隙与 one-off 条件的模式匹配问题需要考虑内部重复出现的情况.因此,本文提出了内部检测机制,通过减少重复检测的次数来提高执行效率.具体检测过程如算法 2 所示.本文采用线性表的结构,并增加了一个布尔型的标志数组 *incheck*,用于标记 $p_j$ 和 $p_i$ 的出现位置是否有重复的可能(其中,0≤*j*<*i*≤*m*−1),*incheck* 的初始值为 false(第 1 行).由上述分析可知,如果模式串中间隙全为非负间隙,或者不包含重复字符则不会有内部重复的情况出现(第 2 行~第 6 行);否则,从 $p_{m-2}$ 开始依次向前查找,检查 $p_{m-2}$ 是否会与 $p_{m-1}$ 发生重复.如果 $p_{m-2}=p_{m-1}$ 并且 $\min_{m-2}<0$,则将 $p_{m-2}$ 的 *incheck* 设置为 true,否则继续向前检测 $p_{m-3}$.以此类推,直到检测完 $p_0$ 为止(第 8 行~第 15 行).如,*P*=*a*[−2,3]*c*[0,4]*a*[−1,3]*a*[−1,5]*g*[0,1]*g*,标记 *incheck*={1,0,1,0,0,0},即当匹配模式串的 0,2 位置上的字符时,需要内部重复检测.

**算法 2**. inside_Checking.

输入:模式串 *P*.

输出:标志数组 *incheck*.

1. 初始化 *incheck* 标志数组全为 false
2. if ($\forall \min_i \geqslant 0, 0 \leqslant i \leqslant m-2$) then
3. return *incheck*;
4. end if
5. if ($\forall p_i \neq p_j, 0 \leqslant i,j \leqslant m-1, i \neq j$) then
6. return *incheck*;
7. else
8. for *i*=*m*−2 downto 0
9. for *j*=*i*+1 to *m*−1
10. if $(p_i = p_j \text{ and } \sum_{k=0}^{j-i-1} \min_{i+k} + pos_{\min} \leqslant 0 \leqslant \sum_{k=0}^{j-i-1} \max_{i+k} + pos_{\max})$ then
11. *incheck*[*i*]=true;
12. break;

13. end if
14. end for
15. end for
16. end if
17. return *incheck*

3.1.3 回溯机制

一般间隙约束的模式匹配比非负间隙约束的模式匹配更为复杂,这是由于在非负间隙约束的模式匹配中,模式是顺序匹配的,即 $p_{j+1}$ 匹配的位置一定在 $p_j$ 之后;而在一般间隙约束的模式匹配中,$p_{j+1}$ 可能出现在 $p_j$ 之前.为了提高解的完备性,本文提出了回溯机制,即当 $p_{j+1}$ 的一次匹配不成功时,回溯到上一层 $p_j$ 重新选择一条路径再次进行匹配,直至回溯到 $p_{m-1}$ 时还没有找到满足条件的路径时,则结束本次匹配.具体思路如例 1 所示.

例 1: 将序列串 *S*=*aacccccccacc*,模式串 *P*=*a*[−1,2]*c*[−1,2]*c*[−1,2]*c* 转置,即得到序列串 *S*′=*ccaccccccccaa*,模式串 *p*′为 *c*[−1,2]*c*[−1,2]*c*[−1, 2]*a*,进行模式匹配.当 *end*=2 时,如图 3 所示,按照文献[8]提出的 SAIL 算法,即最左优先策略,所匹配的路径为图中点线圆形箭头的指向,则出现数为 0.这时,需要利用回溯机制回到 $p'_1$ 层重新选择,但是能和 $p'_1$ 匹配的位置只有 $s'_1$,则继续回溯到 $p'_2$ 的位置重新选择,此时, $p'_2$ 如果选择 $s'_1$,即依照实线箭头所指示的方向匹配,同理出现数为 0;再次回溯到 $p'_2$ 重新选择 $s'_3$,按照虚线箭头所示的路径进行匹配,满足一般间隙约束完成一次匹配,该出现为⟨1,0,3,2⟩.如果 $p'_2$ 选择 $s'_3$ 时也不满足条件,则继续回溯到 $p'_3$ 结束本次匹配.具体的算法设计如算法 3 所示.

| | $S'$ | $s'_0$ | $s'_1$ | $s'_2$ | $s'_3$ | $s'_4$ | $s'_5$ |
|---|---|---|---|---|---|---|---|
| $P'$ | | *c* | *c* | *A* | *c* | *c* | *C* |
| $p'_0$ | *c* | 1 | **1** | 0 | 1 | 1 | 1 |
| $p'_1$ | *c* | **1** | 1 | 0 | 1 | 1 | 1 |
| $p'_2$ | *c* | 1 | 1 | 0 | **1** | 1 | 1 |
| $p'_3$ | *a* | 0 | 0 | **1** | 0 | 0 | 0 |

Fig.3 Search table for sequence *S*′ and pattern *p*′

图 3 序列 *S*′和模式 *p*′对应的搜索表

**算法 3**. Backward phase.

输入:*table* 搜索表,$p_{m-1}$ 在序列中的位置 *end*.

输出:出现 *occ*.

1. $occ_{m-1}$=*end*
2. *back_flag*=false;
3. 计算标志数组 *incheck* //算法 2 *inside_Checking*
4. for *i*=*m*−2 to 0
5. if (!*back_flag*) then 计算 $k_{\min}$,$k_{\max}$
6. else $k_{\min}=occ_i+1$
7. end if
8. for $k=k_{\min}$ to $k_{\max}$
9. if (*table*[*i*][*k*−*start*]=1) then
10. if (*incheck*[*i*]&&内部重复) then continue; //只有当 *incheck*[*i*]为真时,才判断是否内部重复
11. else
12. $occ_i=k$
13. *back_falg*=false;
14. break;

15. end if
16. end if
17. end for
18. if ($k>k_{max}$) then
19. if ($i=m-1$) then return *occ*=NULL;
20. else then
21. *back_flag*=true;
22. $i=i+2$; //回溯机制回溯
23. end if
24. end if
25. end for
26. return *occ*;

Forward phase 完成搜索表的建立后,启动 Backward phase,按照最左优先匹配策略,返回一个出现 *occ*.首先从 $p_{m-1}$ 开始向左进行匹配,因此,第 1 行~第 3 行将 $p_{m-1}$ 在序列中的位置 *end* 存放于 $occ_{m-1}$,设置回溯标志位 *back_flag* 初值为 false,并调用算法 2 *inside_Checking* 内部重复检查,以判断模式串中需要检查的字符位置;然后遍历模式串并按照最左优先匹配的策略进行模式匹配(第 4 行~第 25 行),其中,第 5 行、第 6 行计算 $p_i$ 可以匹配的范围,$k_{min}$,$k_{max}$ 分别为 $p_i$ 可以到达的最小值和最大值,由于间隙$[min_i,max_i]$的范围不同,可分为 $min_i\geqslant 0$,$max_i<0$,$min_i<0\leqslant max_i$ 这 3 种情况计算,当需要回溯时,则回溯到 $p_{i+1}$ 位置,由上次 $p_{i+1}$ 选择的后一个节点开始;然后,在$[k_{min},k_{max}]$范围内遍历搜索表(第 8 行),如果搜索表在此范围内有标记,则需要判断 *incheck*[*i*]是否为真.

- 如果 *incheck*[*i*]为真,则需要内部检测才进行内部重复的判断,而如果内部有重复,则进入$[k_{min},k_{max}]$范围内的下一位置的判断(第 9 行、第 10 行).
- 如果 *incheck*[*i*]为假,或者 *incheck*[*i*]为真而不存在内部重复,则位置 *k* 作为 $p_i$ 匹配的位置(第 11 行~第 14 行).

