基于QSA数组计算序列中所有NE重复模式的算法

木妮娜·玉素甫[1] 古丽娜·玉素甫[2] 张海军[1]
(新疆师范大学计算机科学技术学院 乌鲁木齐 830054)[1]
(新疆师范大学教育科学学院 乌鲁木齐 830054)[2]

摘 要 序列中重复模式的识别与提取算法在数据挖掘、模式识别、数据压缩、生物信息学等领域中具有广泛的实际应用。提出一种全新的基于QSA数组计算所有带有约束条件的NE重复模式的算法RPT。算法设计中充分考虑了NE重复模式的特征,以建立特征和重复模式检测结果之间的统计联系;算法中的约束条件包括最小周期 p_{min} 和最大间距 g_{max},其可用于筛选符合条件的NE重复模式,并可按照递增序输出所有NE重复模式的出现位置。与已有的基于后缀索引的算法相比,此算法的空间效率得到了提高。在分类属性数据样本集上进行的实验表明,算法RPT对生物序列尤其是DNA序列以及维吾尔语Web文本中NE重复模式的识别与提取都很有效。

关键词 重复模式,数据挖掘,统计特征,约束条件,生物计算,维吾尔语Web文本
中图法分类号 TP391 **文献标识码** A

Algorithm for Computing All NE Repeats in Sequence Based on QSA Array

Munina YUSUFU[1] Gulina YUSUFU[2] ZHANG Hai-jun[1]
(School of Computer Science and Technology, Xinjiang Normal University, Urumqi 830054, China)[1]
(School of Education Science, Xinjiang Normal University, Urumqi 830054, China)[2]

Abstract The sequence repeating pattern recognition and extraction algorithms have a wide range of practical applications in the fields of data mining, pattern recognition, data compression and bioinformatics. This paper proposed a new algorithm RPT for computing all NE repeats with constraints based on QSA array. The characteristics of the NE repeats are fully considered in the algorithm design procedure for creating a statistical relation between the features and the repeats detection results. Algorithm constraints include minimum period p_{min} and maximum gap g_{max} which are used to filter the qualified NE repeats, and algorithm can output all occurrences of the NE repeats in ascending order. Compared with the existing algorithm based on suffix index, the space efficiency of the algorithm is improved. The experiments on classified data sets show that the algorithm RPT is effective for recognizing and extracting the NE repeats on biological sequences, in particular DNA sequences, as well as Uyghur Web texts.

Keywords Repeats, Data mining, Statistical characteristics, Constraints, Bioinformatics, Uyghur Web texts

1 引言

序列中重复模式的识别与提取算法的研究涉及到很多相关的计算机学科领域知识,如组合数学、图论、自动机理论及机器学习等,具有重要的理论研究价值,并在数据挖掘、模式识别、数据压缩、生物信息学、自然语言处理等领域中具有极其广泛的实际应用。

序列中的重复模式又称为重复片段、重复子串、重复序列、频繁模式(重复模式+设定阈值)。查找序列中的重复模式,首先要形式化地定义重复模式,而后是查找满足定义重复模式。G. Benson[1]曾指出重复模式查找的一个难题是如何准确地定义它们。重复模式的定义不同会导致查询的结果不同,查询的时间复杂度也会随之变化。

序列中的精确重复模式(Repeats)根据出现位置不同,可以分为:串联(Repetitions or Tandem Repeats)、分隔(Split)、交叉(Overlapping)的重复模式,由此又衍生出其它重复模式定义,比如最大串联重复模式(Runs)、不可扩展重复模式(Nonextendible Repeats,简称NE重复模式)或最大重复模式(Maximal Repeats)等。

生物序列中含有大量的各种重复片断。例如,在人类基因组的约 3.2×10^9 个碱基对中超过50%已被识别为各种重复元素[2]。重复片断识别对于生物序列的特征分析与研究非常重要,现已有多种针对生物序列的各种重复片断的识别算法和软件[3,4]。文献[5]对DNA序列具有的高度相似性进行

