有效挖掘闭合组合序列模式

闫雷鸣[1] 孙志挥[1] 张柏礼[1] 杨 明[2] 姚 蓓[3]
(东南大学计算机科学与工程学院 南京 210096)[1] (南京师范大学计算机科学与技术学院 南京 210097)[2]
(南京擎天科技有限公司 南京 210002)[3]

摘 要 序列模式的挖掘是近年来的研究热点之一,目前很多研究都集中在闭合频繁项集与闭合序列模式的挖掘,较少涉及更加复杂、有重要应用价值的组合序列模式。针对任意长度和任意组合次数的频繁组合序列模式,提出了一种挖掘全部闭合的组合序列的算法CloCSP。为克服指数量级的候选序列进行闭合检验的困难,提出了既能生成频繁组合序列,又能有效剪枝,并同时完成闭合检验的混合扩展策略,该策略无需维护候选集。实验表明,CloCSP算法能够有效挖掘出隐藏在序列数据中,尤其是稠密数据集内的闭合组合序列模式,有助于揭示更加复杂的序列模式。

关键词 频繁序列,闭合组合序列,组合模式,数据挖掘
中图法分类号 TP311 **文献标识码** A

Mining Closed Composite Sequential Patterns Efficiently

YAN Lei-ming[1] SUN Zhi-hui[1] ZHANG Bai-li[1] YANG Ming[2] YAO Pei[3]
(School of Computer Science & Engineering, Southeast University, Nanjing 210096, China)[1]
(School of Computer Science & Technology, Nanjing Normal University, Nanjing 210097, China)[2]
(Nanjing Sinosoft Technology Company, Nanjing 210002, China)[3]

Abstract Sequential pattern mining has been an essential mining task and an active research area in recent years. However, existing sequential pattern mining algorithms are designed for closed itemsets or simple closed sequential patterns, and can hardly extract composite sequential patterns, an important class of patterns consisting of several short segments separated by gaps. An efficient algorithm for mining frequent closed composite sequences with any number of segments of different lengths, CloCSP, was proposed. It adopts a novel composite strategy called Mixed Composite, which not only can produce all of closed composite sequential patterns, but also can efficiently prune the composite space and simultaneously check the sequential patterns closure, accordingly reduces the cost in both runtime and space usage. Experiments on both synthetic and real data have demonstrated that CloCSP can significantly discover all of closed composite sequential patterns.

Keywords Frequent sequences, Closed composite sequences, Composite Motif, Data mining

1 引言

序列模式挖掘的研究内容非常广泛,涉及DNA序列模式[1]、闭合频繁项集[2,3]和闭合频繁序列模式[4,5]等,在分析客户购买行为模式、Web访问模式、数据流分析等方面都有重要的应用。目前已有的序列模式挖掘方法主要针对简单的频繁项集或者序列进行挖掘,较少考虑序列在时间、子模式间彼此关联等方面的因素。然而,许多实际应用中有价值的模式往往比较复杂,例如DNA序列中,序列的顺序和距离都是至关重要的,除了形如ACCT,ATTG等的传统频繁项集,一些结构化的频繁项集具有更高的研究价值,文献[1]在真实DNA数据中成功发现了一些已知的、典型的基因序列模式,例如CGG(11)CCG,TGTGA(6)TCACA,其中(11)和(6)分别表示两个频繁模式之间的距离,或者相隔的任意字符的个数。这类模式是由多个频繁序列模式组合而成,一般称为组合模式(Composite Pattern或Composite Motif)。在数据流频繁模式和Web访问行为等方面,也可以发现类似的组合模式。该类模式考虑了时间或位置因素,蕴涵的信息比传统模式更丰富。

考虑如图1(a)所示的序列,若频繁阈值为3,3个序列自左至右对齐,用闭合的频繁序列模式挖掘算法[4,5]可以很容易地发现频繁序列,例如AC,BB,DC和ACBBDC。但是,得不到包含位置因素的组合序列模式。如果进行恰当的移位,如图1(b)所示,则不但可以发现连续的频繁模式AC,BB和DC,还可得到组合模式AC(2)BB,BB(2)DC,AC(6)DC,AC(2)BB(2)DC。

到稿日期:2009-12-31 返修日期:2010-03-01 本文受国家自然科学基金(60873176,60803061),江苏省自然科学基金(Bk2008293)资助。
闫雷鸣(1973—),男,博士,CCF会员,主要研究领域为数据挖掘、生物信息学等,E-mail:yan_leiming@163.com;孙志挥(1941—),男,教授,博士生导师,主要研究领域为数据库理论与应用、知识发现等;张柏礼(1970—),男,博士,讲师,主要研究领域为数据仓库、数据挖掘和传感器网络等;杨 明(1964—),男,教授,博士生导师,主要研究领域为数据挖掘、机器学习、粗糙集理论等。