如果超出$[k_{min},k_{max}]$范围还没有 $p_i$ 匹配的位置,则需要回溯机制(第 18 行~第 24 行).其中,如果已经回溯到 $p_{m-1}$,则该搜索表上的匹配不成功,出现为空(第 19 行);否则,将 *back_flag* 标记为真,回溯到 $p_{i+1}$ 重新匹配(第 20 行~第 23 行).

3.1.4 SAING 算法设计

SAING 为主程序,获得所有出现以及出现的个数,具体设计如算法 4 所示.首先对序列串 *S* 由 $s_0$ 开始扫描,当序列串中某个字符与模式串的尾字符相同时,完成 Locate 阶段(第 2 行).第 3 行利用 Forward phase 判断搜索表能否成功建立:如果能够成功建表,则启动 Backward phase,找出一个最优出现(第 4 行~第 6 行).如果 Backward phase 匹配的解不为空,则把本次查找到的出现 *occ* 加入到出现集合 *OCC* 中,并将出现中的位置标记为已使用(第 7 行~第 12 行),继续向后扫描,扫描至序列串 *S* 结束为止.

**算法 4**. SAING.

输入:$P=p_0[min_0,max_0]p_1\ldots[min_{m-2},max_{m-2}]p_{m-1}$,$S=s_0s_1\ldots s_{n-1}$.

输出:$|OCC_{SPMGOO}|$.

1. for $i$=0 to $i<n$
2. if ($s_i=p_{m-1}$) then //Locate phase
3. *flag_table=Forward phase*(); //算法 1
4. if (*flag_table*)
5. *occ=Backward phase*(); //算法 3
6. end if
7. if (*occ*)

8. $OCC_{SPMGOO}=OCC_{SPMGOO}\cup occ$
9. for $j$=0 to $m$−1
10. $used[occ_j]$=true
11. end for
12. end if
13. end if
14. end for
15. return $|OCC_{SPMGOO}|$

SAING 算法是利用线性表的结构实现一般间隙与 one-off 条件下的模式匹配,为了进一步提高匹配的完备性,本文在 SAING 算法的基础上,采用 Reverse 策略提出了 MSAING 算法.该算法通过对模式串结构以及符号集 $\Sigma$中各个元素在序列串中的频度进行分析,判断模式串是否需要转置处理,以使出现数目达到最大值.

### 3.2 Reverse策略与MSAING算法

#### 3.2.1 Reverse 策略

**定理 2**. 对序列串 $S$、模式串 $P$ 同时执行转置操作,不影响出现的正确性.

证明:给定序列串 $S=s_0\ldots s_{n-1}$,模式串 $P=p_0\ldots p_{m-1}$,同时对其执行转置操作,分别表示成 $S'=s'_0\ldots s'_{n-1}$, $P'=p'_0\ldots p'_{m-1}$,其中, $s'_0=s_{n-1},\ldots,s'_{n-1}=s_0$, $p'_0=p_{m-1},\ldots,p'_{m-1}=p_0$. 若 $P'$在 $S'$中有出现 $\langle s'_i,s'_q,\ldots,s'_k\rangle$, 由对称性可知,$P$ 在 $S$ 中必有一次对应的出现为$\langle s_{n-i-1},s_{n-q-1},\ldots,s_{n-k-1}\rangle$. □

**引理 2**. 在模式匹配的过程中,使频度较大的字符出现在模式前端,可以提高匹配的成功率.

证明:形如 $P=p_0[\min_0,\max_0]p_1\ldots[\min_{j-1},\max_{j-1}]p_j\ldots[\min_{m-2},\max_{m-2}]p_{m-1}$,当 $p_{m-1}=s_j$ 完成定位时,即 $end=j$,开始启动 Forward 阶段建立搜索表.建立搜索表的过程由 $p_0$ 开始建立,$p_0$ 出现的频率越高,可供 $p_1$ 选择的范围越大,匹配的位置越多.而 $p_1$ 匹配的位置越多,相应地,$p_2$ 选择的范围越大,匹配的位置就越多.以此类推,建立的搜索表能够形成的回溯路径越多,则 Backward 阶段可供选择的路径增加,得到的匹配解的个数也会增多. □

**引理 3**. 若模式串 $P$ 为 RT 模式,将模式串 $P$ 与序列串 $S$ 同时转置,可以提高匹配的完备性.

证明:给定序列串 $S=s_0\ldots s_{start}\ldots s_{end}\ldots s_{limit}\ldots s_{n-1}$,其中,$0\leqslant start<limit\leqslant n-1$,给定 RT 模式 $P$ 为 $P=p_0[\min_0,\max_0]\ldots p_i\ldots[\min_{m-2},\max_{m-2}]p_{m-1}$,其中,$p_i=p_{i+1}=\ldots=p_{m-1}$.假设 $p_{m-1}=s_{end}$,$p_i$ 匹配的位置为 $s_j(start\leqslant j\leqslant limit)$.如果在$[s_{start},s_{limit}]$有一个出现 $occ$,当间隙区间较小时,由于对于序列串 $S$ 继续向后扫描,$[s_j,s_{limit}]$相对集中出现了字符 $p_{m-1}$,导致$\Sigma$中的其他字符因为不满足间隙约束,在$[s_{start},s_j]$区间匹配的概率大为降低.间隙区间越小,重复的字符数目越多,匹配的完备性就越差. □

模式串 $P$、序列串 $S$ 分别转置为 $P'$,$S'$,表示为 $S'=s_{n-1}\ldots s_{limit}\ldots s_{end}\ldots s_{start}\ldots s_0$,$P'=p_{m-1}[\min_{m-2},\max_{m-2}]\ldots p_i\ldots[\min_0,\max_0]p_0$,其中,$1\leqslant i\leqslant m-1$,$p_{m-1}=p_{m-2}=\ldots=p_i$,则模式 $P'$为 RH 模式.$p_i$ 匹配的位置为 $s_j(start\leqslant j\leqslant limit)$,同理可得,$[s_{limit},s_j]$相对集中出现 $p_{m-1}$,$[s_j,s_{start}]$区间内字符分布均匀,所以在下一次匹配过程中,$[s_j,s_{start}]$区间内未被使用的位置可被其他字符使用.引理 3 得证.对于 Reverse 策略,通过例 2 进行说明.

例 2:给定序列串 $S=aaccccccacc$,模式串 $P=a[-1,2]c[-1,2]c[-1,2]c$,SAING 算法首先对模式串建立标志位数组 $incheck$={0,1,1,0};然后,在定位阶段扫描序列串 $S$,找到 $s_2=p_{m-1}=p_3$,所以 $end$=2;启动 Froward 阶段,根据模式串的间隙,计算模式串在序列串上可以匹配的范围为[0,5],$start$=0,$limit$=5,建立搜索表 $len$=6,如图 4 所示.

| | $S$ | $s_0$ | $s_1$ | $s_2$ | $s_3$ | $s_4$ | $s_5$ |
|---|---|---|---|---|---|---|---|
| $P$ | | $a$ | $a$ | $c$ | $c$ | $c$ | $c$ |
| $p_0$ | $a$ | 1 | **1** | 0 | 0 | 0 | 0 |
| $p_1$ | $c$ | 0 | 0 | 1 | 1 | **1** | 1 |
| $p_2$ | $c$ | 0 | 0 | 1 | **1** | 1 | 1 |
| $P_3$ | $c$ | 0 | 0 | **1** | 0 | 0 | 0 |