到稿日期:2013-05-08 返修日期:2013-09-08 本文受国家自然科学基金(61263044,61163045),新疆维吾尔自治区高校科研计划重点项目(XJEDU2011I40),新疆维吾尔自治区自然科学基金(2012211A056),新疆师范大学计算机应用技术重点学科招标课题(12XSXZ0602)资助。
木妮娜·玉素甫(1963—),女,博士,副教授,主要研究方向为数据挖掘、模式识别、自然语言处理,E-mail:munina.yu@gmail.com;古丽娜·玉素甫(1974—),女,硕士,副教授,主要研究方向为信息化教育理论与实践研究、计算机教育应用;张海军(1973—),男,博士,副教授,主要研究方向为自然语言处理。

了描述，并指出 DNA 序列的这种相似性特点是当前众多 DNA 序列压缩算法的重要基础。

一般来说，文档中频繁模式的查找问题包括查找已知的特定模式、对于内容未知但长度给定的模式以及所有不定长的模式。对于比较复杂的最后一类问题，文献中有许多种解决方案，其中基于后缀索引的方法目前是最常用、效率较高的频繁模式发现方法。后缀索引的有效实现方式及在重复模式识别领域中的应用主要有后缀树及其变种，包括后缀数组、增强后缀数组和压缩后缀数组。

重复模式查询中的一个瓶颈问题是查询所用的索引结构所需的空间过大。一般来说，对于长度为 n 的序列，后缀树所需的空间为 $20n$ 到 $40n$，增强的后缀数组所需的空间为 $8n$，后缀数组所需空间只有 $4n$，但是后缀数组经常需要配合使用最长公共前缀数组，因此会增加空间复杂度。

针对以上问题，本文将设计全新的基于 QSA 数组计算序列中所有带有约束条件的 NE 重复模式的算法。与同类算法相比，主要贡献有：(1)与已有的基于后缀索引(后缀树、后缀数组等)的算法相比，由于只使用了 QSA 数组，此算法的空间效率得到了提高，运行时只使用略大于 $5n$ 的内存空间，对于占用空间较大的应用较为有效；(2)算法设计中充分考虑了 NE 重复模式的特征，用以建立特征和重复模式检测结果之间的统计联系；(3)算法中的约束条件包括最小周期 $p_{\min}$ 和最大间距 $g_{\max}$，可用于筛选符合条件的 NE 重复模式；(4)可按照递增序输出所有 NE 重复模式的出现位置。

2 查找 NE 重复模式算法的相关研究

文献[6]首次提出利用后缀数组查找序列中所有 NE 重复模式的算法，且时间复杂度达到了 $O(n)$。但是此算法需要计算两次后缀数组和最长公共前缀数组的值，将在预处理阶段耗费不必要的时间，且空间复杂度为 $17n$。Narisawa 等[7]提出了一种新模式计算方法，称为 substring equivalence classes，此算法稍加修改，即可输出所有 NE 重复模式，但是此算法用了一个辅助数组，增加了空间复杂度，大约为 $12n$。文献[8]设计了几种算法识别序列中所有 NE 重复模式，其中有 2 种算法达到了线性时间，空间复杂度原为 $9n$。如果用 LCP 值覆盖 SA 值，则可将空间复杂度降低到 $5n$，但是 LCP 数组的计算仍然需要 $6n$。如果需要输出 NE 重复模式的出现位置，则仍需调入 SA 的值，还原到 $9n$。这些算法都是不带(或带很少)约束条件的算法。

3 基于 QSA 数组计算所有 NE 重复模式的算法

3.1 NE 重复模式的形式化描述

给定一个长为 n 的序列 $x[1\cdots n]$，我们通过以下属性来定义重复模式的完全性、不可扩展性及约束条件。

定义 1 设 u 为 x 的一个子串，则重复模式(Repeats)是一个多元组(tuple)：$M_{x,u}=(p;i_1,i_2,\cdots,i_e)$，且有：

(1) $p\geqslant 1, e\geqslant 2$；

(2) $1\leqslant i_1\leqslant i_2\leqslant\cdots\leqslant i_e\leqslant n$；

(3) $u=x[i_1\cdots i_1+p-1]=x[i_2\cdots i_2+p-1]=\cdots=x[i_e\cdots i_e+p-1]$；

则称 p 为 $M_{x,u}$ 的周期；e 为 $M_{x,u}$ 的指数；u 为 $M_{x,u}$ 的生成元，也称为 x 的重复子串，$i_1,i_2,\cdots,i_e$ 为重复子串起始位置。

