DOI:10.13374/j.issn1001-053x.2006.05.039

基于广义后缀树的事件序列频繁情节挖掘算法

曲文龙[1,2) 杨炳儒[2) 张克君[2)

1) 石家庄经济学院,石家庄 050031 2) 北京科技大学信息工程学院,北京 100083

摘 要 为了有效地挖掘事件序列频繁情节,提出了一种广义后缀树结构发现和存储频繁情节.此结构利用广义后缀概念并且树中只包含频繁情节结点,用频繁情节发生列表逐层构建的方法提高了建树效率.该方法充分利用了事件序列的有序特点,可用于发现各类频繁情节.实验结果表明该算法性能优于Apriori-like频繁情节发现算法.

关键词 事件序列;频繁情节;数据挖掘;广义后缀树

分类号 TP311.13

事件序列是按时间或空间连续发生的事件集合,事件序列中的频繁情节挖掘广泛应用于金融市场、气象预报、网络报警、web使用模式、基因和蛋白值序列分析等众多领域,因此日益受到重视和关注.交易数据库中的频繁项集发现[1]和序列模式挖掘[2]是发现交易内项目之间的关联,而事件序列中频繁情节挖掘与此不同,频繁情节是发现事件序列中重复出现的事件组合,发现的是事件之间的关联,原有的算法不能直接使用.Mannila等提出发现频繁情节的WinEPI和MinEPI算法[3],前者采用事件持续窗口限制,后者采用情节最小发生限制.Casa-Garriga提出采用事件间隔限制发现无限长度频繁情节[4].Frank Hoppner从时间序列的定性行为(状态序列)发现频繁情节[5],Harms提出了事件序列闭情节发现算法[6].

以上算法都是将事件序列通过情节窗口转换为交易数据库,然后采用基于Apriori-like序列模式发现算法,需要生成大量的候选情节且需对事件序列反复扫描计数,因此其效率和可用性受到限制.本文针对事件序列的特点提出了一种广义后缀树结构,利用广义后缀建立频繁情节树型结构,利用上层结点的频繁情节发生列表对搜索空间进行划分,在相应的投影子序列中扫描下层情节,缩小了模式搜索空间.基于广义后缀树的频繁情节发现算法(Generalized Suffix Tree Algorithm,GSTA)不需生成大量候选并且对搜索空间进行了逐层划分,因此可以提高发掘的效率.实验表明该方法优于基于Apriori-like的频繁情节发现算法.

1 事件序列频繁情节相关概念

对事件序列进行知识发现是一个重要的领域.该问题不同于常规交易数据库和序列数据库,其数据不是交易的集合而是以时间顺序发生的事件流,因此其支持度需重新定义,还需对频繁情节加以约束,以防止组合爆炸和产生无意义模式.

分析事件序列的一个基本问题发现频繁情节,如"B在A后,C在B后"发生多次,则构成一个频繁情节,即为由一组事件组成的偏序集[3].频繁情节挖掘即发现在序列中的所有频繁出现的情节,图1给出一个事件序列.

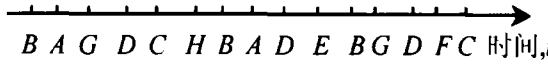

图1 事件序列

Fig.1 Event sequence

定义1 给定事件类型的集合E,事件是一个二元组(A,t),其中$A\in E$,t是事件发生的时间.事件类型可以包含多个属性,这里只考虑单一属性.

定义2 事件序列S是事件类型E上的一个三元组(s,T_s,T_e),其中:

$S=\langle(A_1,t_1),(A_2,t_2),\cdots,(A_n,t_n)\rangle$,

$A_i\in E_i(i=1,\cdots,n)$且$t_i\leqslant t_{i+1}(i=1,\cdots,n)$,$T_s$称为事件序列开始时间,$T_e$称为事件序列结束时间,并且有$T_s\leqslant t_i\leqslant T_e(i=1,\cdots,n)$.事件序列

收稿日期:2005-03-14 **修回日期**:2005-10-11

基金项目:北京市自然科学基金资助项目(No.4022008)

作者简介:曲文龙(1970—),男,博士研究生;杨炳儒(1943—),男,教授,博士生导师

S 是一组按时间(或空间)有序排列的事件.