Fig.4 Search table when $end$=2

图 4 $end$=2 时的搜索表

由于 *end*=2 位置可以标记为 1,则启动 Backward 阶段匹配出最优出现,从 $p_3$=2 开始,然后按照最左优先的策略匹配,在执行 $p_2$ 时,由于 $s_0$,$s_1$ 均标记为 0,且 *incheck*[2]=1,需要内部检测是否重复,而 $s_2$ 已经被 $p_3$ 占用,所以若 $s_2$ 不满足条件则选择 $s_3$,同理可得 $p_1$ 选择 $s_4$,$p_0$ 选择 $s_1$.图 4 中,加粗、下划线的⟨1,4,3,2⟩即表示为一次出现,将序列串中的位置⟨1,4,3,2⟩标记为已使用.继续向后扫描序列,当 *end*=5 时,进入 Forward 阶段,计算 *start*=0,*limit*=8,建立 *len*=9 的搜索表,$p_0$ 在 $s_0$ 标记为 1,但由于序列中⟨1,4,3,2⟩已经被使用,所以在[−1,2]的间隙内 $p_1$ 无法标记,因此搜索表无法成功建立,不启动 Backward 阶段.继续扫描向后扫描序列,当 *end*=6 时,计算 *start*=0,*limit*=9,建立 *len*=10 的搜索表,如图 5 所示(其中带*的列,表示已被使用),序列中⟨9,8,7,6⟩的位置为一次出现.同理,继续向后扫描,若都不满足条件,则结束程序.出现集合为{⟨1,4,3,2⟩,⟨9,8,7,6⟩},$|OCC_{SPMGOO}|$=2.

| | *S* | $s_0$ | $s_1$* | $s_2$* | $s_3$* | $s_4$* | $s_5$ | $s_6$ | $s_7$ | $s_8$ | $s_9$ |
|---|---|---|---|---|---|---|---|---|---|---|---|
| *P* | | *a* | *a* | *c* | *c* | *c* | *c* | *c* | *c* | *c* | *a* |
| $p_0$ | *a* | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | **1** |
| $p_1$ | *c* | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | **1** | 0 |
| $p_2$ | *c* | 0 | 0 | 0 | 0 | 0 | 0 | 0 | **1** | 1 | 1 |
| $p_3$ | *c* | 0 | 0 | 0 | 0 | 0 | 0 | **1** | 0 | 0 | 0 |

Fig.5 Search table when *end*=6

图 5 *end*=6 时的搜索表

例 2 中的模式串 *P* 为 RT 模式,并且序列串字符“*c*”出现的频度较高,符合 Reverse 策略.因此,对序列串 *S*、模式串 *P* 同时进行转置操作,分别得到 *S*′=*ccacccccccaa*,模式串 *p*′=*c*[−1,2]*c*[−1,2]*c*[−1,2]*a*.然后,再按照 SAING 算法进行模式匹配,扫描序列串 *S*′,当 *end*=2 时,$p'_3 = s'_2$,进入 Forward 阶段,计算得到 *start*=0,*limit*=5,建立 *len*=6 的搜索表,如图 3 所示,其中,按照虚线箭头指示的匹配路径,加粗、下划线的位置⟨1,0,3,2⟩即为一次出现,并将该匹配位置标记为已使用.同理,继续扫描序列,可得到如图 6 的匹配过程,其中,带*的位置表示已被使用,实线窗口为第 1 次匹配的范围,实线箭头表示第 1 次匹配的过程,则匹配的位置为⟨4,5,7,10⟩;虚线窗口表示第 2 次匹配的范围,虚线箭头表示第 2 次匹配的过程,其匹配的位置为⟨6,8,9,11⟩.转置后,模式串 *P*′在序列串 *S*′出现的集合为 $OCC'_{SPMGOO}$ ={⟨1,0,3,2⟩,⟨4,5,7,10⟩,⟨6,8,9,11⟩},由定理 2 可知,对应模式串 *P*,在序列串 *S* 出现的集合为 $OCC_{\max}$={⟨5,3,2,0⟩,⟨7,6,4,1⟩,⟨10,11,8,9⟩},因此证明了 Reverse 策略的有效性.

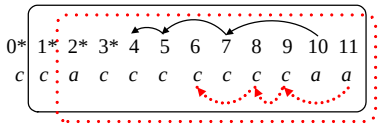



Fig.6 Pattern matching when *end*=10,11

图 6 *end*=10,11 时的模式匹配

### 3.2.2 MSAING 算法

由例 2 可知,通过对模式串 *P* 的形式分析,判断是否需要转置,可以在很大程度上提高 SAING 算法的完备性.对于给定的模式串 *P*,可分为以下几种情况来讨论.

1) 模式串 *P* 是 RT 模式不是 RH 模式,则需要对模式串和序列执行转置操作(第 1 行~第 3 行).
2) 模式串 *P* 不是 RT 模式而是 RH 模式,不需要转置..
3) 模式串 *P* 既是 RT 模式又是 RH 模式,这时需要进一步比较模式前端与后端字符重复的数目:如果后端相同字符数目较多,则需要对模式串与序列串执行转置操作;如果前端相同字符数目较多,则不需要转置;如果前端与后端相同字符数目相同,则需比较两端重复字符在序列串中的频度,如果后端频度大,则需要转置,否则不需要转置(第 4 行~第 15 行).
4) 模式串 *P* 既不是 RT 模式又不是 RH 模式,需要比较模式前端与后端第 1 个不相同的字符在序列串中出现的频度:如果后端出现的频度大,则需要转置;否则不需要转置(第 16 行~第 20 行).具体详细过程见算法 5.

**算法 5**. MSAING 算法.

输入:$P=p_0[\min_0,\max_0]p_1\ldots[\min_{m-2},\max_{m-2}]p_{m-1}$,$S=s_0s_1\ldots s_{n-1}$.

输出:最大的出现数目$|OCC_{\max}|$.

1. if (*RT*(*P*) and !*RH*(*P*)) then
2. *reverse*(*P*,*S*) //转置 *P*,*S*
3. end if
4. if (*RT*(*P*) and *RH*(*P*)) then
5. *hcount*={$i|p_0=p_1=\ldots=p_i,1\leqslant i\leqslant m-2$} //首字符重复的数目
6. *tcount*={$m-j|p_j=p_{j+1}=\ldots=p_{m-1},1\leqslant j\leqslant m-2$} //尾字符重复的数目
7. if (*tcount*>*hcount*) then
8. *reverse*(*P*,*S*)
9. end if
10. if (*tcount*=*hcount*) then
11. if (*frequence*($p_0$)<*frequence*($p_{m-1}$))
12. *reverse*(*P*,*S*)
13. end if
14. end if
15. end if
16. if (!*RT*(*P*) and !*RH*(*P*))
17. if (*frequence*($p_i$)<*frequence*($p_{m-1-i}$)) //$p_i$ 为首尾两端第 1 个不相同的字符
18. *reverse*(*P*,*S*)
19. end if
20. end if
21. 调用算法 4:SAING 算法
22. return $|OCC_{\max}|$

### 3.3 算法分析

通过对以上 5 种算法的介绍可知,MSAING 算法的时间复杂度等于 SAING 算法复杂度.SAING 算法时间主要消耗在定位阶段、Forward 阶段和 Backward 阶段.其中,定位阶段时间复杂度为 $O(n)$,主要对序列的一次完整的扫描;Forward 阶段时间复杂度为 $O(lmg)$,其中,$l=\sum_{i=0}^{m-2}\max_i-\min_i+m$ 为最坏情况下建立的搜索表长度,$m$ 为模式串 $P$ 的长度,$g=\max\{\max_i-\min_i\}(0\leqslant i\leqslant m-2)$;Backward 阶段时间复杂度为 $O(m^2g)$,因为在最差情况下,对搜索表需要全部回溯,遍历所有的路径.所以 $O(MSAING)=O(SAING)=O(n+kmg(l+m))$,其中,$k$ 表示字符 $p_{m-1}$ 在序列串 $S$ 中出现的频度.MSAING 的空间复杂度为 $O(lm)$,主要消耗在建立长为 $l$、宽为 $m$ 的搜索表上.