由定义 1 知，x 的重复模式 $M_{x,u}$ 不仅给出了 x 的重复子串 u，还给出了 u 在 x 中的(至少两个)出现位置。如果 $M_{x,u}$ 给出了 u 在 x 中的所有出现，则称 $M_{x,u}$ 是完全的。

定义 2 设 $x=uvu$，其中 v 不为空，则称重复子串 u 是分隔的；设 $x=uu$，则称重复子串 u 是串联的；设 $x=ababa$，则其中重复子串 $u=aba$ 是交叉的。

定义 3 设 $M_{x,u}=(p;i_1,i_2,\cdots,i_e)$，则 $M_{x,u}$ 可左扩展(Left Extendible，简称 LE)当且仅当 $(p+1;i_1-1,i_2-1,\cdots,i_e-1)$ 也是一个 Repeat。

从定义 3 可引申出不可左扩展(NLE)定义：至少存在一对 s,t $(1\leqslant s<t\leqslant e)$，使得 x 的第 (i_s-1) 个字符和第 (i_t-1) 个字符不相同，此时称 $M_{x,u}$ 为不可左扩展的。

定义 4 设 $M_{x,u}=(p;i_1,i_2,\cdots,i_e)$，则 $M_{x,u}$ 可右扩展(Right Extendible，简称 RE)当且仅当 $(p+1;i_1,i_2,\cdots,i_e)$ 也是一个 Repeat。

从定义 4 可引申出不可右扩展(NRE)定义：至少存在一对 s,t $(1\leqslant s<t\leqslant e)$，使得 x 的第 (i_s+p) 个字符和第 (i_t+p) 个字符不相同，此时称 $M_{x,u}$ 为不可右扩展的。

定义 5 如果一个 Repeat 既是 NLE 又是 NRE，则称其 NE[6,8]，或最大化[9]。

为了快速抽取序列中的绝大多数重复模式，NE 重复模式(最大化重复模式)的引入是必要也是足够的。它能够简洁地捕获序列中的所有有意义的重复结构，同时可避免产生大量无谓的输出：可扩展或非最大化的重复模式无需报告，因为它们必定包含于某些最大化的重复模式中[6,8,9]。

在图 1 中，序列 x 中共有 11 个重复模式，包括串联、分隔及交叉重复模式，即 $M_{x,a}=(1;1,3,4,6,7)$，$M_{x,b}=(1;2,5,8)$，$M_{x,aa}=(2;3,6)$，$M_{x,ab}=(2;1,4,7)$，$M_{x,ba}=(2;2,5)$，$M_{x,aab}=(3;3,6)$，$M_{x,aba}=(3;1,4)$，$M_{x,baa}=(3;2,5)$，$M_{x,abaa}=(4;1,4)$，$M_{x,baab}=(4;2,5)$，$M_{x,abaab}=(5;1,4)$。我们的算法 RPT 只输出其中 3 个 NE 重复模式，即为 $M_{x,a}$，$M_{x,ab}$，$M_{x,abaab}$，大大减少了输出次数，但是却隐含了所有重复模式。

	1	2	3	4	5	6	7	8
X=	a	b	a	a	b	a	a	b

图 1 序列 $x=abaabaab$

定义 6 根据定义 1，设 $g_1=i_2-i_1-p$，$g_2=i_3-i_2-p$，$\cdots$，$g_{e-1}=i_e-i_{e-1}-p$；则 $g_1,g_2,\cdots,g_{e-1}$ 称为重复子串之间的距离。

定义 7 约束条件包括最小周期 $p_{\min}$ 和最大间隔 $g_{\max}$。如果算法中设定了 $p_{\min}$ 和 $g_{\max}$ 的值，则算法只输出周期大于等于 $p_{\min}$ 并且 $g_1,g_2,\cdots,g_{e-1}$ 小于等于 $g_{\max}$ 的 NE 重复模式。