定义 3　情节 α 定义为一个三元组 $(V, \leqslant, g)$，其中 V 情节中结点的集合，$\leqslant$ 是 V 上的偏序，$g: V \to E$ 是情节中的结点到对应事件类型的映射，即在情节中事件 $g(V)$ 的出现必须满足偏序 $\leqslant$ 定义的顺序. $|\alpha|$ 表示情节 α 的长度，$|\alpha|$ 的值为情节中的结点数 $|V|$. 情节可以用有向无环图(DAG)表示，见图 2.

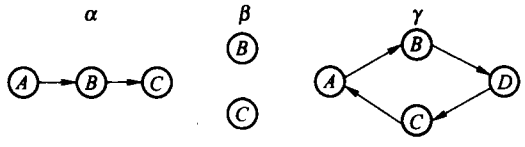

图 2　情节 α, β 和 γ

Fig.2　Episodes α, β and γ

如果关系 $\leqslant$ 为全序(即 $\forall x, y \in V$，必有 $x \leqslant y$ 或 $y \leqslant x$)，则称 α 为串行情节，例如 α. 如果关系 $\leqslant$ 为空(即 $\forall x, y \in V, x \neq y$ 不存在 $x \leqslant y$ 或 $y \leqslant x$)，则称为并行情节，例如 β. 同时包含串行并行的情节称混合情节例如 γ. 如果 $g: V \to E$ 为单射则称为 α 单射情节，即情节 α 中不存在重复发生的事件.

定义 4　给定情节 $\beta = (V', \leqslant', g')$ 和 $\alpha = (V, \leqslant, g)$，如果存在一个单射 $f: V' \to V$，使得对于 $\forall v \in V'$ 都有 $g'(v) = g(f(v))$ 成立，并且 $\forall v, w \in V', v \leqslant' w$ 都有 $f(v) \leqslant f(w)$ 成立，则称情节 α 为情节 β 的一个子情节，记为 $\beta \subseteq \alpha$. 子情节 DAG 是情节 DAG 的子图. 例如图 1 中 β 是 γ 的子情节.

定义 5　设 $\beta = (V', \leqslant', g')$ 是 $\alpha = (V, \leqslant, g)$ 的子情节且 $|\beta| < |\alpha|$，设单射为 $f: V' \to V$，若存在 $v, u \in V (v \neq u)$，使得存在 $f(v') = v (v' \in V')$ 但不存在 $f(u') = u (v' \in V')$，必有 $u \leqslant v$ 不成立，则称 β 为 α 的前缀子情节，称 α 为 β 扩充情节. 当 $|\alpha| = |\beta| + 1$ 时，则称 β 为 α 的相邻前缀子情节，称 α 为 β 相邻扩充情节.

定义 6　给定情节 $\alpha = (V, \leqslant, g)$ 和事件序列 $S = \langle (A_1, t_1), (A_2, t_2), \cdots, (A_n, t_n), T_s, T_e \rangle$，如果存在一个从情节结点到事件的单射 $h: V \to |1, \cdots, n|$，使得对 $\forall x \in V$ 都有 $g(x) = A_{h(x)}$ 成立，并且对 $\forall x, y \in V (x \neq y, x \leqslant y)$ 都有 $t_{h(x)} < t_{h(y)}$ 成立，则称情节在事件序列的一次发生.