### 3.4 正确性和完备性

**引理 4**. 给定序列串 *S*,模式串 *P*,*occ* 为模式串 *P* 在序列串 *S* 上满足一般间隙约束的一个出现,若 *occ* 没有被 MSAING 算法作为一个出现,则 *occ* 与出现集合 *OCC* 中至少一个出现不满足 one-off 条件.

证明:为了方便证明,不妨证明其逆否命题.假设 *OCC* 为 MSAING 算法输出的出现集合,*occ* 与 *OCC* 中的任意一个出现都满足 one-off 条件,则 *occ* 一定会被 MSAING 算法作为一个出现的解.

由于 MSAING 算法在 Forward 阶段将所有满足条件的位置进行标记,Backward 阶段是由 $p_{m-1}$ 开始按照最左优先进行模式匹配,则 *occ* 中的位置为线性表中的一组匹配.由 one-off 条件的性质可知,*occ* 中的任意位置都没有被使用,则在匹配的过程中由 $occ_{m-1}$ 开始,根据假设,在匹配过程中 *occ* 未被作为出现,则一定是由于另一个出现 *occ*′选择了 *occ* 的位置,其中,$occ_i<occ'_i$ $(0\leqslant i\leqslant m-2)$.因为 *occ* 与 *occ*′都使用了 $occ_{m-1}$,所以不满足 one-off

条件,假设成立.由于逆否性质可知,引理得证. □

**定理 3**. 在 NR 模式情况下,MSAING 算法取得完备解.

证明:利用反证法进行证明.设 *OSM* 为 MSAING 算法计算的解,而 *OSC* 为完备解.假设|*OSM*|<|*OSC*|,为了简化证明不妨假设|*OSM*|=1,而|*OSC*|=2,其中,*OSM*={*O*},*OSC*={*A*,*B*}:

- $O=\{o_0,o_1,\ldots,o_{m-1}\}$;
- $A=\{a_0,a_1,\ldots,a_{m-1}\}$;
- $B=\{b_0,b_1,\ldots,b_{m-1}\}$.

若 *O* 与{*A*,*B*}满足 one-off 条件,则|*OSC*|=3,与假设矛盾;由引理 4 知,*O* 与 *A*、*O* 与 *B* 不满足 one-off 条件.

所以,$\exists 0\leqslant u,w,i,k<m$,使得 $a_u=o_i,b_w=o_k$ (i)

则 $s_{a_u}=s_{o_i},s_{b_w}=s_{o_k}$.由出现的定义可知,$s_{a_u}=p_u,s_{o_i}=p_i,s_{b_w}=p_w,s_{o_k}=p_k$.

因此,$p_u=p_i,p_w=p_k$.

下面分两种情况分别加以讨论.

1. 当 $u\neq i$ 或者 $w\neq k$ 时,模式 *P* 为 R 模式,与 *P* 为 NR 模式条件相矛盾.
2. 当 $u=i,w=k$ 时,下面再分为情况(1)、情况(2)两种情况来讨论.

   (1) 当 $u=w$ 时,由(i)可知,$a_u=o_i=o_u,b_w=o_k=o_w$,而 $u=w$,则 $a_u=o_i=o_k=b_w$.

   所以,当 $u=w$ 时,出现 *A* 与出现 *B* 不满足 one-off 条件,与假设相矛盾.

   (2) 当 $u\neq w$ 时,为了不失一般性,不妨假设 $u<w$,

   由于 *O* 是 MSAING 算法采用最左优先策略匹配的解,

   所以,$o_u=b_u,o_w=a_w$. (ii)

   所以,由(i)、(ii)得:$a_u<b_u,b_w<a_w$,则出现 *A*,*B* 可以表示为如下的形式:

   $$A=....a_u.....a_w...,$$
   $$B=.....b_u.....b_w...$$

   所以,$\exists t,u\leqslant t<m$ 满足下面的条件;又由于模式 *P* 为一般间隙约束,则可以分为形式①~形式④这 4 种形式进行分析:

   ① $a_t<b_t<b_{t+1}<a_{t+1}$,则 $b_{t+1}-b_t<a_{t+1}-b_t<a_{t+1}-a_t$;

   ② $a_t<b_{t+1}<b_t<a_{t+1}$,则 $b_{t+1}-b_t<a_{t+1}-b_t<a_{t+1}-a_t$;

   ③ $a_{t+1}<b_t<b_{t+1}<a_t$,则 $a_{t+1}-a_t<a_{t+1}-b_t<b_{t+1}-b_t$;

   ④ $a_{t+1}<b_{t+1}<b_t<a_t$,则 $a_{t+1}-a_t<a_{t+1}-b_t<b_{t+1}-b_t$.

上述 4 种情况表明,$a_{t+1},b_t$ 之间满足间隙约束条件,所以,$\langle b_0,\ldots,b_u,\ldots,b_t,a_{t+1},\ldots,a_w,\ldots,a_{m-1}\rangle$是模式串 *P* 在序列串 *S* 上,与出现 *O* 满足间隙约束与 one-off 条件的另一个出现,而与引理 4 相矛盾.故假设不成立,定理 3 得证.□

## 4 实验结果及分析

### 4.1 实验环境

本节采用真实生物数据对比 SAIL_Gen,RSAIL_Gen,SGSP_Gen,SBO_Gen,RBCT_Gen,DCNP 和 MSAING 算法的性能,其中,前 4 种算法分别为在一般间隙约束下对文献[8,10,13,20,24]中算法 SAIL,RSAIL,SGSP,SBO,RBC 的改进,即不改变算法的执行步骤,只是扩展间隙约束的范围,使其可以在一般间隙约束条件下进行模式匹配;DCNP 来自于文献[13].实验运行的软、硬件环境为:Intel(R) Core(TM)i3-4170 CPU@3.70GHz,4.0GB 内存,操作系统为 Window7,64 位操作系统,程序使用 C++语言编写,并且采用 Microsoft Visual Studio 2013 集成开发环境进行编译和运行.为了验证算法的性能,本文采用文献[13]所使用的真实数据——序列数据库 SDB1 作为测试序列.同时,为了更好地验证算法在实际应用中大规模的数据上的性能,实验还分别采用了表 2 中描述的序列数据库 SDB2,SDB3,SDB4,其中,SDB2,SDB3 在文献[25]中作为大规模的数据集,用于算法性能检测.表 2 中所有的 DNA 片段、DNA 序列库都可以从美国国家生物计算信息中心的网站下载(http://www.ncbi.nlm.nih.gov/).蛋

白质序列库可以从 http://gi.cebitec.uni-bielefeld.de/comet/force/indexOld.html 上下载.表 3 描述的是文献[13]中的序列数据库 SDB1 包含的 12 个序列的特征.