在图 2 中，$M_{x,AGC}=(p;i_1,i_2,\cdots,i_e)=(3;2,9,13,22)$；$M_{x,AGC}$ 不可左扩展，因为 $(4;1,8,12,21)$ 分别代表 AAGC，AAGC，TAGC，GAGC 的子串，已不是一个 Repeat；同理，$M_{x,AGC}$ 不可右扩展，所以 $M_{x,AGC}$ 是周期为 3、指数为 4 的 NE 重复模式。其中相邻重复子串之间的距离是 4，1，6。如果算法中设定 $g_{\max}=5$，则只输出 $M_{x,AGC}=(3;2,9,13)$。如果算法中设定 $p_{\min}=4$，则只输出周期大于等于 4 的 NE 重复模式，如 $M_{x,AGCC}=(4;13,22)$。

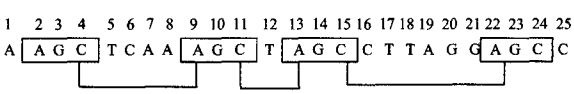

图 2 序列 x=AAGCTCAAAGCTAGCCTTAGGAGCCA 中 u=AGC

3.2 QSA 数组

为了实现算法 RPT，我们引入一种新的数据结构，称为准后缀数组(Quasi Suffix Array (QSA))，此数据结构最初见于文献[10]，后被应用于多种数据挖掘与处理算法。

定义 8 对于递增长度 $p=1,2,\cdots$(重复模式的周期)和递减位置 $i=n,n-1,\cdots,1$，算法 RPT 计算 $QSA[i]=j$，此处 j 为不大于 i 的最大整数，使得下式成立：

$$x[j\cdots j+p-1]=x[i\cdots i+p-1]$$

当满足条件的 j 不存在时，$QSA[i]=0$。

计算图 1 中序列 x 的 QSA 数组，可得：当 $p=1$ 时，$QSA[1\cdots n]=[0,0,1,3,2,4,6,5]$；当 $p=2$ 时，$QSA[1\cdots n-1]=[0,0,0,1,2,3,4]$；以此类推。从定义中可看出，QSA 数组计算得到的是距离最近的重复子串的起始位置(从右向左计算)。

3.3 查找 NE 重复模式的算法 RPT 设计

算法设计中充分考虑了 NE 重复模式的特征，用以建立特征和重复模式检测结果之间的统计联系，通过将待验证子串的特征分类为 LE、NLE 且 RE、NLE 且 NRE 3 种，进而得到 NE 重复模式。

算法 RPT 完全避免了计算后缀树或后缀数组等的值，而是直接利用数组 QSA 的值计算所有的 NE 重复模式，约束条件包括最小周期 $p_{\min}$ 和最大间隔 $g_{\max}$。

算法 RPT 首先需要初始化 QSA 数组和 n'。快速排序：初始化 QSA ($p=1$)，使得 $QSA[i]=j$，当且仅当 j 为首个位置使得 $x[i]=x[j]$；如果条件不成立，则 $QSA[i]=0$；初始化 n' 为 QSA 中的位置为零的个数。表 1 给出了初始化算法。

表 1 初始化算法——初始化 QSA 数组和 n'

```
count[1…α]=0^α;n'=0
for i=1 to n do
    QSA[i]=count[x[i]]
    if QSA[i]=0 then n'=n'+1
    count[x[i]]=i
```

对某些最大的 $i=i_1$，结果就是一个“链”(chain)：对 i_1，$i_2,\cdots,i_k$，长度 $k\geqslant 1$，对每个 $h\in 1\cdots k-1$，$QSA[i_h]=i_h+1$，并且 $QSA[i_k]=0$。因此当 $k\geqslant 2$ 时，每个链 (p,i) 定义了一个序列 x 中长度为 p 的完全重复模式。

表 2 给出了主算法 RPT，用于输出所有 NE 重复模式(周期$\geqslant p_{\min}$，间距$\leqslant g_{\max}$)。从表 2 中可看出，算法 RPT 对每个新的链依据其扩展性进行了分类：

1)如果当前链是 LE，则位置 $i=i_1,i_2,\cdots,i_k$ 将不被认为是属于任何一个 NLE 重复模式的一部分，因而我们对这些 i，设置 $QSA[i]=0$。