为发现兴趣情节，必须对限制情节中事件的发生的时间接近度加以限制(但不一定全部相邻). 一种方法是采用窗口定义情节的最大持续期 maxduring，以情节在所有窗口中的出现频率作为其支持度，此方法的缺点是无法发现超过窗口宽度的频繁情节并且对不同长度的频繁情节使用统一的窗口. 另一方法限制相邻事件的最大时间间隔 maxgap，则包含 k 个事件的情节的窗口宽度为 $(k-1) \times$ maxgap，在事件序列 $S = (s, T_s, T_e)$ 中窗口总数为 $T_e - T_s + \text{win} - 1$. 此方法可发现任意长度的情节，其支持度按下式计算：

$$\text{fr}(\alpha, S, \text{win}) = \frac{|S_w \in W(S, \text{win}) | \alpha \in S_w|}{|W(S, \text{win})|} \tag{1}$$

其中，$W(S, \text{win})$ 为所有宽度为 $\text{win} = (k-1) \times \text{maxgap}$ 的窗口集合，S_w 表示情节发生的窗口子序列.

2　频繁情节广义后缀树

后缀树是一棵表示每个序列后缀的一种树状数据结构，主要应用于字符串匹配领域[7]. 事件序列频繁情节发现和字符串模式匹配具有内在相似性，所以本文对后缀树结构进行改造，称为频繁情节广义后缀树，并应用于频繁情节模式发现.

一个长为 n 的字符串 S 共有 $n + (n-1) + \cdots + 1 = (n^2 + n)/2 = O(n^2)$ 个子字符串 $S[i..j] (1 \leqslant i \leqslant j \leqslant n)$，由于一些子串可能相等，事实上可以利用后缀树在 $O(n)$ 空间列举其所有后缀子串.

定义 7　长为 n 的字符串 S 的后缀树是一颗有 n 个叶结点的根树. 每一内部结点至少有两个孩子结点，每个边用一非空子串标记，每一结点发出的边子串的起始字符均不同. 对于每一叶结点 i，从根结点到叶结点经历的每个边标记子串的连接，对应从第 i 个位置开始的字符串 S 的后缀.

松散后缀树：每边对应一个字符，每个结点至少包含一个孩子结点. 本文数据结构基于松散后缀树. 图 3 给出了字符串 $ABCAB\$$ 的后缀树和松散后缀树，$\$$ 表示串结束符.

利用广义后缀树发现频繁场的主要思想如下：

(1) 采用广义后缀树发现和存放频繁情节，实现共享前缀压缩. 构造事件序列的完全后缀树可以简单的用于频繁情节发现，但需要存放所有出现的情节，需占用大量内存空间. 为了发现至少出现 minsupp 次的频繁情节，必须采用相应策略裁剪树的规模，实际上只需构建至少有 minsupp

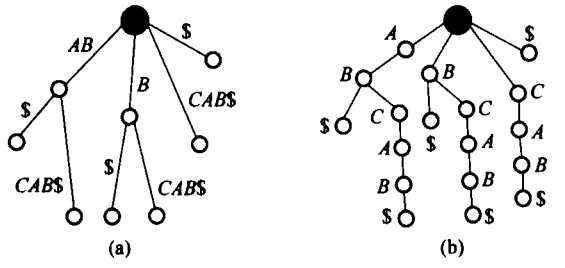

图 3 后缀树. (a) *ABCAB* $ 的后缀树; (b) *ABCAB* $ 的松散后缀树

Fig.3 Suffix tree: (a) suffix tree of string *ABCAB* $; (b) loose suffix tree of string *ABCAB* $

次发生的频繁场景结点.

(2) 后缀树中每个叶结点表示一个连续的后缀(见图 3),但无法发现如 B^*A(* 为对模式无贡献的任意类型事件)等的不连续的情节,因此需对后缀概念扩充,称这种有事件间隙的后缀为广义后缀,如 B^*A^* 也是序列的后缀. 称采用这种方法构造的频繁不完全松散后缀树为广义后缀树.

(3) 广义后缀使后缀树规模急剧增大,因此需施加约束以发现感兴趣情节. 常用的方法是采用情节最大持续窗口 maxduring,但这样无法保证发现所有的频繁情节且最大持续窗口难以事先估计. 因此本文采用一种相邻事件最大间隙限制 maxgap,可以发现任意长度的频繁情节,例如图 1 中当最大事件间隙为 3 时,第一个事件 A 的后续事件为 B, C, A,生长出的长为情节为 AB, AC 和 AA.