**Table 2** Characteristic of experimental data

**表 2** 实验数据集特征

| 序列数据库 | 数据集名称 | 序列类型 | 序列个数 | 总长度 |
|---|---|---|---|---|
| SDB1 | DNA 片段 | DNA 片段 | 12 | 327 535 |
| SDB2 | WO02059377 | DNA 序列 | 70 | 190 533 |
| SDB3 | ASTRAL95_1_161 | 蛋白质序列 | 507 | 91 875 |
| SDB4 | 文献[26] | 文本字符串 | 1 | 8 191 |

**Table 3** Sequences of real biological data in SDB1

**表 3** SDB1 真实生物数据片段

| 序号 | 位点 | 片段长度 | 序号 | 位点 | 片段长度 |
|---|---|---|---|---|---|
| $S_1$ | CY058560 | 844 | $S_7$ | CY058563 | 2 286 |
| $S_2$ | CY058557 | 982 | $S_8$ | CY058562 | 2 299 |
| $S_3$ | CY058558 | 1 418 | $S_9$ | AX829178 | 5 393 |
| $S_4$ | CY058559 | 1 516 | $S_{10}$ | AX829174 | 10 011 |
| $S_5$ | CY058556 | 1 720 | $S_{11}$ | AB038490 | 131 892 |
| $S_6$ | CY058561 | 2 169 | $S_{12}$ | AL158070 | 167 005 |

### 4.2 实验结果及分析

#### 4.2.1 DNA 片段上算法的比较

DNA 控制 RNA 的转录以及遗传信息,因此,通过对 DNA 片段进行特定的模式匹配,找出匹配解的个数,对于致病基因以及遗传信息的检测、病毒传播的预防等起着重要的作用.一般间隙模式匹配有利于灵活地找出模式匹配个数的精确解,通过对匹配解的数目分析,对生物学中基因遗传的研究具有重要的价值.下面将具体介绍 MSAING 以及对比算法在 DNA 片段上的性能.为了分别对比测试 MSAING 算法在模式串不存在重复字符以及存在重复字符时的求解性能,选取了 $P_1$~$P_4$ 模式为不存在重复字符的 4 种模式,$P_5$~$P_9$ 模式为存在重复字符的 5 种模式,具体模式串在表 4 中给出.这 7 种算法的模式串的测试结果见表 5,其中给出了各种算法在不同模式下得到的出现数的总和;表 6 给出了各种算法在不同模式以及序列下的消耗时间总和.

**Table 4** Patterns

**表 4** 模式串

| 序号 | 模式串 |
|---|---|
| $P_1$ | $a[-5,6]c[-4,7]g[-3,8]t$ |
| $P_2$ | $c[-1,2]a[-2,3]t[-3,4]g$ |
| $P_3$ | $g[1,2]t[0,3]c[-3,4]a$ |
| $P_4$ | $t[-2,2]c[-2,2]a$ |
| $P_5$ | $g[-1,5]t[0,6]a[-2,7]g[-3,9]t[-2,5]a[-4,9]g[-1,8]t[-2,9]a$ |
| $P_6$ | $g[-1,5]t[0,6]a[-2,7]g[-3,9]t[-2,5]a[-4,9]g[-1,8]t[-2,9]a[-1,9]g[-1,9]t$ |
| $P_7$ | $t[-1,7]t[-1,7]a[-1,7]g[-1,7]t[-1,7]a[-1,7]g$ |
| $P_8$ | $a[-1,7]a[-1,7]a[-1,7]a[-1,7]c$ |
| $P_9$ | $c[-1,7]a[-1,7]a[-1,7]a[-1,7]a$ |

**Table 5** Comparison of the number of occurrences of $P_1$~$P_9$ between 7 algorithms

**表 5** 7 种算法在 $P_1$~$P_9$ 上的出现个数

| 算法名称 | 模式串中不存在重复字符 | | | | 模式串中存在重复字符 | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | $p_1$ | $p_2$ | $p_3$ | $p_4$ | $p_5$ | $p_6$ | $p_7$ | $p_8$ | $p_9$ |
| SAIL_Gen | 36 327 | 23 885 | 13 257 | 31 132 | 10 961 | 8 289 | 17 790 | 18 192 | 18 401 |
| RSAIL_Gen | 36 327 | 23 885 | 13 257 | 31 132 | 10 961 | 8 289 | 17 790 | 18 192 | 18 342 |
| SGSP_Gen | 35 876 | 23 520 | 13 238 | 30 667 | 13 548 | 10 004 | 17 766 | 18 096 | 19 033 |
| SBO_Gen | 36 267 | 23 793 | 13 250 | 30 984 | 14 101 | 10 528 | 18 056 | 18 831 | 19 204 |
| RBCT_Gen | 36 253 | 23 763 | 13 213 | 30 947 | 14 125 | 10 593 | 18 052 | 18 822 | 19 433 |
| DCNP | 36 267 | 23 793 | 13 250 | 30 984 | 14 152 | 10 673 | 18 070 | 18 865 | 19 898 |
| MSAING | 36 327 | 23 885 | 13 257 | 31 132 | 14 158 | 10 786 | 18 094 | 20 136 | 20 166 |

**Table 6** Comparison of the running time on $P_1$~$P_9$ between 7 algorithms (s)

**表 6** 7 种算法在 $P_1$~$P_9$ 上的运行时间 (s)

| 算法名称 | 模式串中不存在重复字符 | | | | 模式串中存在重复字符 | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | $p_1$ | $p_2$ | $p_3$ | $p_4$ | $p_5$ | $p_6$ | $p_7$ | $p_8$ | $p_9$ |
| SAIL_Gen | 0.98 | 0.93 | 0.85 | 0.82 | 4.17 | 5.54 | 2.67 | 2.03 | 2.05 |
| RSAIL_Gen | 1.26 | 1.17 | 1.98 | 1.89 | 4.21 | 6.36 | 3.23 | 3.18 | 3.56 |
| SGSP_Gen | 640.9 | 521.9 | 491.21 | 166.3 | 2 759.1 | 3 175.2 | 1 796.7 | 1 243.8 | 868.96 |
| SBO_Gen | 838.5 | 790.9 | 720.3 | 315.0 | 3 163.3 | 3 439 | 1 796.7 | 1 552.1 | 478.84 |
| RBCT_Gen | 0.82 | 0.78 | 0.83 | 0.89 | 3.25 | 4.12 | 1.05 | 1.14 | 1.39 |
| DCNP | 852 | 802.4 | 781.71 | 361.1 | 3 562.2 | 3 887.4 | 2 065.6 | 1 635.7 | 1 208.6 |
| MSAING | 0.93 | 0.83 | 0.81 | 0.8 | 7.2 | 9.69 | 3.14 | 2.1 | 3.37 |

(1) 在模式串中不存在重复字符的情况下,MSAING,SAIL_Gen,RSAIL_Gen 均属于完备算法.文献[8,24]证明了这种情况,从表 5 中可以看出,MSAING 能够达到完备的情况,与定理 2 的理论分析相一致.

(2) 在模式串中有重复字符的情况下,MSAING 算法性能最好,DCNP 算法性能次之,SAIL_Gen 算法性能最差.$p_5$~$p_9$ 模式串中具有重复的字符,其中,$p_8$ 为首部重复模式,$p_9$ 为尾部重复模式,从表 5 可以清晰地看到,SAIL-Gen 不能在任何实例上取得最好的结果;而且模式串 $p_5$~$p_9$ 在 12 个序列的出现总和中,SAIL-Gen 算法取得了最小值,占 MSAING 算法匹配数的 88%,而 DCNP 算法获得了次优匹配,占 MSAING 算法匹配数的 97%.这充分说明,对于 R 模式不宜采用 SAIL-Gen 算法进行求解.DCNP 算法在处理 RH 模式和 RT 模式上完备性较低,产生这种现象的原因是:DCNP 算法在 SBO_Gen 算法的基础上动态更新节点属性,采取每次选择出现相关数较少的策略,因而其可以很好地解决 one-off 条件下一般间隙的模式匹配问题.但是,由于只是局部最优的贪婪算法,因此未必能够达到全局最优,这种缺陷在连续重复字符的模式串,如 RH 和 RT 模式串上表现更为明显.而 MSAING 算法通过最左侧匹配以及回溯方法使每次匹配达到局部最优,同时采用内部检测机制避免内部重复,利用 Reverse 策略对模式串的结构以及符号集$\Sigma$中各个元素在序列串中的频度进行分析,使其达到最佳的匹配状态,保证匹配达到全局最优.