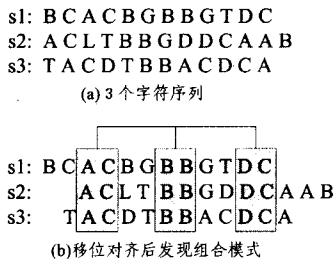

图 1 组合模式示例

2 相关工作

最近几年,有关 DNA Motif 的挖掘研究非常活跃[6-8],早期的组合模式挖掘工作源自对 DNA 序列组合模式的研究。文献[1]首次定义了结构化 DNA 序列模式(Structured Motif)挖掘问题,之后的研究大多称其为 Composite Motif 挖掘问题。挖掘时需要预先指定模式中所含子序列的长度、子序列的个数和子序列间的距离范围,例如按照模板 NNN(5,6)NNN 挖掘,其中 NNN 表示长度为 3 的频繁子序列,(5,6)表示两个序列间距离为 5 或 6。如果挖掘前对存在的模式缺乏了解,则只能猜测一个结构模板。目前已有的几个算法,例如 RISO 算法[6]和文献[7],基本都采取了这种挖掘方式。

该方法存在两个明显的局限:第一,虽然指定模板可以有效降低挖掘组合模式的难度,但也极大地降低了算法发现未知模式的能力,因为在现实数据中存在很多结构未知的模式。事实上,在研究真核生物的基因转录时发现存在更加复杂的组合 motif 结构[9,10],在某些情况下,甚至可以有 12 个基部[11]。第二,算法一次运行只能发现一种结构的组合模式,如果需要挖掘全部组合模式,则需要穷举各种结构模板并多次运行,挖掘效率低。

针对上述不足,有必要设计不需限定组合模式的结构,即无需指定模板、并能挖掘全部组合序列模式的算法,这对于缺乏数据中模式结构先验知识的挖掘任务无疑是有益的。

此外,组合序列模式的另一个意义在于它适用于对稠密序列数据的挖掘。传统的频繁序列挖掘算法通常是针对稀疏序列数据集设计的,并不适于挖掘稠密序列数据。例如,根据文献[5]的实验,算法 CloSpan 和 BIDE 在挖掘稠密数据时,当最小支持度小于 88%时,获得的频繁闭合序列数量高达 10^6 量级,同时算法运行时间也达 10000s 量级。然而,挖掘组合序列模式,则可在较短时间得到一些值得探究的结果。

算法面临的主要困难在于,潜在的组合序列模式数量是指数级的。虽然挖掘闭合的组合序列模式将使结果更紧凑,但挖掘难度和复杂度更高。传统的闭项集挖掘算法通常都需要维护候选项集,以便检验新生成的候选项是否已被包含,或者已有的候选项是否被新生成的候选项包含,这些都将导致巨大的时间与空间开销。

针对上述挑战,本文针对一般序列数据,定义了包含任意数目子序列、不限定子序列长度和彼此间距离的闭合组合序列模式,提出了一个可挖掘全部闭合组合序列模式的算法 CloCSP,算法采用的混合搜索策略可以直接输出闭合序列,无需维护候选闭合模式集合。

3 闭合组合序列挖掘算法

设 Σ 为字母表,M 是序列数据库 D 上的频繁序列模式,并且是连续的无间隔的序列模式,$M=a_i a_j \cdots a_k$,$a_i \in \Sigma$,称 M 为简单序列模式。

定义 1(组合序列模式,composite sequential pattern) $S=M_i(ni)M_{i+1}(nj)\cdots(nq)M_{i+k}$,其中 $M_i, M_{i+1}, \cdots, M_{i+k}$ 表示 k 个长度$\geqslant len$ 的简单序列模式,$k\geqslant 2$,(ni)表示两个简单序列模式之间的距离,对应 ni 个任意字母,$ni\geqslant 1$,len 为简单序列的最短长度。

为了简化形式,在需要时也可表示为 $S= M_i\_M_{i+1}\_\cdots\_M_{i+k}$,其中下划线_表示任意距离。

对于 $S:k$ 的形式,正整数 k 表示该模式的支持度。

与文献[1]不同,本文定义的组合序列模式不限制简单序列模式的个数、长度和间隔距离。

定义 2(闭合组合序列模式) 对组合序列模式 S,不存在组合序列模式 S',使 $S\sqsubset S'$,并且 $\sup(S')=\sup(S)$,则称 S 是闭合组合序列模式。

从图 1 可以看到,为了获取组合序列模式,应该按照一定的方案将各个序列对齐。显然,不同的对齐方案可能得到不同的组合序列模式。序列对齐是一个时间代价高昂的过程,如果穷举对齐方案,其时间复杂度为各序列长度的乘积。

