基于二级索引结构无候选项闭合序列模式挖掘算法*

缪裕青，吴孔玲，朱晓雁，张锦杏
（桂林电子科技大学 计算机科学与工程学院，广西 桂林 541004）

摘　要： 针对 CloSpan 算法分两个阶段挖掘闭合序列模式中第一阶段需要保持候选序列且未充分利用项的位置信息、存在对数据库重复扫描和计算大小的不足，提出了 posCloSpan 算法。算法通过对二级索引结构进行检索实现向前剪枝，避免数据库重复扫描以及对超序索引表、子序索引表的检测，实现非闭合序列的修剪，无须保存候选序列。实验结果证明，算法在处理较长序列以及存在大量重复投影数据库的数据源时，有效降低了时间上的开销。

关键词： 数据挖掘；序列模式挖掘；闭合序列；CloSpan
中图分类号： TP391　　**文献标志码：** A　　**文章编号：** 1001-3695(2012)10-3672-05
doi:10.3969/j.issn.1001-3695.2012.10.018

Closed sequential pattern mining algorithm with no candidate sequence based on two-level index structure

MIAO Yu-qing, WU Kong-ling, ZHU Xiao-yan, ZHANG Jin-xing
(*School of Computer Science & Engineering, Guilin University of Electronic Technology, Guilin Guangxi* 541004, *China*)

Abstract: Aiming at the defects of CloSpan algorithm when mining closed sequential pattern that it needs to maintain the candidate sequences in the first stage and do not make full use of the location information, exists repeatedly scanning database calculating database size, this paper put forward posCloSpan algorithm. By detecting the two-level index structure, the algorithm achieved forward pruning, avoided repeatedly scanning database. At the same time, it trimed non-closed sequences through detecting sup-sequence index table and sub-sequence index table, without saving candidate sequence. Experimental result shows that the algorithm can effectively reduce the time consumption in dealing with longer sequence and the data source that has a large number of duplicated project database.

Key words: data mining; sequential pattern mining; closed sequence; CloSpan

0　引言

序列模式挖掘算法 AprioriAll[1]、SPADE[2]、SPAM[3]、PrefixSpan[4]等都是在数据库中挖掘出所有满足最小支持度的频繁序列，并且对由短序列组成的数据库取得了很好性能。但是当支持度很低或挖掘长序列数据库时，频繁序列会呈指数级增长，算法性能大大降低；同时大量的挖掘结果不仅导致存储空间的巨大消耗，也降低了从挖掘结果中提取有用信息的效率。如何压缩挖掘结果，降低存储空间成为研究热点，于是提出了闭合序列模式挖掘。闭合序列模式是指在支持度相同的条件下，不存在被其他任何包含的序列模式，它不仅可以完全表达结果的完全集，有更精简的结果，而且不存在信息的衰减，对它进行挖掘被认为是压缩挖掘结果的有效途径。

Yan 等人[5]提出的 CloSpan 算法是第一个挖掘闭合序列模式算法，它在 PrefixSpan[4]算法的基础上加入两种剪枝策略，并用哈希算法优化搜索空间，可分为两个阶段：第一阶段产生候选序列，并运用数据库大小哈希、子模式回溯、超模式回溯等策略提前结束序列的增长；第二阶段运用支持度哈希，修剪非闭合序列模式，从而得到全部闭合序列模式。实验证明，CloSpan 算法效率比 PrefixSpan 算法有很大提高。

BIDE 算法[6]克服了需要维护候选序列的不足，基于伪投影，采用双向扩展闭合检测方法，通过向后扫描和跳跃扫描优化技术剪枝搜索空间，比 CloSpan 有更好的算法性能。该算法主要应用于简单数据序列，即序列的一个元素为单个项，如蛋白质序列、网络点击流序列等。随后金沙等人[7]提出的 PosD 算法利用支持度、约束策略和位置信息来减少搜索空间。而林颖提出的 PosD* 算法是在原 PosD 算法中加入了时间约束，进一步减小了搜索空间[7]。

通过研究发现，CloSpan 算法在第一阶段需要维护候选序列，且在模式增长过程中，未充分利用末项的位置信息，对每个投影数据库都要进行扫描统计其大小，存在着重复扫描统计的

收稿日期： 2012-03-14；**修回日期：** 2012-04-24　　**基金项目：** 广西可信软件重点实验室开放基金资助项目；广西研究生科研创新资助项目(2011105950812M22)

作者简介： 缪裕青(1966-)，女，副教授，博士，主要研究方向为数据挖掘、生物数据挖掘(miaoyuqing@guet.edu.cn)；吴孔玲(1986-)，女，硕士，主要研究方向为数据挖掘、序列模式挖掘；朱晓雁(1979-)，女，硕士，主要研究方向为管理学、营销管理；张锦杏(1986-)，男，硕士，主要研究方向为数据挖掘、云计算.