(4) 对长序列构造完全后缀树时间空间消耗较大,本文提出一种频繁情节位置逐层扩展方法,子结点利用父结点频繁情节位置列表生成,以逐层构造广义后缀树. 方法利用频繁情节位置列表实现对序列的逐层划分,在相应的投影子库上进行情节计数,缩小了搜索空间. 树的每个结点包括事件类型 E,支持计数 supp,频繁情节位置列表 positionlist,发生时间(t_{start}, t_{end}),利用这种广义后缀树结构,可以发现满足约束的所有频繁情节.

3 广义后缀树频繁情节挖掘算法

定义 8 广义后缀情节树是一颗根树,每个结点 N 对于一个情节 α,可以有多个子结点 childlist,情节 α 表示从根结点 root 到该结点 N 的结点标签 label 串连组成的情节. 每个结点包括如下域:事件标签 $N(\alpha)$.label$\in E$,发生次数 $N(\alpha)$.supp,发生位置列表 $N(\alpha)$.positionlist,孩子指针列表 $N(\alpha)$.childlist.

$N(\alpha)$.positionlist 每一项为(t_{start}, t_{end}),表示该情节一次发生的开始和结束时间,这样设置是为了防止情节重叠. 另外采用链表结构存放事件序列,每个结点 $L(A)$存放一个事件(A, t). 包括以下域:事件类型 L.event$\in E$,发生时间 L.time,后续事件指针 L.next.

性质 1 一个结点所表示的情节 β 是其子结点表示的情节 α 的前缀子情节.

证明:设结点 N 对应的情节为 $\beta=(V', \leqslant', g')$,其子结点 N.child 对应的情节为 $\alpha=(V, \leqslant, g)$, E 为事件类型集. 由于 $\alpha=(V, \leqslant, g)$是由 $\beta=(V', \leqslant', g')$增加一个后续事件 e 构成,因此 $V=V'\cup\{e\}$, $g=g'\cup(e\rightarrow E)$,取 $V'\rightarrow V$ 的单射为 $f(v)=v$,由定义 4 可得 β 是 α 的子情节. 另外任给 $u\in V(u\neq e)$,必有 $e\not\preceq u$ 成立,且 $|\alpha|=|\beta|+1$,所以 β 为 α 的相邻前缀子情节.

定理 1 子结点所表示的情节的支持度小于其父结点所表示的情节的支持度.

证明:由于父结点情节 $\alpha=(V, \leqslant, g)$是子结点情节 $\beta=(V', \leqslant', g')$的前缀子情节,再根据定义 6,每当 $\alpha=(V, \leqslant, g)$发生时 $\beta=(V', \leqslant', g')$也必发生,所以子结点场景的发生次数不大于父结点场景的发生次数. 由 $|\alpha|<|\beta|$,可得到 $|W(S, \text{win}_\alpha)|<|W(S, \text{win}_\beta)|$,再根据公式(1)的支持度定义,可得结论.

该定理表明频繁情节具有与交易数据库中频繁项集 apriori 特性相似的性质,因此可在树生长的每一层进行支持度剪枝,不会丢失频繁情节.

基于广义后缀树的频繁情节发现算法(GSTA)首先扫描序列得到所有频繁 1-情节的发生位置集,满足最小支持度 minsupp 的加入情节树中;对每一频繁 1-情节利用其发生位置集,得到所有满足约束的后续事件及极其发生位置集,将频繁后续事件作为该 1-情节的孩子结点插入情节树中;此过程反复进行直到没有后续频繁事件为止. 每当一结点的所有后续事件发生位置集产生后,释放该结点的发生位置集,以减少空间耗费.