3.1 基本算法

算法主要分为两个阶段。

第一阶段,获取序列数据的各种对齐方案,并挖掘频繁序列模式,即获取简单序列模式集合 Q。

第二阶段,逐个提取 Q 中的简单序列模式 Mi,作为种子模式,搜索组合空间,生成全部包含 Mi 的组合序列模式,若闭合,则输出闭合组合序列模式。

在第一阶段,本文利用泛化后缀树[12]的优良性能,设计有效算法。有关后缀树的研究和应用非常广泛,主要用于字串搜索、提取最长公共字串等。其突出的特性是,可以将一个长度为 n 的序列在 $O(n)$时间内插入后缀树。在将序列插入到后缀树的过程中,例如插入序列 abcdef,算法将分别把后缀 abcdef,bcdef,cdef,def,ef,f 插入。如果在插入时,将序列本身的插入位置保存下来,就可以在各节点中获得各种序列对齐方案,遍历树中的节点可以获得频繁的序列模式。

性质 1 泛化后缀树中每个非根节点都对应一个字符序列。对于支持度大于给定频繁阈值的内部节点 n_i,如果存在后缀链 $n_j\rightarrow n_i$,并且支持度 $\sup(n_j)<\sup(n_i)$,则 n_i 对应一个闭合频繁序列。

证明:当存在后缀链 $n_j\rightarrow n_i$ 时,n_i 是 n_j 的后缀,节点 n_j 对应的序列比 n_i 的长 1,当 $\sup(n_j)<\sup(n_i)$时,n_j 不能包含 n_i,所以 n_i 对应一个闭合频繁序列。

同理可证性质 2。

性质 2 设泛化后缀树中内部节点 n_j 的孩子节点中有叶子节点 $leaf$,存在后缀链 $n_j\rightarrow n_i$,n_i 中有与 $leaf$ 对应相同序列的叶子节点 $leaf'$,如果 $leaf$ 的支持度大于给定频繁阈值,并且 $\sup(leaf')>\sup(leaf)$,则 $leaf'$ 对应一个闭合频繁序列。

根据上述性质,可以设计出从泛化后缀树中挖掘简单序列的算法。

算法 1 ExtractMotif(M_i,Δ,$flag$)

输入:字符序列数据,频繁阈值

输出:简单序列模式队列

```
1  if(flag=1)字符匹配方式为:相对位置相同并且符号相同
2  else 字符匹配方式为:仅需符号相同
3  将节点 Mi 所含各序列数据插入泛化后缀树
4  while(还有节点未访问) { //遍历后缀树
5      提取节点 node
6      if(sup(node)>Δ) then {//如果是频繁序列
7          if(node 是内部节点) then {
8              if(不存在后缀链 nodei→node)
                   Queue=input(node)
9              else if(存在后缀链 nodei→node,
                       且 sup(nodei)< sup(node))
10                 Queue=input(node)}
11         else if(node 是叶子节点) then {
12             if(存在 nodei→father(node),且存在 leaf∈chil-
    dren(nodei)的字符序列与 node 相同)
13                 if(sup(leaf) < sup(node))
                       Queue=input(node)
14                 else Queue=input(node)}
15 } // while 循环结束
```

上述算法第 6 行和第 11 行提取闭合的简单序列，第 12 行提取当前无法判断是否闭合的简单序列。该阶段不需要一定提取闭合的简单序列，闭合性将在第二阶段完成。

每个简单序列都存储了当前的序列对齐方案。另外，为了获取序列的各种对齐方案，根据后缀树性质，对于同一序列内的重复子序列，将重复插入树中同一个节点，因为每次插入时对应的位置不同，所以视作独立的序列，每次插入都记一个支持度。

针对类似 $A\_B\_A\_A$ 这种模式，当缺乏位置信息时，无法区分第一个 A 与第二个、第三个 A 的不同，即存在类似 $A\rightarrow\cdots\rightarrow A$ 的环。在组合过程中，必须判断并恰当地终止因环所导致的循环。解决方法是，在第二阶段，提取简单序列模式内的对齐方案，将该模式对应的序列集合重新插入一个新的后缀树，在字符匹配时，相对位置相同且符号相同才视为匹配，这样因为位置确定，可将类似 $A\_B\_A\_A$ 的有环模式转化为形如 $A_1\_B_1\_A_2\_A_3$ 的无环模式，简化了组合问题。

3.2 混合扩展

本文提出了一种混合扩展策略来生成组合序列模式。该策略结合剪枝技术，可以直接得到闭合模式，而不必维护候选集合用于闭合性检验，从而节省时间与空间开销。

所谓混合扩展，指在由简单序列扩展出组合模式过程中，混合使用双向扩展与前向扩展的组合策略，以求快速并无重复地生成全部组合序列模式。

定义 3(前向扩展) 设组合序列模式 $S=M_i(ni)M_{i+1}\cdots M_{i+k}$，扩展为 $S'=M_i(ni)M_{i+1}\cdots M_{i+k}(nk)M_{i+k+1}$，称为前向扩展，又记为 $S'=S(nk)M_{i+k+1}$。