现象。本文提出的 posCloSpan 算法,利用二级索引结构,进行向前剪枝,避免重复扫描投影,同时进行超序检索、子序检索修剪非闭合序列,即可得到闭合序列模式,不需要维护候选序列。实验证明,算法在处理较长序列以及相似度较高的数据集时,有效降低了时间上的消耗。

1　基本术语与问题描述

设 $I=\{i_1,i_2,\cdots,i_n\}$ 是所有不同属性的集合,每个属性又称为项(item)。项集(itemset)是项组成的非空集合,是 I 的子集。

序列(sequence)是项集的有序表,记做 $S=\langle s_1,s_2,\cdots,s_n\rangle$,其中 $s_k(1\leqslant k\leqslant n)$ 是非空项集,称为序列的元素(element)。序列包含的项的个数称为序列的长度,长度为 k 的序列记为 k-序列。

给定两个序列 $A=\langle a_1,a_2,\cdots,a_m\rangle$ 和 $B=\langle b_1,b_2,\cdots,b_n\rangle$,如果存在一组整数 $j_1<j_2<\cdots<j_m$,使得 a_1 包含于 b_{j1}、a_2 包含于 b_{j2}、$\cdots$、a_m 包含于 b_{jm},则称序列 A 包含于序列 B,A 是 B 的子序列,记为 $A\subseteq B$。

序列数据库 D 是元组(sid,S)的集合,其中 sid 为序列号,S 为序列。如果序列 T 是 S 的子序列(即 $T\subseteq S$),称元组(sid,S)包含序列 T。序列 T 在序列数据库 D 中的支持度是数据库中包含 T 的元组个数,即 support$D(T)=|\{(\text{sid},S)\mid(\text{sid},S)\in D\wedge T\subseteq S\}|$,记做 support($T$)。给定支持度阈值 ξ,如果序列 T 在序列数据库中的支持度不低于 ξ,则称序列 T 为序列模式,长度为 l 的序列模式记为 l-模式。

a)前缀。设每个元素中的所有项目按照字典序排列。给定序 $\alpha=\langle e_1,e_2,\cdots,e_n\rangle$,$\beta=\langle e_1',e_2',\cdots,e_m'\rangle$ $(m\leqslant n)$,如果 $e_i'=e_i(i\leqslant m-1)$,$e_m'\subseteq e_m$,并且(e_m-e_m')中的项目均在 e_m'中项目的后面,则称 β 是 α 的前缀。如表 1 所示的序列数据库中,序列 CAA 是序列 CAABC 的前缀,而序列 AA 则不是。

b)投影。给定序列 α 和 β,如果 β 是 α 的子序列,则 α 关于 β 的投影 α'必须满足:β 是 α'的前缀,α'是 α 的满足上述条件的最大子序列。如表 1 中,序列 BC 是序列 ABCB 的子序列,则序列 ABCB 关于序列 BC 的投影是 BCB。

c)后缀。序列 α 关于子序列 $\beta=\langle e_1,e_2,\cdots,e_{m-1},e_m'\rangle$ 的投影为 $\alpha'=\langle e_1,e_2,\cdots,e_n\rangle$ $(n\geqslant m)$,则序列 α 关于子序列 β 的后缀为$\langle e_m'',e_{m+1},\cdots,e_n\rangle$,其中 $e_m''=(e_m-e_m')$。如表 1 中,序列 ABCB 关于子序列 BC 的后缀为 B。

d)投影数据库。设 α 为序列数据库 S 中的一个序列模式,则 α 的投影数据库为 S 中所有以 α 为前缀的序列相对于 α 的后缀,记为 $S|\alpha$。

e)项集扩展。给定一个序列 $s=\langle t_1,\cdots,t_m\rangle$和一个项 α,对任意的 $k\in t_m$,$k<\alpha$,则 $s\diamondsuit_i\alpha=\langle t_1,\cdots,t_m\cup\{\alpha\}\rangle$被称做项集扩展,记为 I 扩展。即向原序列的最后一个项集元素中添加一个比该项集中任何一项都大的新项 α,从而生成新序列。