(3) 根据表 6 时间消耗可知,DCNP 算法消耗时间最长,RBCT_Gen 算法消耗时间最少;而在不存在重复字符的模式串中,MSAING 算法消耗的时间次之;在存在重复字符的模式串中,SAIL-Gen 算法消耗的时间次之.SAIL 是文献[8]提出的一种在线算法,具有最好的时间性能来解决非间隙模式匹配问题,SAIL-Gen 和 MSAING 算法继承了这种特性,当模式串不存在重复字符,进行一般间隙模式匹配时,SAIL-Gen 需要判断是否内部重复,消耗了大量的时间,而 MSAING 算法具有内部检查机制使时间消耗减少;在存在重复字符的模式串匹配中,MSAING 算法为了提高解的完备性,增加了回溯的功能,使消耗的时间变多.虽然 SAIL-Gen 耗时较少,但是基于一般性模式匹配问题时,求解性能很差,因此不宜采用 SAIL-Gen 算法进行求解.同理,RBCT_Gen 利用 CluTree 结构,在匹配的过程中减少了时间的消耗,但是相对于 MSAING 和 DCNP 算法,求解性能较差.DCNP 算法在 SBO_Gen 算法的基础上动态更新节点属性,因此所用的时间最长.综上所述,MSAING 算法解的性能优于 DCNP 和 SBO_Gen 算法.

通过将 MSAING 算法与其他各种算法在 SDB1 数据集上进行对比可知:MSAING 算法的匹配解比 DCNP 提高了 2%,比 SAIL_Gen 提高了 12%.MSAING 算法是通过 Reverse 策略提高匹配解的数目的,下面将通过实验进一步验证 Reverse 机制的有效性.

4.2.2 Reverse 机制有效性验证

为了验证 Reverse 机制的有效性,在数据集 SDB1 上选取需要转置的模式串 $p_5$,$p_6$,$p_9$,计算每个模式串在 12 条 DNA 片段上解的和.由图 7 可知,MSAING 算法转置后的解的完备性较高,说明了 Reverse 机制的有效性.这是由于 MSAING 算法通过对模式串结构的分析以及序列串中个字符元素的频度的计算,找出最佳的模式匹配形式,使解的完备性得到了提高.

本实验验证了 Reverse 策略的有效性,使匹配解的平均数目提高了 5.6%.下面将通过在序列数据库中比较算法的性能,证明 MSAING 算法能够在较大规模的数据库上进行有效的模式匹配.

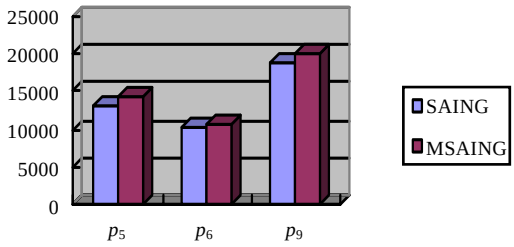



Fig.7 Validation of effectiveness of Reverse strategy

图 7 Reverse 机制有效性验证

4.2.3 序列库中算法性能的比较

随着蛋白质产品在人们日常生活中应用的普及,蛋白质检测在生物学领域得到了广泛的研究.食品中营养蛋白的检测、药物中蛋白酶以及胰岛素的检测以及蛋白质中氨基酸序列的检测等,都需要对蛋白质序列进行某些特定的模式串匹配.一般间隙的模式匹配能够灵活地找出匹配解的数量,在生物学中某种蛋白质含量的监测中有着重要的应用.下面的实验将比较 MSAING 与其他算法在 DNA 库以及蛋白质库中的性能.

为了更有效地对比各种算法匹配解的个数以及运行的速度,在 DNA 序列库 WO02059377、蛋白质序列库 ASTRAL95_1_161 上做了对比实验.

在 DNA 序列库上的模式匹配,模式串为表 3 中的 $p_1$~$p_9$,计算每个模式串在整个序列库中解的和.由图 8 可知:在 NR 模式串 $p_1$~$p_4$ 的匹配中,SAIL_Gen,MSAING 取得了完备解;在 R 模式串 $p_5$~$p_9$ 中,MSAING 取得了最优解,而 SAIL_Gen 算法解的完备性最差.这是由于 SAIL_Gen 算法只适合在线的模式匹配,在离线的情况下仅利用最左优先策略,而没有考虑回溯的情况,导致完备性较差.图 9 中可以看出:MSAING,SAIL_Gen 和 RBCT_Gen 算法的运行消耗的时间都比较少,而 DCNP 消耗的时间最多.这是由于 DCNP,SBO_Gen 都需要建立网树结构,消耗了大量的时间.

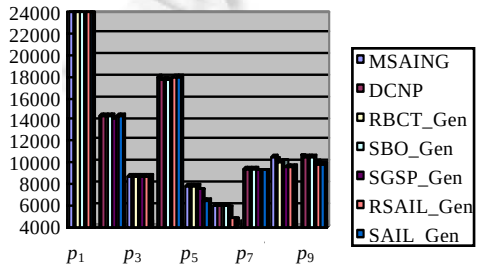



Fig.8 Comparison of the number of occurrences on DNA sequence database

图 8 DNA 序列库上出现的数目对比

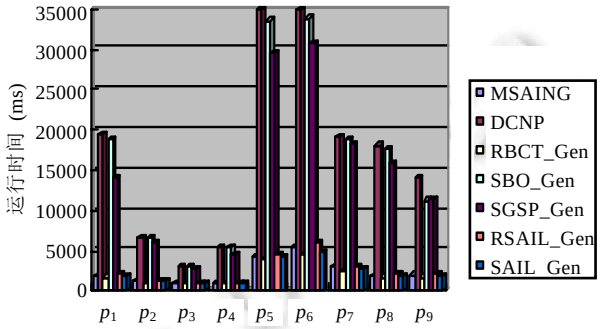
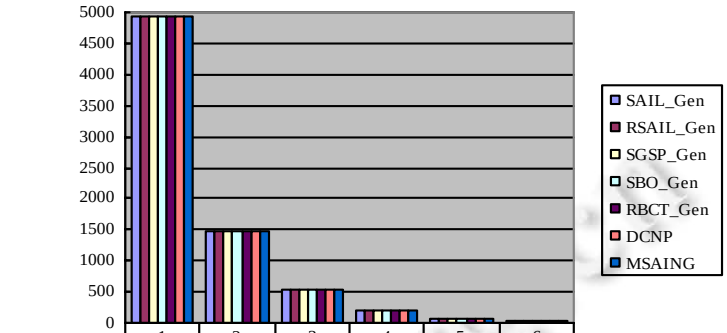