定义 4(后向扩展) 设组合序列模式 $S=M_i(ni)M_{i+1}\cdots M_{i+k}$，扩展为 $S'=M_{i-1}(nk)M_i(n_i)M_{i+1}\cdots M_{i+k}$，称为后向扩展，又记为 $S'=M_{i-1}(nk)S$。

由简单序列 M_i 扩展生成的全部组合序列模式集合记为 $CSP(M_i)$。

由于作为种子的简单序列 M_i 记录了对齐方案，因此该序列模式所包含的各序列间的相对位置确定，对于可以和 M_i 进行组合的候选简单序列模式，可按照相对位置进行排序。

排序方法：以种子 M_i 为基准位置，以各候选简单序列 M_j 中首字符相对基准的位置作为该简单序列的起始位置，位置编号较小的排在前面，即如果 $pos(M_j)\leqslant pos(M_{j+1})\leqslant\cdots\leqslant pos(M_{j+k})$，其中 $pos(M_j)$ 表示简单序列的位置编号，则排序为 $M_j\leqslant M_{j+1}\leqslant\cdots\leqslant M_{j+k}$。

显然，如果候选模式集合中各简单序列是有序的，则相对位置靠前的将先被扩展，那么在其后的简单序列，只需要做前向扩展，故可得如下性质。

性质 3 如果候选的简单序列是有序的 $M_j\leqslant M_{j+1}\leqslant\cdots\leqslant M_{j+k}$，则对候选的简单序列只需做前向扩展，不必做后向扩展。

算法 ExtractMotif() 提取的种子是一个二元组 $Node(Mi)=(M_i,S_i)$，其中 M_i 是对应的简单序列模式，$S_i=\{s_k\mid\forall s_k,M_i\subseteq s_k\wedge s_k\in D\}$ 是支持 M_i 的字符序列集合。$Node(M_i)$ 与 $Node(M_j)$ 的组合操作定义为：$Node(M_i)\circ Node(M_j)=\{M_i\_M_j \text{ or } M_j\_M_i, S_i\cap S_j\}$，简记为 $M_i\circ M_j$。

设 $Q_{seed}=\{M_i\mid M_i$ 是候选的简单序列模式$\}$，集合 P_{seed} 为扩展完毕的种子集合。

基本混合扩展过程：

1)提取一个种子 $M_i\in Q_{seed}$；

2)如果 Q_{seed} 无序，双向扩展，$Q^0_{seed}=M_i\circ Q_{seed}=\{M_i\circ M_k\mid M_k\in Q_{seed}\}$；

3)否则，前向扩展，

$Q^0_{seed}=M_i\circ Q_{seed}=\{M_i\circ M_k\mid M_k\in Q_{seed}\wedge M_i\leqslant M_k\}$

4)如果 Q^0_{seed} 不为空，对 Q^0_{seed} 中简单序列模式按相对位置先后排序，并插入组合模式树，如图 2 所示；

5)对 Q^0_{seed} 中的每个简单序列模式，$Q_{seed}=Q^0_{seed}$，执行第 1)－5)步骤。

6)将扩展完毕的 M_i 加入集合 P_{seed}，$P_{seed}=P_{seed}+M_i$。

如图 2 所示，树中每个节点代表一个简单序列模式，每个分枝可以拼出一个组合模式。上述过程会重复生成一些组合模式，下面通过剪枝避免重复生成相同的组合序列模式，并提高模式组合的效率。

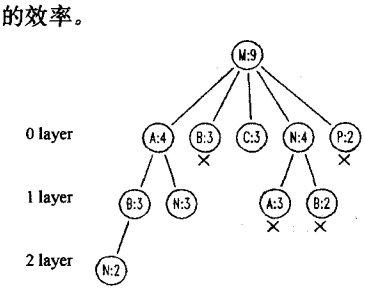

图 2 组合模式树示例

3.3 剪枝与闭合性检验

为便于描述，以 $Father(B)$ 表示节点 B 的父亲节点，以 $Prev(B)$ 表示上一层中同样是 B 的节点；如果组合模式 S 是由 A 组合生成的，记为 $A\rightarrow S$，即 $S\in CSP(A)$。

定理 1(剪枝定理) 设简单序列 $A\in P_{seed}$，候选组合序列模式 S，如果 $S'=S(nk)A$ 或者 $S'=A(nk)S$，则 $A\rightarrow S'$，可以安全地将 S' 删除。

证明：因为 A 已经被扩展过，即存在 $CSP(A)$，使得 $S'\in CSP(A)$，所以 $A\rightarrow S'$，同理 $CSP(S')\subseteq CSP(A)$，没有必要继续扩展 S'，故可删除 S'。

推论 1 设 $Prev(A)\in P_{seed}$，$M\rightarrow S$，如果 $S'=S(nk)A$ 或者 $S'=A(nk)S$，并且支持度 $\sup(S')=\sup(M)$，则 $CSP(S')=$