f)序列扩展。给定一个序列 $s=\langle t_1,\cdots,t_m\rangle$和一个项 α,对于任意的 $k\in t_m$,$k<\alpha$,有 $s\diamondsuit_s\alpha=\langle t_1,\cdots,t_m,\{\alpha\}\rangle$称做序列扩展,记为 S 扩展。即向原序列末尾添加一个新项 α,形成新序列。

g)闭合序列。对于序列模式 s,如果不存在 s 的真超序列 s',并且 s'和 s 有相同的支持度,则称序列模式 s 是闭合的。

闭合序列模式挖掘就是要找出序列数据库中所有闭合序列模式,即支持度不小于用户给定的支持度阈值的所有闭合序列。

2　posCloSpan 算法

2.1　形式化定义

项位置序偶(item position ordered pair)是一个有序对,表示为(sid,fid)。其中,sid 为序列号,表示该项所在的序列;fid 为位置号,表示该项在序列中第一次出现的位置。项位置序偶序列(item position ordered pair sequence)由一系列项位置序偶组成。例如表 1 所示的数据库中,项 C 的位置序偶(1,1)表示项 C 出现在序列 CAABC 的第一个位置;而项 C 的位置序偶序列(1,1)(2,3)(3,1)(4,4)则表明数据库中各序列首次出现项 C 的位置。

项位置序偶表(item position ordered pair table)包含两个域:项位置序偶序列和指针域。指针域指向前缀搜索树中具有该位置序偶序列的频繁项。

频繁项头表由频繁项和指针域构成。指针域指向一张与该指针域相对应的频繁项的位置序偶序列表。

二级索引结构由频繁项头表和项位置序偶表组成,如图 1 所示。

表1　序列数据库

sid	序列
1	CAABC
2	ABCB
3	CABC
4	ABBCA

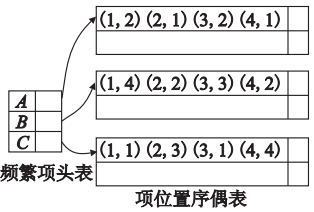

图1　二级索引结构

扩展前缀搜索树中的节点包括五个域:频繁项、层数、支持数、父指针、闭合标记。为简便起见,称扩展前缀搜索树为前缀搜索树。前缀搜索树中,当前节点代表从根节点到该节点的序列,节点层数代表序列长度,支持度表示序列支持度,父指针域指向该节点的上一层父节点。

超序索引表包括频繁项和指针链两部分,指针链指向前缀树中以此频繁项为结束项的一系列频繁序列。

子序索引表由频繁项和指针链组成,指针链中的每个指针指向前缀搜索树中以该频繁项为末项的闭合序列。

定理 1　给定两个序列 s 和 s',$s\subseteq s'$且有相同末项 γ,那么序列 s 和 s'的投影数据相等的充分必要条件是序列 s 末项 γ 的位置序偶序列与序列 s'末项 γ 的位置序偶序列相等。

证明　由项位置序偶序列的定义可知,它表示的是某项在数据库中各序列第一次出现的位置。显而易见,当末项位置序偶序列相等时,则表明处在数据库中同一位置,对同一位置投影产生的投影数据库相等;反之,当投影数据库相等,且是对具有相同末项的序列投影,则它们的末项处在数据库中相同位置,具有相同的位置序偶序列。

引理 1　给定两个序列 s 和 s',$s\subseteq s'$且有相同末项 γ,如果序列 s 末项 γ 的位置序偶序列与序列 s'末项 γ 的位置序偶序列相等,则对于任意的 ε,$s\diamondsuit\varepsilon$ 的支持度等于 $s'\diamondsuit\varepsilon$。

推论 1　扩展子模式回溯。给定一个序列 $s<s'$,$s\supset s'$且有相同末项 γ,如果序列 s 末项 γ 的位置序偶序列与序列 s'末项 γ 的位置序偶序列相等,则停止搜索前缀搜索树中 s'的任何

后代。

推论 2 扩展超模式回溯。给定一个序列 $s<s'$,$s\subset s'$且有相同末项 γ,如果序列 s 末项 γ 的位置序偶序列与序列 s'末项 γ 的位置序偶序列相等,则停止搜索前缀搜索树中 s'的任何后代,把 s 的后代移植给 s',作为 s'的后代。