2)如果当前链是 NLE 并且 RE，则对满足条件的 $p'>p$，输出链 (p',i)，并且链中的这些位置的 PROC 值将会随之改变。

3)如果当前链是 NLE 并且 NRE(证明是 NE 重复模式)，将会输出这个链。

主算法 RPT 中各个函数作用如下：

当前链为 LE 时，使用函数 setzero，对当前链 (p,i) 中的每个使得 $QSA[h]!=0$ 的位置 h，设置 $QSA[h]=0$，并且 $n'=n'+1$。

在函数 outchain 中，如果 $p\geqslant p_{\min}$，输出包含在链 (p,i) 中的间距小于 $g_{\max}$ 的元素。对于链中位置 $pos>j$，设置 $PROC[pos]=$TRUE；并设置 $QSA[j]=-QSA[j]$。

表 2 主算法 RPT——输出所有 NE 重复模式(周期$\geqslant p_{\min}$，间距$\leqslant g_{\max}$)

```
p=1;max=n
while n' < n do
    PROC[1…max]=FALSE^max
    for i=max downto 2 do
        if not PROC[i] then PROC[i]=TRUE;q=QSA[i]
            if q > 0 then (LE,j,j')=checkchain(p,i,q)
                if LE then (QSA,n')=setzero(p,i,q,n')
                else if j'=0 then PROC=oldchain(p,i,q)
                else
                    outchain(p,i,q,j,p_min,g_max)
                    if j=i then q=-q
            if q < 0 then (QSA,n')=splitchain(p,i,q)
p=p+1;max=max-1
```

函数 oldchain 和 splitchain 如表 3 所列。函数 oldchain 的作用是当识别出当前链是 NLE 并且 RE，则设置 $PROC[t]=$TRUE，并跳过当前的 p。

表 3 函数 oldchain 和 splitchain

```
function oldchain(p,i,q)
    t=q
    while t>0 do PROC[t]=TRUE;t=QSA[t]
function splitchain(p,i,q)
    i'=i+p
    if i'>n then QSA[i]=0;n'=n'+1
    else
        t=-q;λ=x[i']
        while t > 0 and λ!=x[t+p] do t=QSA[t]
        QSA[i]=t
        if t=0 then n'=n'+1
    t=-q;QSA[t]=-QSA[t]
```

在以 p 为周期的链的内部，以负 QSA 的值确定、识别出最左边的位置(如果不存在，则设为零)，即为周期是 $(p+1)$ 的链的元素。这是函数 splitchain 的作用。

可以利用函数 checkchain 中已知信息确定在链中 j,j' 位置，也就是 RE 的终止位置。函数 checkchain 如表 4 所列。

表 4 函数 checkchain

```
function checkchain(p,i,q)
    h=i;h'=q;LE=TRUE;RE=TRUE
    while h'> 0 and (LE or RE) do
        if LE and x[h-1] !=x[h'-1] then LE=FALSE
        if RE then
            if x[h+p] !=x[h'+p] then RE=FALSE;t=h;t'=h'
        h=h';h'=QSA[h']
    if h'=0 and RE then return (LE,h,h')
else return (LE,t,t')
```

对图 1 中序列应用算法 RPT 后，定位和抽取得到 $M_{x,a}=(1;3,4,6,7)$，$M_{x,ab}=(2;1,4,7)$，$M_{x,abaab}=(5;1,4)$ 3 个 NE 重复模式，如果设置最小周期 $p_{\min}=2$，则只输出后两个。对图 2 序列 x 中完全不可重复模式 $M_{x,AGC}=(3;2,9,13,22)$，如果算法设置 $g_{\max}=4$，则只输出 $M_{x,AGC}=(3;2,9,13)$，为不完全的。

3.4 算法分析与比较

现在分析算法 RPT 的时间复杂度和空间复杂度。RPT

在 for 内循环中调用 3 个函数：checkchain、oldchain 和 splitchain，它们的时间复杂度都为 $O(n)$。由于 RPT 中 for 循环的循环控制变量为 $\max=n,n-1,n-2,\cdots,n-k$，而 k 的值依赖于 n' 的设定值，因此 for 循环的执行时间在最坏情况下可为 $O(n^2)$。则算法 RPT 在最坏情况下的时间复杂度是 $O(n^3)$。对一些特定的序列，比如含有较少重复模式的序列，算法 RPT 的运行速度还是会很快，因为会很快跳出内层循环；而且也依赖于 $p_{\min}$ 和 $g_{\max}$ 的输入值，$p_{\min}$ 的设定值越大，$g_{\max}$ 的设定值越小，执行速度会越快。