图 4 说明树构造过程(maxgap 设为 1, maxduring$=\infty$, minsupp 为 2),由于图 1 例子中未给出事件发生的准确时间,因此以其位置序号视为时间. 其中加外框的结点表示频繁情节(加入树中),无外框的表示不频繁情节(不加入树

中). 根据图 1 的序列 *BAGDCHBADEBGDFC* 得到每个事件的位置索引，即 1-episode 的位置列表；1-episode 的位置列表中 *A*/*B*/*C*/*D* 为频繁，插入情节树中，*E*/*F*/*H* 不频繁，删除其位置列表；对每一频繁 1-episode，例如 *A* 根据其位置列表(2)(8)找到其后续事件进行扩充，得到 *AD* 位置列表(8,9)表示情节 *AD* 的开始时间 8 和结束时间 9，*AG* 位置列表(2,3)，由于 *AG* 和 *AD* 的出现次数都小于 minsupp，因此不加入情节树且不再进行扩充.

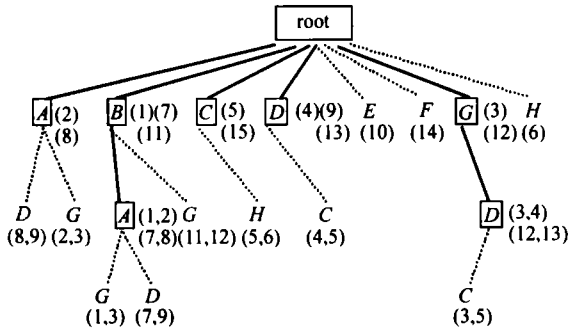

图 4　广义频繁情节后缀树的构建

Fig.4　Construction of the generalized frequent episodes suffix-tree

当 maxgap=2，对当前频繁情节结点后续两个事件进行扩充，如 2-episode 的位置列表为(1,2)(7,8)，其扩充情节为 *BAG*(1,3) *BAD*(1,4) *BAD*(7,9) *BAE*(7,10)，由于 *BAD* 发生次数≥minsupp，所以加入树中，并继续扩充为 *BADC*(2,5) *BADH*(2,6) *BADE*(7,10) *BADB*(7,11)，由于发生次数均 < minsupp，所以不加入树中. maxgap=2 时的 FrequentEpisodeTree 的构造结果见图 5.

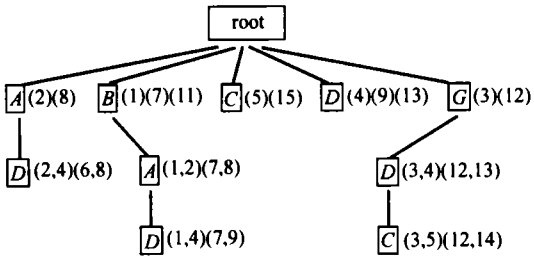

图 5　maxgap = 2 时的广义频繁情节后缀树

Fig.5　Generalized frequent episodes suffix-tree when maxgap = 2

定理 2　该算法可以发现所有满足约束的频繁情节.

证明：算法逐层剪除了不频繁情节所对应的结点，由定理 1，其子结点对应的情节一定是不频繁的，因此树裁剪不会剪除频繁情节；另一方面任何频繁情节对应结点的父结点为该情节的子情节，一定是频繁的，不会被剪除；因此该频繁情节生成算法可以发现所有满足约束的频繁情节.

基于后缀树的频繁情节发现算法(GSTA)如下：

算法 1：基于后缀树的频繁情节发现.

输入：事件序列 *S*，事件类型 *E*，最小支持度 min_supp，事件最大间隔 maxgap，情节最大持续期 maxduring.

输出：频繁情节模式树 *T*.

(1) sequence = createlink(null);

(2) read *S* to sequence in order;

(3) for each $e \in E$ position_list(e) = NULL;

(4) scan sequence to fill position_list(e)

(5) root = createtreenode(node, null, 0, null);

(6) for each $e \in E$

(7) if $|\text{position\_list}(e)|/|s| \geqslant \text{min\_supp}$ then

(8) root.child = createnode(node, e, |position_list(e)|, position_list(e));

(9) add e to frequent_event;

(10) append node to active_queue;

(11) else free(position_list(e));

(12) while active_queue not empty

(13) N = fetchfirstnode(active_queue);

(14) set next_event_set = NULL

(15) for each $e \in E$ position_list(e) = NULL;