Fig.9 Comparison of the running time on DNA sequence database

图 9 DNA 序列库上运行时间对比

在蛋白质序列库上,分别选择长度为 1~6、间隙为[–2,7]的模式串,其中,长度为 1 模式串为{*a*,*c*,*d*,*e*,*f*},长度为 2 模式串是由长度为 1 的字符组合而成的 25 条模式串,模式长度为 3 的模式串 125 条,以此类推,模式串长度为 6 的 15 625 条模式串;然后,求相同长度的模式串在蛋白质序列库上出现解的和,并计算其平均值.通过图 10 可知:随着模式长度的变大,每个模式串出现的平均个数在减小.这是由于长度为($m$+1)的模式串是在长度为 $m$ 的模式串的基础上满足 $p_m$ 的匹配.因此,模式越长,出现解的概率越低.其中,MSAING 算法的完备性最高,DCNP 算法的完备性次之.图 11 中比较的是各种算法的完备性,图中纵坐标是各种算法解的数目比上 MSAING 算法解的数目所获得的百分数.随着模式长度的增加,MSAING 算法的完备性相对于其对比算法的完备性越来越高.这是由于随着模式长度的增加,模式串中含有重复的字符越来越多,DCNP,RBCT_Gen,SBO_Gen,SGSP_Gen,RSAIL_

Gen,SAIL_Gen 算法只是利用局部最优策略,而没有考虑到模式串的结构与序列串中各个字符出现的频度之间的关系,导致模式串长度越大,解的完备性就越差.图 12 为每条模式串在蛋白质序列库上运行的平均时间,可以看出:随着模式越来越长,模式匹配消耗的时间越来越多.其中,RBCT_Gen,SAIL_Gen,MSAING 增长速度比较缓慢;而 DCNP,SBO_Gen,SGSP_Gen 增长速度较快,消耗时间较多.




| | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| SAIL_Gen | 4936.2 | 1467.32 | 524.408 | 198.0768 | 76.09248 | 29.44774 |
| RSAIL_Gen | 4936.2 | 1469.21 | 524.413 | 199.635 | 78.02596 | 30.02466 |
| SGSP_Gen | 4936.2 | 1470.12 | 524.696 | 200.8736 | 79.45216 | 31.88486 |
| SBO_Gen | 4936.2 | 1470.36 | 526.424 | 201.5344 | 79.64032 | 31.92928 |
| RBCT_Gen | 4936.2 | 1470.36 | 526.703 | 201.624 | 79.65146 | 31.94521 |
| DCNP | 4936.2 | 1470.36 | 526.728 | 201.6864 | 79.66912 | 31.96653 |
| MSAING | 4936.2 | 1470.72 | 527.088 | 202.024 | 80.35456 | 32.50893 |

Fig.10 Comparison of the number of occurrences on protein sequence database

图 10 在蛋白质序列库上出现的对比

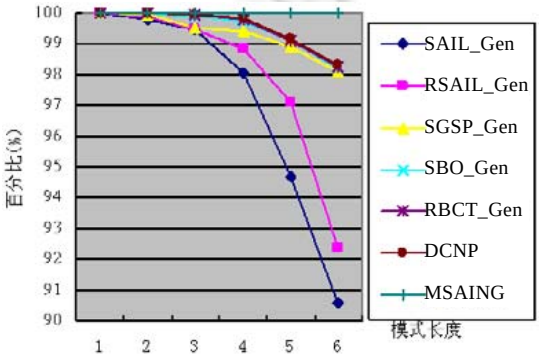



Fig.11 Percentage of each algorithm with MSAING on protein sequence database

图 11 在蛋白质序列库上各算法的出现与 MSAING 算法的百分比

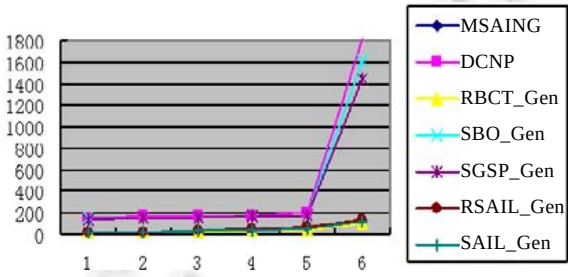



Fig.12 Comparison of the running time on protein sequence database

图 12 在蛋白质序列库中运行时间对比

在生物序列上的实验,验证了 MASING 算法进行较大规模数据库模式匹配时的性能.为了进一步证明该算法解决某些实际问题的有效性,本文把它应用在文本信息的检索中.

4.2.4 文本信息检索中的应用

在文本信息检索中考虑一般间隙更具有实际意义,例如在文献[26]中,出现 frequent closed subsequence 与带负间隔的模式 closed frequent subsequence 表示相同的意思.通过统计文本中模式出现的次数,可以实现文本中关键词的抽取.带一般间隙的模式匹配将文本中单词、短语视为模式串,通过提高文本中模式出现解的完备性,

从而可以进一步提高关键词抽取的精确度,在文本信息检索的过程中获得更多有价值的信息.

以文献[26]作为信息检索文本,分别检索 data mining,closed subsequence,frequent closed subsequence 在文本中出现的次数,由表 7 可知:一般间隙的模式匹配出现的次数更多,更具有灵活性.为了更好地验证一般间隙的性质,计算在文献[26]中所有长度为 2 的模式串出现的次数之和,非负间隙将给定的间隙的最小值设为 0.从图 15 可以看出:随着模式串的间隙的增大,解的个数不断增多,一般间隙比非负间隙获得更多的解.这是由于一般间隙在非负间隙的基础上又考虑了负间隙的情况,即模式串中字符的顺序发生颠倒的情况;同时,间隙越大,满足匹配出现的概率也越大,解的个数也就越多.

**Table 7** Comparison of the number of occurrences about no-negative gap with general gap

**表 7** 非负间隙与一般间隙出现个数比较

| 模式 | 非负间隙 | 一般间隙 |
|---|---|---|
| Data mining | 2 | 3 |
| Closed subsequence | 3 | 4 |
| Frequent closed subsequence | 4 | 9 |

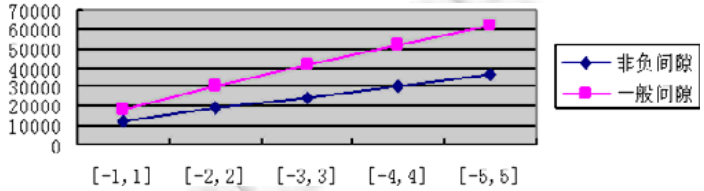



Fig.13 Comparison of the number of occurrences about no-negative gap with general gap

图 13 非负间隙与一般间隙解的出现对比

## 5 结 论

本文提出了一般间隙与 one-off 条件约束的模式匹配问题——SPMGOO 问题.该问题的研究允许用户更加灵活地设定模式串.该问题具有如下特点:间隙可以为负;同时,序列串中任何字符最多只能使用 1 次.本文把线性表应用于该问题的求解,并基于线性表的结构提出了 MSAING 算法.算法首先采用 Reverse 策略使模式与序列达到最佳的匹配状态,克服了某些算法容易陷入局部最优的缺点;其次,利用线性表的结构使匹配过程中的时间和空间消耗大为减少,并利用回溯的方法提高匹配的成功率;最后,根据 *inside_Checking* 机制判断模式串内部是否会产生重复现象,有效提高算法运行效率.最后,本文从理论和实验两个方面验证了 MSANIG 算法匹配的有效性.

本文只是对一般间隙与 one-off 条件的模式匹配问题进行了研究,而模式匹配是序列模式挖掘的基础,因此,下一步将对一般间隙的序列模式挖掘问题进行研究.该研究将有助于挖掘更多有价值的频繁模式.此外,在实际应用中有很多模式匹配是近似模式匹配,这不但具有实际意义,而且研究难度也会更大,这些问题均是未来研究的方向.