$CSP(M)\subseteq CSP(A)$，可以安全地将 M 从候选集合内删除。

证明：因为 $M\rightarrow S$，且 $S\rightarrow S'$，所以 $M\rightarrow S'$，即 $S'\in CSP(M)$，又 $\sup(S')=\sup(M)$，所以 $CSP(S')=CSP(M)$，根据剪枝定理，$CSP(S')\subseteq CSP(A)$，则 $CSP(S')=CSP(M)\subseteq CSP(A)$，所以无需再对 M 和 S' 做进一步的组合了，可将 M 从候选集合内删除。

推论 2 设 $Prev(A)\in P_{seed}$，并且 $\sup(A)=\sup(Father(A))$，则 $CSP(Father(A))\subseteq CSP(A)$，可以安全删除 $Father(A)$。

证明：设 $S=Father(A)$对应的模式，S 扩展为 $S'=S(nk)A$ 或者 $S'=A(nk)S$，因为 $\sup(A)=\sup(Father(A))$，所以 $\sup(S')=\sup(S)$，则 $CSP(S)=CSP(Father(A))=CSP(S')$，根据推论 1 可得，$CSP(Father(A))=CSP(S')\subseteq CSP(A)$，$CSP(S)$已经被 $CSP(A)$包含，故可以安全删除 S。

推论 2 是推论 1 的特例。

推论 3 设 $Prev(A)\notin P_{seed}$，并且 $\sup(A)=\sup(Father(A))$，则无需扩展 A，将 A 从候选简单序列集合中移出，与 $Father(A)$合并。

证明：设 $S=Father(A)$对应的模式，S 扩展为 $S'=S(nk)A$ 或者 $S'=A(nk)S$，因为 $\sup(A)=\sup(Father(A))$，所以 $\sup(S)=\sup(S')$，则 $CSP(S)=CSP(Father(A))=CSP(S')$，因此不需对 A 做组合，只需组合扩展 S'(将 A 与 $Father(A)$合并)。

同理可证推论 4。

推论 4 设 $Prev(A)\notin P_{seed}$，$S'=S(nk)A$ 或者 $S'=A(nk)S$，如果 $\sup(A)=\sup(Prev(A))$，则 $CSP(Prev(A))\subseteq CSP(S')$，可以安全删除 $Prev(A)$。

定理 2 闭合定理(充要条件)

当且仅当(1)$\sup(S)>\mathrm{Max}\{\sup(S_k)\mid S_k\in S\circ Q_{seed}\}$，即 S 的支持度大于 S 经一步扩展所得的每个模式的支持度，并且(2)S 中不存在某个简单序列 $M_i\in P_{seed}$，则 S 为非重复生成的闭合组合序列模式。

充分性证明：若 S 非闭合且是重复生成的，则只有两种情况，(1)S 被已生成的模式包含，(2)已生成的模式被 S 包含。只要证明上述两种情况不存在即可。

反证：混合扩展可以生成由种子序列 M 扩展出的全部组合序列，设当前组合模式为 S，如果 $\exists$ 闭合组合序列 X，使得 $S\subseteq X$，则 $\sup(S)=\sup(X)$，说明 X 可以进一步扩展，即 $\exists S_k$，$\sup(S_k)=\sup(X)$，$S_k\in X\circ Q_{seed}$，这与条件(1)矛盾。

对已有的模式 X，必 $\exists M_i$，使得 $M_i\rightarrow X$ 且 $M_i\in P_{seed}$，如果 $X\subseteq S$，则有 $X\rightarrow S$，根据剪枝定理必有，$M_i\rightarrow S$，$M_i\in P_{seed}$，S 应该在扩展过程中被删除，这与条件(2)相矛盾。

必要性证明：若 S 为闭合组合序列模式，根据闭合定义，不存在 S_k，$S\rightarrow S_k$，使得 $\sup(S)\leqslant\sup(S_k)$，即条件(1)成立；若 S 为非重复生成的，则重复的模式被提前删除，这由剪枝定理保证，即条件(2)成立。

满足闭合定理的组合序列模式是闭合的非重复生成的，因此可以直接输出结果，不需要维护候选闭合模式集合以验证闭合性，从而节省了这部分的空间与时间开销。

3.4 CloCSP 算法

下面给出 CloCSP 算法的具体描述。

算法 2 剪枝算法 PatternsPruning(*Queue*)

输入：未被扩展过的候选简单序列队列
输出：剪枝后的候选简单序列队列

```
1 while(候选简单序列队列 Queue 不为空){
2 M=dequeue(Queue)
3 if(PrevM)∈P_seed)
        //如果 M 的上一级节点已经被扩展过
4 then if (sup(M)= sup(father))
        //如果 M 的支持度等于父亲节点的支持度
5       then P_seed=+father,return //推论 2
6       else P_seed=+M  //剪枝定理
7 else //上一级 M 尚未被扩展过
8       if(sup(M)= sup(father)) then
9       将 father=M∪father,P_seed=+M //推论 3
10      else if(sup(M)= sup(Prev(M)))
11          then P_seed=+ Prev(M) //推论 4
12              M'= Prev(M)
13              while(sup(M')= sup(Prev(M'))){
14                  P_seed=+ Prev(M') //推论 4
15                  M'= Prev(M') }
16 } // while 循环结束
```