算法最多使用略大于 $5n$ 字节的内存来存储序列 x、数组 QSA 及位矢量 PROC，因而空间效率比文献[6,7]中同类算法有很大提高，比文献[8]中的算法也有所提高。

与同类算法比较，文献[6,7]算法都没有设定约束条件，文献[8]中算法仅设定了最小周期 $p_{\min}$ 约束条件；在这些算法中有几种达到了线性时间，但是实际运行最快的算法是最差时间复杂度为 $O(n^2)$ 的算法[8]；由于这几种算法都使用了后缀数组，其输出结果为 $(p;i,j)$ 形式，即周期为 p 且后缀数组下标为从 i 到 j（区间）的 NE 重复模式，还原到后缀数组则为 $(p;SA[i],SA[i+1],\cdots,S[j])$，而不是按照递增序输出所有 NE 重复模式的出现位置。如图 1 所示，x 序列中 NE 重复模式 $M_{x,a}$ 的输出形式是 $(1;1,5)$[8]，也即 $(1;6,3,7,4,1)$，而非 $(1;1,3,4,6,7)$。如要求满足按照递增序输出的约束条件，则必须对输出结果进行排序，使用最差时间复杂度较低的排序法进行排序，这也会使得整体算法的最差时间复杂度达到 $O(n^3\log n)$；如再要求算法满足最大间距 $g_{\max}$ 的约束条件，则会进一步提高其时间复杂度。

4 实验设计及结果分析

本节对算法 RPT 进行测试与分析。需要指出的是，本文中实现的算法没有使用后缀索引结构，而文献[6-8]的同类算法中算法整体时间的绝大部分被后缀数组和最长公共前缀数组的计算所占，由于所用的计算这两种数组的算法不同，算法整体时间将会有很大差别，而且这些算法没有本文中算法的约束条件，因而时间复杂度与实际运行时间都有很大区别。因此本文所设计算法 RPT 没有与这些算法进行实验比较。

4.1 分类属性数据样本集

结合作者目前所研究课题，测试语料主要采用了生物序列和维吾尔语 Web 文本，作为对比，也测试了两种随机语料与英语 Web 语料。表 5 列出了算法测试所用实验数据表的详细信息。生物序列主要来源于 Genbank 等大型 DNA 数据库，维吾尔语 Web 文本来自于网络，我们利用搜索引擎下载维吾尔语网站的页面，然后对图像、声音等非结构的数据进行清除，仅保留维吾尔语文本并转换成 Unicode 标准编码。因此实验所用分类属性数据样本集主要为以下 4 类：

- 字符集为 4 个碱基{A,T,C,G}的 DNA 序列；
- 字符集为 20 个氨基酸残基符号的蛋白质序列；
- 字符集较大的序列（维吾尔语与英语 Web 文本）；
- 含有较少重复模式的序列（字符集较小或很大的随机字符串）。

表 5 算法测试所用实验数据表

编号	文件类型	文件名称	文件长度 (Byte)	文件描述
1	DNA 序列	ecoli	4638690	大肠杆菌基因组
		chr21	34553758	人类第 22 号染色体
		chr19	63811651	人类第 19 号染色体
2	Genbank 蛋白质序列数据库	pro-a	16777216	样本
3	Web 语料库	uyghur-1	4136688	维吾尔语 Web 文本
		eng-1	4047399	英语 Web 文本
4	随机	rand2	8388608	字符集=2
		rand21	8388608	字符集=21

4.2 实验结果分析

算法用 C++实现，编译器使用了 GNU 的 g++，并使用了 O3 优化选项。实验中的时间取运行 10 次的最小值，并且没有包括读取输入文件所花费的时间。运行时间以C++标准库中的 clock 函数记录。

对测试最终结果，我们给出了每个字节所使用的时间，因为此数字对相同类型的语料来说一般是稳定的，整体变化较小。测试过程中 $p_{\min}$ 取值从 1 逐渐递增到 100，每次递增值为 5，同时 $g_{\max}$ 保持不变；$g_{\max}$ 取值从 1000000 到 0，每次递减 10，同时 $p_{\min}$ 保持不变。表 6 中给出了算法 RPT 对所用 4 类实验数据的计算结果，其中 $p_{\min}$ 取值为 1，$g_{\max}$ 取值为 1000000，是接近最差情况下的运行时间。