定理 2 给定一个当前序列 s,对于任何一个先于 s 发现的闭合序列 s_1,若有 $s\subset s_1$,则不存在已发现的闭合序列 $s_2\subset s$。

证明 假设存在 $s_2\subset s$,又由条件 $s\subset s_1$可知,$s_2\subset s\subset s_1$,由此得出 $s_2\subset s_1$。又因为 s_1 和 s_2 都是闭合序列,不存在包含关系,得出的结论与此相矛盾,所以假设不成立。此定理可以用来当检查出存在超序时,只进行超序检查,而不必进行子序的检查。

定理 3 支持度剪枝。对于支持度为 n 的当前序列 s ,当进行模式增长时,发现存在有支持度为 n 的频繁项 α,那么 s 一定不是闭合序列。

证明 因为 s 的支持度为 n,α 的支持度也为 n,那么存在序列 $s_1=\langle s,\alpha\rangle$ 的支持度也为 n,且 $s\subset s_1$,所以 s 不是闭合序列。

定理 4 超序剪枝。对于当前闭合序列 s,若超序索引表指向的所有序列中,存在序列 $s_1\supset s$,那么 s 不是闭合序列。

证明 超序索引表指向的是前缀搜索中的频繁序列,若当前序列是其子序列,那么显然由闭合模式的定义可知,当前序列不是闭合序列。又由定理 2 可知,当检测出存在超序,则可以跳过子序的检测。

定理 5 子序剪枝。对于当前闭合序列 s,若在子序索引表指向的所有序列中存在 s_1,且 $s_1\subset s$,那么 s_1 不是最终所求的闭合序列,需要从子序索引表中删除。

证明 子序剪枝在超序剪枝后进行,当闭合序列 s 经超序剪枝未被修剪,则可以说明序列 s 不是任何已发现的闭合序列的子序列,但是仍然有可能包含着已发现的某些闭合序列。那么对于这些不是最终的闭合序列,必须进行修剪。

2.2 posCloSpan 算法描述

posCloSpan 主要步骤如下:

a)扫描数据库一次,找出所有频繁项,并记录各频繁项在数据库序列中首次出现的位置和支持度,形成各项的位置序偶序列。根据频繁项对搜索空间进行划分,初始化二级结构。

b)在各个子搜索空间上递归,进行以下操作:

(a)对当前节点 α,获取其位置序偶序列。

(b)从二级结构中查找是否存在有相同 α 位置序偶序列的子序列或超序。

(c)若存在,则根据推论 1 和 2 进行扩展子模式回溯或扩展超模式回溯,提前终止序列模式增长。返回。

(d)若不存在,则将 α 位置序偶序列加入二级结构,节点 α 加入前缀搜索树。对 α 进行投影,寻找各局部频繁项,并记录各局部频繁项的位置序列以及支持度。

(e)根据各局部频繁项的支持度,对当前节点 α 所代表序列进行闭合性判断。若存在支持度与 α 相同的局部频繁项,则该序列一定是非闭合序列,然后进行子序索引表闭合性检索,再进行模式增长,返回步骤(a)。若各局部频繁项的支持度均小于 α 的支持度,则需要先进行超序索引表闭合性检索,然后进行子序索引表闭合性检索,最后进行模式增长,返回步骤(a)。

递归进行各步骤操作,直到不产生新闭合序列。最后形成的子序索引表指向的闭合序列就是最终结果。

下面给出 posCloSpan 算法的形式化描述。

算法 posCloSpan($s\diamond\alpha$,$D_s\diamond\alpha$,min_sup,T,FP,SupH,SubH)

输入:序列 $s\diamond\alpha$,投影数据库 $D_{s\diamond\alpha}$,最小支持度 min_sup,前缀搜索树 T,二级索引结构 FP,超序索引表 SupH,子序索引表 SubH。

输出:子序索引表 SubH,前缀搜索树 T。

1:调用函数 pos_Label = chenkFP($s\diamond\alpha$,FP,α_pseq,T)

2:if pos_Label = 1 then return;

3:end if

4:if pos_Label = 0 then

5:设 sup_Label = 1,扫描 $D_{s\diamond\alpha}$一次,找出所有局部频繁项,并记录它们的支持度以及在投影数据各序列中首次出现的位置,即各项的位置序偶序列;

6:if $\exists$ support(α) = support(γ),γ 为局部频繁项中任何一项;

7:then return;

8:else if support(α) > support(γ),γ 为局部频繁项中任何一项 or 不存在局部频繁项;