**References**:

[1] Wu XD, Zhu XQ, He Y, Arslan AN. PMBC: Pattern mining from biological sequences with wildcard constraints. Computers in Biologyand Medicine, 2013,43(5):481−492. [doi: 10.1016/j.compbiomed.2013.02.006]

[2] Zhang C, Zheng Y, Ma XL, Han JW. Assembler: Efficient discovery of spatial co-evolving patterns in massive geo-sensory data. In: Proc. of the 21th ACM SIGKDD Int'l Conf. on Knowledge Discovery and Data Mining. New York: ACM Press, 2015. 1415−1424. [doi: 10.1145/2783258.2783394]

[3] Shokoohi-Yekta M, Chen YP, Campana B, Hu B, Zakaria J, Keogh E. Discovery of meaningful rules in time series. In: Proc. of the ACM SIGKDD Int'l Conf. 2015. 1085−1094. [doi: 10.1145/2783258.2783306]

[4] Chou CP, Jea KF, Liao HH. A syntactic approach to twig-query matching on XML streams. Journal of Systems and Software, 2011, 84(6):993−1007. [doi: 10.1016/j.jss.2011.01.033]

[5] Manber U, Baeza-Yates R. An algorithm for string matching with a sequence of don't cares. Information Processing Leters, 1991, 37(3):133−136. [doi: 10.1016/0020-0190(91)90032-d]

[6] Bille P, Gørtz IL, Vildhøj HW, Wind DK. String matchingwith variable length gaps. In: Chavez E, *et al*., eds. Proc. of the 17th Int'l Conf. on String Processing and Information Retrieval. Berlin: Springer-Verlag, 2010. 385−394. [doi: 10.1007/978-3-642-16321-0_40]

[7] Fischer MJ, Paterson MS. String matching and other products. In: Proc. of the String-Matching and Other Products. Cambridge: Massachusetts Institute of Technology, 1973. 113−125.

[8] Chen G, Wu XD, Zhu XQ, Arslan AN, He Y. Efficient string matching with wildcards and length constraints. Knowledge and Information Systems, 2006,10(4):399−419. [doi: 10.1007/s10115-006-0016-8]

[9] Zhu XQ, Wu XD. Discovering relational patternsacross multiple databases. In: Proc. of the IEEE 23rd Int'l Conf. on Data Engineering. 2007. 726−735. [doi: 10.1109/ICDE.2007.367918]

[10] Liu YL, Wu XD, Hu XG, Gao J. A matching algorithm in PMWL based on CluTree. New Generation Computing, 2014,32(2): 95−122. [doi: 10.1007/s00354-014-0201-3]

[11] Fredriksson K, Grabowski S. Efficient algorithms for pattern matching with general gaps and character classed. In: Crestani F, *et al*., eds. Proc. of the Int'l Conf. on String Processing and Information Retrieval. Berlin: Springer-Verlag, 2006. 267−278. [doi: 10.1007/11880561_22]

[12] Fredriksson K, Grabowski S. Efficient algorithms for pattern matching with general gaps, character classes, and transposition invariance. Information Retrieval, 2008,11(4):335−357. [doi: 10.1007/s10791-008-9054-z]

[13] Chai X, Jia XF, Wu YX, Jiang H, Wu XD. Strict pattern matching with general gaps and one-off condition. Ruan Jian Xue Bao/ Journal of Software, 2015,26(5):1096−1112 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/4707.htm [doi: 10.13328/j.cnki.jos.004707]

[14] Wu YX, Fu S, Jiang H, Wu XD. Strict approximate pattern matching with general gaps. Applied Intelligence, 2014,42(3): 1−15. [doi: 10.1007/s10489-014-0612-3]

[15] Wu YX, Liu YW, Guo L, Wu XD. Subnettrees for strict pattern matching with general gaps and length constraints. Ruan Jian Xue Bao/Journal of Software, 2013,24(5):915−932 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/4381.htm [doi: 10.3724/SP.J.1001.2013.04381]

[16] Kalai A. Efficient pattern-matching with don't cares. In: Proc. of the 13th ACM-SIAM Symp. on Discrete Algorithms. Society for Industrial and Applied Mathematics, 2002. 655−656.

[17] Kucherov G, Rusinowitch M. Matching a set of strings with variable length don't cares. In: Nivat M, *et al*., eds. Proc. of the Theoretical Computer Science. Berlin: Springer-Verlag, 1995. 230−247. [doi: 10.1007/3-540-60044-2_46]

[18] Wu YX, Wang LL, Ren JD, Ding W, Wu XD. Mining sequential patterns with periodic wildcard gaps. Applied Intelligence, 2014, 41(1):99−116. [doi: 10.1007/s10489-013-0499-4]

[19] Myers EW. Approximate matching of network expressions with spacers. Journal of Computational Biology, 2009,3(1):33−51. [doi: 10.1089/cmb.1996.3.33]

[20] Wu YX, Wu XD, Jiang H, Min F. A heuristic algorithm for MPMGOOC. Chinese Journal of Computers, 2011,34(8):1452−1462 (in Chinese with English abstract). [doi: 10.3724/SP.J.1016.2011.01452]

[21] Lam H, Morchen F, Fradkin D, Calders T. Mining compressing sequential patterns. Statistical Analysis and Data Mining, 2012,7(1): 34−52. [doi: 10.1002/sam.11192]

[22] He D, Wu XD, Zhu XQ. SAIL-APPROX: An efficient on-line algorithm for approximate pattern matching with wildcards and length constraints. In: Proc. of the 2007 IEEE Int'l Conf. on Bioinformatics and Biomedicine. Washington: IEEE Computer Society, 2007. 151−158. [doi:10.1109/BIBM.2007.48]

[23] Ding B, Lo D, Han JW, Khoo SC. Efficient mining of closed repetitive gapped subsequences from a sequence database. In: Proc. of the 2009 IEEE Int'l Conf. on Data Engineering. Washington: IEEE Computer Society, 2009. 1024−1035. [doi: 10.1109/ICDE.2009.104]

[24] Wang HP, Xie F, Hu XG, Li PP, Wu XD. Pattern matching with flexible wildcards and recurring characters. In: Proc. of the 2010 IEEE Int'l Conf. on Granular Computing. Washington: IEEE Computer Society, 2010. 782−786. [doi: 10.1109/GrC.2010.156]

[25] Zhou K. Mining sequential patterns with periodic general gap constraints [MS. Thesis]. Shijiazhuang: Hebei University of Technology, 2014 (in Chinese with English abstract).

[26] Li C, Wang JY. Efficiently mining closed subsequences with gap constraints. In: Proc. of the SIAM Int'l Conf. on Data Mining. 2008. 313−322. [doi: 10.1137/1.9781611972788.28]

## 附中文参考文献:

[13] 柴欣,贾晓菲,武优西,江贺,吴信东.一般间隙及一次性条件的严格模式匹配.软件学报,2015,26(5):1096−1112. http://www.jos.org.cn/1000-9825/4707.htm [doi: 10.13328/j.cnki.jos.004707]

[15] 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. http://www.jos.org.cn/1000-9825/4381.htm [doi: 10.3724/SP.J.1001.2013.04381]

[20] 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. [doi: 10.3724/SP.J.1016.2011.01452]

[25] 周坤.一般周期间隙约束的序列模式挖掘[硕士学位论文].石家庄:河北工业大学,2014.

**刘慧婷**(1978－),女,安徽阜阳人,博士,副教授,CCF 专业会员,主要研究领域为数据挖掘,机器学习.

**黄厚柱**(1991－),男,硕士,主要研究领域为模式匹配,序列挖掘.

**刘志中**(1990－),男,硕士,主要研究领域为模式匹配,序列挖掘.

**吴信东**(1963－),男,博士,教授,博士生导师,主要研究领域为数据挖掘,基于知识的系统,万维网络信息探索.