算法 3 混合扩展算法 Mixed-Composite(*Queue*, *S*)

输入：候选简单序列队列，组合模式
输出：闭合组合序列模式

```
1  PatternsPruning(Queue)
2  if (sup(S)>Max{sup(M_i)| M_i∈S∘Queue } AND S 中简单序列
   数目>1)
3      输出闭合组合序列模式 S
4  M_i= dequeue(Queue)
5  Queue1=ExtractMotif(M_i,Δ,1)//插入泛化后缀树，位置相同并
   且符号相同的字符才视为匹配
6  对 Queue1 中的简单序列排序
7  while(Queue1≠∅) Mixed-Composite(Queue1,S) //递归调用
8  P_seed=+M_i//标记 M_i 为已扩展
```

算法 4 闭合组合序列模式算法 CloCSP(*SeqData*, Δ)

输入：字符序列数据集，频繁阈值
输出：全部闭合组合序列模式

```
1  Queue=ExtractMotif(SeqData,0,Δ)//将原始数据插入泛化后缀
   树，匹配方式为仅需符号相同
2  while(种子队列 Queue≠∅){
3      M= dequeue(Queue)
4      Queue1= ExtractMotif(M,Δ,1)//将 M 中包含的各序列数
   据插入泛化后缀树中，匹配方式为位置相同并且符号相同的字符
   才视为匹配
5      Mixed-Composite(Queue1,S=∅) //混合扩展
6      P_seed=+M } //标记种子 M 为已扩展
```

3.5 算法复杂度分析

设 n 为数据库中序列的个数，序列平均长度为 m，组合模式中最大组合个数为 k，简单序列最小长度为 len。

算法第一阶段，主要是后缀树的创建和遍历，根据后缀树可在序列长度的线性时间内创建的性质易知，树的创建与遍历的时间复杂度为 $O(mn)$。设算法第一阶段生成 N 个简单序列模式。

算法第二阶段，对 N 个简单序列模式递归地进行混合扩展，最大递归层数即组合模式中最大组合个数为 $k\leqslant m/(len+1)$。扩展时主要耗费为对每个种子模式重新创建后缀树，复杂度为 $O(mn)$，k 次递归，对候选简单序列模式队列进行 k 次

排序，复杂度为 $O(kN\log N)$，与每个候选简单序列做组合。因为组合时候选模式队列是有序的，所以每次递归时，候选模式都至少比上一层少 1，根据组合模式树，可以计算组合次数：

第 0 层有 N 个简单序列，对第一个种子，

第一层组合次数$\leqslant N-1$，第二层组合次数$\leqslant N-2$，…

第 k 层组合次数$\leqslant N-k$，

第一个种子组合次数$\leqslant\sum_{i=1}^{i=k}(N-k)=kN-\frac{k(k+1)}{2}$，

第二个种子有 $N-1$ 个候选简单序列，所以

第二个种子组合次数$\leqslant\sum_{i=1}^{i=k}(N-1-k)=k(N-1)-\frac{k(k+1)}{2}$，

…

第 $N-1$ 个种子组合次数$\leqslant k(N-N+2)-\frac{k(k+1)}{2}$，

所以 $N-1$ 个种子共需最大组合次数为

$$\sum_{r=0}^{r=N-2}\sum_{i=1}^{i=k}(r-k)=k[N+(N-1)+\cdots+2]-\frac{(N-1)k(k+1)}{2}$$

$$=[k(N-1)(N+2)-k(N-1)(k+1)]/2$$

$$=\frac{1}{2}(kN^2-k^2N+k^2-1)$$

所以，算法最坏复杂度为

$$(N-1)O(mn)+(N-1)O(kN\log N)+O(kN^2-k^2N)=O(mnN+kN^2\log N+kN^2-k^2N)$$

4 实验与分析

本文使用 C++语言和标准模板库 STL 实现算法；程序运行平台为 Windows XP 操作系统，PC 机配置为 Intel Core2 CPU 1.8GHz，1GB 内存。

实验数据分别采用 IBM 合成数据生成器 GenData 生成的稠密数据集和真实 DNA 序列数据。因为目前没有其他同类型的挖掘闭合组合序列的算法，所以选取挖掘非闭合组合序列模式的算法 RISO-STA[6]（该算法是同类算法的典型代表）与 CloCSP 做横向比较。