(16) for each pos$\in N$.poslist

(17) if pos.next.*A* frequent_event and

(pos.next.time-pos.endtime≤maxgap and

pos.next.time-pos.starttime ≤ maxduring) and pos.next.*A* not existed in next_event_set then

(18) add pos.next to next_event_set;

(19) for each $p \in$ next_event_set

(20) add p to position_list(p.event);

(21) for each $e \in$ frequent_event

(22) if $|\text{position\_set}(e)|/(|s| + \text{level}(N) - 1) \geqslant \text{min\_supp}$ then

(23) N.child = createnode(node, e, |position_list(e)|, position_list(e));

(24) append node to active_queue;

(25) free(N.poslist);

(26) return root;

算法说明:(1)~(2)初始化存放序列的空链,将序列读入链表中. (3)~(4)初始化每个事件(1-情节)的位置列表,扫描序列以建立每个 1-情节的位置列表集. (5) 建立模式树根结点. (6)~(11)对所有出现次数大于最小支持度的事件:将该事件加入频繁 1-情节集、将该事件添加为根结点的孩子结点、将结点加入待处理队列中,对出现次数大于最小支持度的事件释放其位置列表集. (12)~(13)若待处理队列不空,反复取出队头结点 N 处理. (14)~(15)清空当前情节的后续事件集,清空每一频繁事件的发生位置集,以记录结点 N 的后续事件的发生位置. (16)~(18)若当前情节的后续事件为频繁事件、满足限制,若其位置未出现在后续事件集中则添加. (19)~(20)对后续事件集的每一事件,将其发生位置添加到所属事件类型的发生位置集中. (21)~(24)对当前情节 N 每一类型的后续事件,若其支持度大于最小支持度则创建 N 的孩子结点,加入该孩子结点到待处理队列中. (25)结点 N 处理完成后释放其发生位置列表,以减少内存占用.

通过遍历构建的频繁情节模式树可以生成事件预测规则,算法描述如下.

算法 2:事件预测规则生成算法.

输入:频繁情节模式树 T,最小支持数 minsupp,最小信任度 minconf,序列总长度 len.

输出:预测规则集 ruleset.

(1) FES=NULL;

(2) FES = travel_frequent_epi_tree (T, currentpatterr='');

(3) for each $p \in$ FES

(4) for i = 1 to length(p) - 1

(5) generate rule R.rule = $p[1..i] \Rightarrow p[i+1..\text{length}(p)]$

(6) find episodes q in FES such that q.pattern = $p[1..i]$

(7) R.supp = p.supp/(len/length(p));

(8) R.conf = p.supp/q.supp

(9) if R.supp $\geqslant$ min_supp and R.conf $\geqslant$ min_conf then

(10) add R to ruleset;

(11) return ruleset;

算法 2 说明:(1)初始化频繁情节集合 FES 为空. (2)调用 travel_frequent_epi_tree 得到所有频繁情节 p.episodes 及其支持计数 p.support,加入到频繁模式集 FES 中. (3~10)对频繁模式集的每一模式 p,以所有前缀为规则前件后缀为规则后件生成事件预测规则,其信任度为情节 p 的支持度与前件情节 q 的支持度的比值. 由于 q 为 p 的子情节,因此 q 的支持度不小于 p 的支持度,所以 q 必存在于频繁模式集 FES 中. travel_frequent_epi_tree 采用深度优先遍历频繁事件树 T,每一结点的情节由父结点的情节和本结点的标签 node.label 串联得到.

4 对并行和混合情节的处理方法

算法同样适用于并行和混合情节频繁情节发现,只需在频繁情节树逐层构造过程中允许进行并行情节扩充. 例如设父结点对应的频繁情节为 $\langle AB\rangle$,需对 $\langle AB\rangle$ 位置集中所有满足限制的后续事件的组合进行扩充,设为 $\langle AB\rangle$ 后续出现的满足限制的事件有 C, D, E,设允许的最大并行事件为 2,则其扩充为 $\langle ABC\rangle$, $\langle ABD\rangle$, $\langle ABE\rangle$, $\langle AB(CD)\rangle$, $\langle A(DE)\rangle$, $\langle A(CE)\rangle$, ()表示其中事件先后顺序无限制. 前三个为满足限制的串行扩充,后三个为满足限制的并行扩充,算法中可根据发掘任务添加限制,如情节最大持续期等.