表 6 算法 RPT 对 4 大类别实验数据的计算结果

编号	文件名称	语料规模 (Byte)	计算时间 (ms)	处理速度 (ms/Byte)
1	ecoli	4638690	123922603.35	26.715
	chr21	34553758	636549329.88	18.422
	chr19	63811651	2165639811.64	33.938
2	pro-a	16777216	1098555326.46	65.479
3	uyghur-1	4036688	34988107.10	7.958
	eng-1	4047399	33346520.36	8.239
4	rand2	8388608	14159970.30	1.688
	rand21	8388608	19730006.02	2.352

正如 3.4 节中分析与预测，对于递增的 $p_{\min}$ 和递减的 $g_{\max}$，运行时间逐渐减少，当 $p_{\min}=100$ 时，运行时间大约是 $p_{\min}=1$ 时的一半。因而在实际运行时，当 $p_{\min}$ 的值不是太小，$g_{\max}$ 不是太大时，算法的速度相对是很快的。算法很适合于参数 $p_{\min}$ 和 $g_{\max}$ 的值很重要的情形，比如对某一固定 p 值，当 $g_{\max}$ 的取值很小，而得到的 NE 重复模式数量很多时，说明此种 NE 重复模式在此语料中是密集型的；当不需要很短的重复模式时，可将 $p_{\min}$ 的值根据要求进行调整；当不需要距离大于某个阈值的 NE 重复模式时，可对 $g_{\max}$ 的值进行设定等。

从表 6 中也可看出，算法 RPT 对 4 大类别实验数据的处理速度是不同的，处理速度最快的是含有较少重复模式的随机序列，其次是维吾尔语与英语 Web 语料，这些语料都是由很多长度不同且不同领域文本组成的，这些文本中字词的总数大致为几百万个，且维吾尔语文本未作词干抽取预处理。而对生物序列的处理，是以 DNA 序列为优，RNA 序列稍差。

结束语 本文通过对 NE 重复模式的特征研究，提出了一种全新的算法 RPT，即利用 QSA 数组计算得到距离最近的重复子串的起始位置的特性，将待验证子串的特征分类为 3 种，通过调用不同函数进行处理，进而得到满足限制条件的

（下转第 262 页）

据集的粒化树是对数据集的层层粒化表示。如果一个粒中的数据看作近似相同，则粒化集的每一层是对数据集的近似表示，粒化树使之进一步结构化。这样的数据处理方式与粒计算的数据处理模式相一致，是粒计算方法的体现，更为数据处理的程序化提供了顶层支撑。据此构建算法并程序化后，可利用计算机寻找关联元素和关联链，并由计算机提供人才供需的相关信息，从而达到数据自动处理的目的。

结束语 基于数据集的粒化树是对数据集的逐步划分与细化，这促成了关联元素与关联链之间的相互依存，以及粒度变化与数据关联程度相互联系的数学描述，使得关联链之间的关联元素得以确定，以及通过关联元素寻求关联链的想法成为了可能。因此，本文的讨论提供了针对关联数据处理的一种方法，这种方法一致于粒计算的数据处理模式，是粒计算用于数据处理的尝试。同时，由于树是计算机可以接受处理的数据结构，并能通过算法予以模拟，因而上述的讨论是以粒计算的数据处理模式为依托，探究如何描述关联数据，以及研究如何产生算法的数学基础的过程。

在实际应用中，问题涉及的数据集可能超出两个，形成多个数据集的局面。是否可以利用本文的方法同时处理多个数据集，按照不同的分类规则分别生成粒化树，描述它们之间潜在的联系，建立多维度的数学模型，是下一步将要面对的问题。从粒计算的角度考虑，大数据的分解是今后数据处理的重要方面，是众多学者关注的方向，也可考虑利用粒化树的方法予以研究。目前可以肯定的是，本文的讨论为今后的工作做了铺垫，基于粒化树的数据处理方法与多个数据集相结合的探究可作为今后努力的目标。