9:then call sup_Label = supTheck($s\diamond\alpha$,SupH)

10:if sup_Label = 0

11:then 对于当前分支上任何闭合标记为 0 的节点

12:call sub_Theck($s\diamond\alpha$,SupH,SubH)

13:end if

14:end if

15:end if

16:对局部频繁项中的任何一项 γ

17:call posCloSpan($s\diamond\alpha\diamond\gamma$, $D_{s\diamond\alpha\gamma}$, min_sup, T, FP, SupH, SubH)

18:return;

函数 chenkFP($s\diamond\alpha$,FP,α_pseq,T)

输入:序列 $s\diamond\alpha$,二级结构 FP,α 的位置序偶序列 α_pseq,前缀搜索树 T。

输出:更新后的二级结构 FP,更新后前缀搜索树 T。

1:根据频繁项 α 以及 α 的位置序偶序列 α_pseq 检查二级结构中是否存在相同的 α_pseq;

2:if 二级结构中不存在相同的 α_pseq

3:then 把 α_pseq 加入到二级结构相应的位置序偶表中;把 α 加入前缀搜索树 T;pos_Label = 0;

4:else 检测 α_pseq 指向的序列中是否存在 $s\diamond\alpha$ 的子序或超序;

5:if 存在

6:then 进行扩展子模式回溯或是扩展超模式回溯;

7:修改前缀搜索树 T;

8:pos_Label = 1;

9:end if

10:return pos_Label;

函数 supTheck($s\diamond\alpha$,SupH)

输入:序列 $s\diamond\alpha$,超序索引表 SupH。

输出:更新后的 SupH。

1:设 sup_Label = 0;

2:根据频繁项 α,检索超序索引表中是否存在 α;

3:if 存在

4:then 对与 α 对应的指针链中指向的每个序列 β do

5:if $s\diamond\alpha\subset\beta$

6:then sup_Label = 1;

7:return sup_Label;

8:break;

9:end if;

10:end if;

11:把序列 $s\diamond\alpha$ 中从 α 开始到第一个标记为 1 的父节点之间的所有频繁序列加入到超序索引表 SupH 中;

12:return sup_Label;

函数 sub_Theck($s\diamond\alpha$,SupH,SubH)

输入:序列 $s\diamond\alpha$,超序索引表 SupH,子序索引表 SubH。

输出:更新后的超序索引表 SupH 和子序索引表 SubH。

1:设从 α 到第一个标记为 1 的父节点之前的所有频繁项组成的集合为 A;

2:for 每个 $\alpha\in A$ do

3:在子序索引表中查找是否存在 α;

4:如果存在,则检查与 α 相对应的指针链指向的每个序列 β 是否是当前序列的子序列;

5:如果是当前序列的子序列,则先从 SubH 中删除 β,再从 SupH 中删除 β。

6:end for;

7:将 α 加入 SubH 中;

8:return;

2.3　实例说明

下面以图 1 所示数据库说明 posCloSpan 算法的主要过程,假设最小支持度 min_sup = 2。

a)扫描数据库一次,得到频繁项 A:4,B:4,C:4,同时记录它们的位置序偶序列 A:(1,2)(2,1)(3,2)(4,1);B:(1,4)(2,2)(3,3)(4,2);C:(1,1)(2,3)(3,1)(4,4)。根据频繁项初始化二级结构。

b)划分搜索空间,根据步骤 a)的结果,将搜索空间划分为三个部分。

c)在各个搜索空间上递归进行挖掘,应用各种剪枝策略,直到产生所有闭合序列。

步骤 c)的具体说明如下:

第一层根据默认字典排序。

a)对频繁项 A:4,取它的位置序偶序列 A:(1,2)(2,1)(3,2)(4,1)。

b)在二级结构中寻找是否存在与 A:(1,2)(2,1)(3,2)(4,1)相同的位置序偶序列,此时二级结构为空,所以把此位置序列加入二级结构中,把 A 加入到前缀搜索树 T 中,并把 A 连接到二级结构相应指针域。

c)对 A 进行投影,扫描投影数据库,得到 A 的第二层频繁项 A:2,B:4,C:4,同时记录它们的位置序偶序列 A:(1,3)(4,5);B:(1,4)(2,2)(3,3)(4,2);C:(1,5)(2,2)(3,4)(4,4)。

d)因为 support〈A〉= support〈AB〉= support〈AC〉,所以〈A〉不可能是闭合序列,标记为 0,不加入超序索引表 SupH 和子序索引表 SubH 中。此时结果如图 2 所示。