5 实验结果

实验目的主要是将本文 GSTA 算法与基于候选生成的(Aprior-like)的频繁情节发现算法进行比较,以检验其优越性,算法采用 VC++6.0 实现,运行环境为 pIII800, 256 M 内存, win2000 平台. 实验数据采用 Geneva 大学 ExPASy (Expert Protein Analysis System) 服务器上的 PROSITE 数据库 (http://www.expasy.org/prosite/index.html) 中的表达蛋白序列,PROSITE 包含具有生物学意义的 DNA 和表达蛋白模式,主要用于识别蛋白质序列所属类别. 因为数据集中的模式是已知的,所以实验目的主要是对比验证该算法的有效性和优越性. 选择 DNA 失配修复蛋白(DNA mismatch repair proteins 1, PROSITE entry PS00058)的 79 个序列组合并视为一个事件序列,将其空间顺序视为时间. 在不同序列之间放置结束符 $ 以避免情节跨越不同序列. 序列的总长度为 53 130,包含 20 种不同事件类型,失配修复蛋白的特征模式为 GFRGEAL. 以下对本文提出的 GST-based 算法进行有效性和可伸缩性实验,并与 Apriori-like 算

法的运行时间进行对比实验.

实验 1：考察不同最小支持数 minsupp 下发现的不同长度频繁情节的数目. 表 1 为最小间隔 maxgap = 1 和 maxduring = ∞时的发掘结果，表 2 为最小间隔 maxgap = 2 和 maxduring = ∞时的发掘结果. 随着最小支持数 minsupp 的增大，得到的各长度(L2 ~ L7)的频繁情节数会减少，当 maxgap = 2 时产生的得到各长度的频繁情节数目均大于 maxgap = 1 的频繁情节数目. 实验中发现了特征频繁模式 GFRGEAL.

表 1　不同支持数下的频繁情节数(maxgap = 1)

Table 1　Number of requent episodes with different min_support (maxgap = 1)

Minsupp	L2	L3	L4	L5	L6	L7
10	382	1771	394	238	187	153
20	363	554	98	65	48	38
40	315	107	33	20	14	8
60	279	38	8	5	3	1
80	247	10	0	0	0	0

表 2　不同支持数下的频繁情节数(maxgap = 2)

Table 2　Number of frequent episodes with different min_support (maxgap = 2)

Minsupp	L2	L3	L4	L5	L6	L7
40	361	1731	390	321	376	485
80	315	455	64	41	50	59
140	266	68	5	2	3	6
200	217	6	0	0	0	0

实验 2：考察不同规模数据下运行时间. 蛋白质序列总长度为 53 130，取 maxgap = 1，maxduring = ∞. 由图 6 可见当 minsupp 取值较小时，随数据规模增大，频繁情节数目迅速增加，运行时间增长速度较快. 而当 minsupp 适当取值时，运行时间随数据规模的增长接近线性，表明该算法具有较好的可伸缩性.

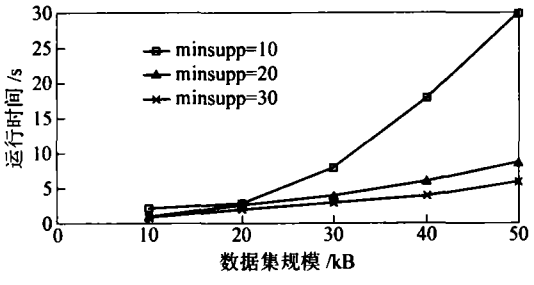

图 6　不同数据规模下算法的运行时间

Fig.6　Algorithm runtime of different data scales

实验 3：考察本文 GSTA 算法与 Apriori-like 算法在相同条件下的运行时间. 蛋白质序列总长度为 53 130，取 maxgap = 1，maxduring = ∞，两种频繁情节发现算法在最小支持数 minsupp 取不同值时的运行时间见图 7. 可见本文算法的运行时间小于 Apriori-like 的算法，分析其原因：一是算法不需生成大量候选情节，二是在情节计数时只需对逐层划分的投影子集进行搜索，不需对序列进行多次完全扫描.