4.1 稠密合成数据

该合成数据为稠密序列数据，共生成 35 个序列，序列平均长度为 100，有 4 个不同的符号。

用该数据考察在不同支持度阈值下，算法的运行时间和内存耗费占用情况。为避免生成大量琐碎的无意义的模式，设置参数 *len*（组合模式中简单序列的最短长度）分别取 3 和 4，调整支持度阈值，考察算法时间与空间耗费，如图 3 所示。从运行时间来看，即便支持度小于 30%，算法时间仍可在 100s 之内。

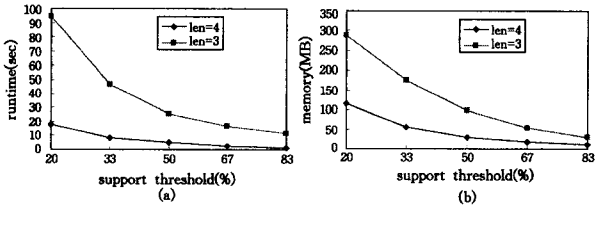

图 3 支持度阈值与时间、空间耗费变化关系

与 RISO-STA 算法比较：

RISO-STA 算法是根据结构模板进行挖掘，一次运行只能提取一种结构的组合序列，而本文的 CloCSP 一次运行能提取全部组合序列模式，故不能直接对比一次运行的时间和内存耗费。

将 RISO-STA 运行两次，分别提取与模板 NNN(2,10)NNN 和 NNN(2,10)NNN(2,10)NNN 匹配的组合模式，将两次的结果合并，与 CloCSP 进行对比，如表 1 所列，当最小支持度低于 33%时，RISO-STA 两次运行的结果数量已经超过 CloCSP 的结果数量，这主要是因为，RISO-STA 的结果是非闭合的，得到的大量结果中有一些是被其他模式包含的，是冗余的，而 CloCSP 得到的是闭合模式。

表 1 发现的组合模式数量比较

最小支持度	闭合模式数量（个）		
	RISO-STA NNN_NNN	RISO-STA NNN_NNN_NNN	CloCSP
20%	3087	7660	9860
33%	1869	806	1152
50%	807	49	193
67%	217	3	52
83%	48	0	47

从运行时间来看，虽然表 2 中的数据显示，RISO-STA 单次运行时间远小于 CloCSP，但是考虑到 CloCSP 挖掘的是全部组合模式，如果以 RISO-STA 挖掘全部组合模式，则需枚举各种可能的组合结构，其运行时间的总和无疑是高昂的。

表 2 运行时间比较

最小支持度	运行时间（秒）		
	RISO-STA NNN_NNN	RISO-STA NNN_NNN_NNN	CloCSP
20%	0.78	3.06	94.422
33%	0.672	1.343	46.688
50%	0.484	0.578	25.6
67%	0.25	0.25	16.844

从两个算法的特点来看，RISO-STA 适合挖掘某种已知结构的组合模式，或者单独挖掘某个特定模式；而 CloCSP 适合挖掘结构未知的模式，或者提取全部的闭合组合序列模式。

4.2 真实数据

采用文献[1]提供的枯草杆菌 B. subtilis 数据，提取 300 个基因序列进行实验，DNA 序列长度从 90 到 300 不等。

本文算法挖掘得到包括文献[1,6]所报告的 TTGACA(18)TATAAT 等大量组合模式，主要为包含 2 个和 3 个简单序列的组合模式，个别模式包含了 4 个简单序列，如表 3 所列。部分组合模式的生物学意义还有待进一步的检验。

表 3 挖掘出的部分组合序列模式

2 段组合序列模式：支持度%	多段组合序列模式：支持度%
TAT(73)AAA:17	TGT (2)GTG (4)TGT:4.7
TTGACA(18)TATAAT:12.7	CCG (7)GGG (10)TTT:4
CAG(16)CCC:3.7	CCC (7)GGGG (37)CGG:4
CCCA(7)GGG:3.7	GGG (10)TTT (24)CGG:3.7
CCA (7)GGGC:3.7	CCCG (7)GGG (10)TTT:3.7
GGG (35)CCGG:3.7	CCC (8)GGG (10)TTT:4.7
CCA (8)GGC:4	CCC (8)GGG (10)TTT (24)CGG:3.7

结束语 本文提出了挖掘序列数据库中复杂序列模式——闭合组合序列模式的问题，并给出了一种有效解决该问题的挖掘算法 CloCSP。为了达成上述任务，提出了混合扩

（下转第 205 页）

工作的初步结果。为了进一步提高这种模型和方法的可用性，还有一些难点需要解决，比如解决多表关联查询问题、多表嵌套查询问题、根据上下文查询问题和带有表达式语句的复杂查询等问题。

参 考 文 献

[1] Androutsopoulos I, Ritchie G, Thanisch P. Natural language interfaces to databases-an introduction [J]. Journal of Language Engineering, 1995, 1(1): 29-81

[2] Johnson T. Natural Language Computing: The Commercial Applications[M]. London: Ovum Ltd., 1985

[3] Androutsopoulos I. Interfacing a Natural Language Front-End to a Relational Database [D]. Department of Artificial Intelligence, University of Edinburgh, 1993