参 考 文 献

[1] Lin Tong-yan. Granular computing on binary relations II: rough set representations and belief functions[C]// Rough Sets and Knowledge Discovery. 1998:122-140

[2] Lin Tong-yan. Granular computing II: infrastructures for AI-Engineering[C]// Proceedings of 2006 IEEE International Conference on Granular Computing. Atlanta, USA, 2006:2-7

[3] Yao Yi-yu. A partition model of granular computing[J]. LNCS Transactions on Rough Sets, 2004(1):232-253

[4] 苗夺谦，李道国. 粗糙集理论、算法与应用[M]. 北京：清华大学出版社，2008

[5] Zadeh L A. Fuzzy sets and information granulation [M]// Advances in Fuzzy Set Theory and Applications. Amsterdam: North-Holland Publishing, 1979

[6] Zhang Ling, Zhang Bo. Theory of fuzzy quotient space(methods of fuzzy granular computing)[J]. Journal of Software, 2003, 14(4):770-776

[7] Qiu Guo-fang, Chen Jin. Concept knowledge system and concept information granular lattice[J]. Chinese Journal of Engineering Mathematics, 2005, 22(6):963-969

[8] Lin Tong-yan. Neighborhood systems and relational database [C]// Proceedings of CSC, 88. New-York, 1988

[9] 胡清华，于达仁，谢宗霞. 基于邻域粒化和粗糙逼近的数值属性约简[J]. 软件学报，2008，19(3)：640-649

[10] 张燕平，张铃，夏莹. 商空间理论与粗糙集的比较[J]. 微机发展，2004，14(10)：21-24

[11] 闫林. 数理逻辑基础与粒计算[M]. 北京：科学出版社，2007：155-160

(上接第 252 页)

所有 NE 重复模式。由于算法中只使用了 QSA 数组，空间效率得到了提高。在分类属性数据样本集上进行的实验表明，算法对生物序列尤其是 DNA 序列以及维吾尔语 Web 文本中 NE 重复模式的识别都很有效。

如前所述，本文算法 RPT 只输出 NE 重复模式，但如果实际应用需要，则只需一个线性时间算法就可输出所有类型的重复模式。约束条件最小周期 p_{min} 和最大间距 g_{max} 的引入，使得使用算法时，可根据应用环境的要求和实际的需要，对某一具体应用所涉及的数据提出不同的约束性条件。

由于使用相似的处理步骤，本文算法可方便地扩展到基于重复模式的聚类算法研究、舆情热点发现、生物序列中公共模体发现、代码克隆检测、数据压缩等方面的研究中。

参 考 文 献

[1] Benson G. Tandem repeats finder: a program to analyze DNA sequences[J]. Nucleic Acids Research, 1999, 27(2):573-580

[2] Lander E S, Linton L M, Birren B, et al. Initial Sequencing and Analysis of the Human Genome[J]. Nature, 2001, 409(6822):860-921

[3] Price A L, Jones N C, Pevzner P A. De novo identification of repeat families in large genomes[J]. Bioinformatics, 2005, 21(Suppl.):351-358

[4] 霍红卫，王小武. DNA 序列中基于适应性后缀树的重复体识别算法[J]. 计算机学报，2010，33(4)：747-754

[5] 纪震，周家锐，姜来，等. DNA 序列数据压缩技术综述[J]. 电子学报，2010，38(5)：1113-1121

[6] Franek F, Smyth W F, Tang Yu-dong. Computing all repeats using suffix arrays[J]. Automata, Languages and Combinatorics, 2003, 8(4):579-591

[7] Narisawa K, Inenaga S, Bannai H, et al. Efficient computation of substring equivalence classes with suffix arrays[C]// Proc. 18th Annual Symp. Combinatorial Pattern Matching. 2007:340-351

[8] Puglisi S J, Smyth W F, Yusufu M. Fast optimal algorithms for computing all the repeats in a string[J]. Mathematics in Computer Science, 2010, 3(4):373-389

[9] 胡吉祥，许洪波，刘悦，等. 重复串特征提取算法及其在文本聚类中的应用[J]. 计算机工程，2007，33(2)：65-67

[10] Franek F, Holub J, Smyth W F, et al. Computing quasi suffix arrays[J]. J. Automata, Languages & Combinatorics, 2003, 8(4):593-606