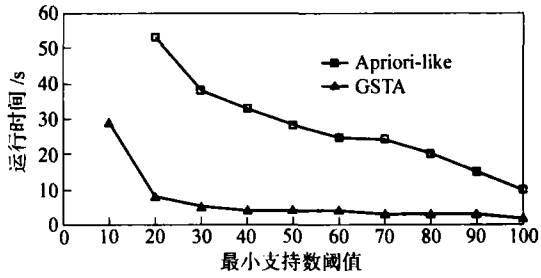

图 7　GSTA 与 Apriori-like 算法运行时间比较

Fig.7　Runtime comparison between GSTA algorithm and Apriori-like algorithm

6　结论

现有的基于 Aprior-like 的频繁情节挖掘算法，采用候选生成—剪除的试错策略，其缺点是需生成大量候选情节且需要反复进行事件序列扫描. 针对事件序列的有序特性，提出一种基于广义后缀树的事件序列频繁情节挖掘算法 GSTA. 该方法采用广义后缀树发现和存储频繁情节，树生成过程采用广度优先逐层剪枝策略以缩减树规模，采用频繁情节发生位置集实现搜索空间的划分，使得模式搜索只在投影子集中进行，有效缩减了搜索空间，提高了模式发现效率. 实验表明该算法优于基于 Aprior-like 的频繁情节发掘算法. 进一步的工作是发现事件序列频繁闭情节，以有效的缩减模式空间，进一步提高发掘算法的性能.

参 考 文 献

[1] Agrawal R, Srikant R. Fast algorithms for mining association rules // Proceedings of the 20th International Conference on Very Large Data Bases. Santiago, 1994: 487

[2] Srikant R, Agrawal R. Mining sequential patterns: generalizations and performance improvements // Proceedings of the International Conference on Extending Database Technology. Avignon, 1996: 3

[3] Mannila H, Toivonen H, Verkamo A I. Discovery of frequent episodes in event sequences. Finland: Department of Computer Science, University of Helsinki, 1997

[4] Casa-Garriga G. Discovering unbounded episodes in sequential data// Proceedings of the 7th European Conference on Principles and Practice of Knowledge Discovery in Databases. Cavtat-Dubvrovnik, 2003: 83

[5] Hoppner F. Discovery of temporal patterns-learning rules about the qualitative behaviour of time series// Proceedings of the 5th European Conference on Principles and Practice of Knowledge Discovery in Databases. Freiburg, 2001: 192 - 203

[6] Harms S, Deogun J, Saquer J, et al. Discovering representative episodal association rules from event sequences using frequent closed episode sets and event constraints// Proceedings of the 2001 IEEE International Conference on Data Mining. California, 2001: 603

[7] Kosaraju S R. Fast pattern matching in trees// Proceedings of the 30th IEEE Symposium on Foundations of Computer Science. New York, 1989: 178

Mining algorithm of frequent episodes in an event sequence based on generalized suffix-tree

QU Wenlong[1,2], *YANG Bingru*[2], *ZHANG Kejun*[2]

1) Shijiazhuang University of Economics, Shijiazhuang 050031, China

2) Information Engineering School, University of Science and Technology Beijing, Beijing 100083, China

ABSTRACT In order to mine frequent episodes from an event sequence efficiently, an algorithm based on generalized suffix-tree was proposed to discover and store frequent episodes, which uses the concept of generalized suffix and contains only frequent episodes' nodes. The occurrence list of frequent episodes was used layer-upon-layer to improve the efficiency of the tree. The algorithm make full use of the order character of an event sequence and may discover the variety of frequent episodes. Experimental results show that the proposed algorithm is superior in runtime to Apriori-like frequent episodes mining algorithm.

KEY WORDS event sequence; frequent episodes; data mining; generalized suffix tree