[4] Woods W A, Kaplan R M, Webber B N. The Lunar Sciences Natural Language Information System: Final Report[R]. BBN Report 2378. Cambridge, Massachusetts: Bolt Beranek and Newman Inc., 1972

[5] Waltz D L. An English Language Question Answering System for a Large Relational Database [J]. Communications of the ACM, 1978, 21(7): 526-539

[6] OWL Web Ontology Language Reference [EB/OL]. http://www.w3.org/TR/owl-ref

[7] Fellbaum C. WordNet: An Electronic Lexical Database [M]. Cambridge, MA: MIT Press, 1998

[8] van Eijck J, Kamp H. Representing Discourse in Context [D]. Elsevier, Amsterdam: Handbook of Logic and Language, 1997: 179-237

[9] Grinberg D, Lafferty J, Sleator D. A robust parsing algorithm for link grammars [R]. CMU-CS-95-125. Proc. Fourth Int. Workshop on Parsing Technologies. Prague, September 1995

[10] The Stanford Parser: A statistical parser[EB/OL]. http://nlp.stanford.edu/software/lex-parser.shtml

[11] Popescu A, Etzioni O, Kautz H. Towards a theory of natural language interfaces to databases [C]// Proceedings of the 8th International Conference on Intelligent User Interfaces. Miami, Florida, USA: ACM Press, January 2003: 327-327

[12] Copestake A J. Natural language interfaces to databases [J]. Knowledge Engineering Review, 1990, 5(4): 225-249

[13] Janus J M. The Semantics-based Natural Language Interface to Relational Databases [M]. Cooperative Interfaces to Information Systems, New York: Springer-Verlag, 1986: 143-187

[14] Kaplan S J. Designing a Portable Natural Language Database Query System [J]. ACM Trans. on Database Systems, 1984, 9(1): 1-19

[15] Trinkunas J, Vasilecas O. Building Ontologies from Relational Databases Using Reverse Engineering Methods [C]// ACM International Conference Proceeding Series. 2007: 285

(上接第 190 页)

展策略，在前、后两个方向上对序列模式进行组合，同时进行剪枝优化和闭合检验。实验结果表明，CloCSP 能够有效挖掘闭合组合序列模式。

参 考 文 献

[1] Marsan L, Sagot M-F. Algorithms for extracting structured Motifs using a suffix tree with an application to promoter and regulatory site consensus identification[J]. J. Computational Biology, 2000, 7(3/4): 345-362

[2] Wang Jianyong, Han Jiawei, Pei Jian. CLOSET+: searching for the best strategies for mining frequent closed itemsets [C]// Proceedings of KDD'2003. 2003: 236-245

[3] Zaki M J, Hsiao C-J. Efficient Algorithms for Mining Closed Itemsets and Their Lattice Structure [J]. IEEE Trans. Knowl. Data Eng., 2005: 462-478

[4] Yan X, Han J, Afshar R. CloSpan: Mining Closed Sequential Patterns in Large Databases [C]//Proc. SIAM Int'l Conf. Data Mining (SDM'03). May 2003: 166-177

[5] Wang Jianyong, Han Jiawei, Li Chun. Frequent Closed Sequence Mining without Candidate Maintenance [J]. IEEE Transaction on Knowledge and Data Engineering, 2007, 19(3): 1042-1056

[6] Carvalho A M, Freitas A T, Oliveira A L, et al. An Efficient Algorithm for the Identification of Structured Motifs in DNA Promoter Sequences [J]. IEEE/ACM Trans. Comput. Biology Bioinform, 2006, 3(2): 126-140

[7] Fassetti F, Greco G, Terracina G. Mining Loosely Structured Motifs from Biological Data [J]. IEEE Trans. Knowl. Data Eng., 2008, 20(11): 1472-1489

[8] Wijaya E, Rajaraman K, Yiu Siu-ming, et al. Detection of generic spaced motifs using submotif pattern mining [J]. Bioinformatics, 2007, 23(12): 1476-1485

[9] Werner T. Models for Prediction and Recognition of Eukaryotic Promoters[J]. Mammalian Genome, 1999, 10(2): 168-175

[10] Tu Z, Li S, Mao C. The Changing Tails of a Novel Short Interspersed Element in Aedes Aegypti: Genomic Evidence for Slippage Retrotransposition and the Relationship between 3' Tandem Repeats and the Poly(da) Tail[J]. Genetics, 2004, 168(4): 2037-2047

[11] Pavesi G, Mauri G, Pesole G. In Silico Representation and Discovery of Transcription Factor Binding Sites[J]. Briefings in Bioinformatics, 2004, 5: 217-236

[12] Bieganski P, Riedl J, Carlis J V, et al. Generalized Suffix Trees for Biological Sequence Data: Applications and Implementation [C]//Proceedings of HICSS (5)'1994. 1994: 35-44