第二层则根据深度优先搜索策略以及默认字典排序,取第一层 A 的第二层频繁项 A:2,B:4,C:4 中的 A:2 进行与第一层相同的操作。

a)对频繁项 A:2,取它的位置序偶序列 A:(1,3)(4,5)。

b)在二级结构中寻找是否存在与 A:(1,3)(4,5)相同的位置序偶序列。因不存在,所以把此位置序列加入二级结构中,把 A 加入到前缀搜索树 T 中,并把 A 连接到二级结构相应指针域。

c)对 A 进行投影,扫描投影数据库,此时 A 不存在下一层的频繁节点。

d)对当前序列进行超序检测,调用函数 supTheck($s\diamond\alpha$,SupH)。此时 SupH 为空,不存在超序,将 A 与上层节点 A 加入到 SupH,返回 sup_Label 值为 0,继续调用函数 sub_Theck($s\diamond\alpha$,SupH,SubH)进行子序检测,此时 SubH 为空,所以将 A 加入 SubH,并把 A 的闭合标志记为 1。此时如图 3 所示。

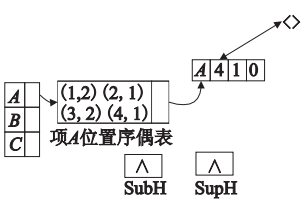

图2　加入A后的前缀树

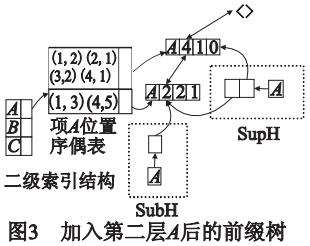

图3　加入第二层A后的前缀树

由于节点 A 无下层频繁节点,继续对第二层的频繁项 B:4,C:4 做相同的操作,得到图 4 所示的结果。图中虚线表示存在扩展子模式回溯或扩展超模式回溯,提前结束模式增长。

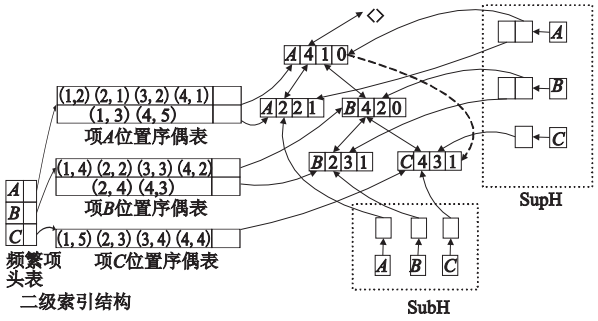

图4　第一层A挖掘结束的前缀树

做完上述操作后,递归返回第一层,对剩余频繁项 B:4,C:4 进行同样操作,即可得到最后结果,如图 5 所示。

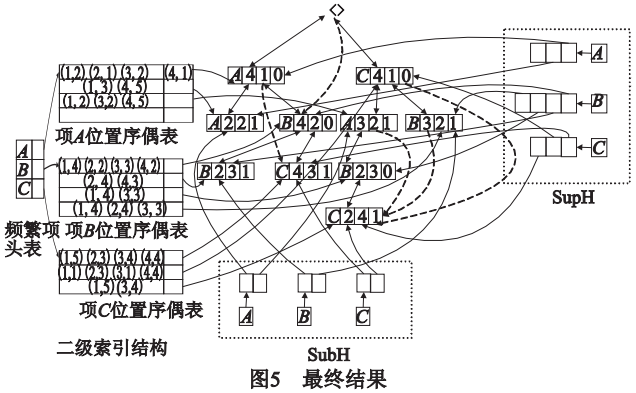

图5　最终结果

遍历图中的子序索引表指针链指向的序列,即得到所有的闭合序列〈AA〉:2,〈ABB〉:2,〈ABC〉:4,〈CA〉:3,〈CABC〉:2,〈CB〉:3。

3 实验结果

本实验环境为 Intel® Core™ i3-2100 CPU 2.1 GHz,2.00 GB 内存,Windows 7 操作系统。算法采用 C++ 编写,在 Visual C++ 6.0 环境下编译。实验数据采用 IBM Quest Synthetic Data Generator 生成,测试数据参数说明如下:D 表示顾客数,默认值为 1 000;C 表示每个顾客的平均事务数,默认值为 10;T 表示每个事务平均项数;N 为总项数,默认为 1 000;S 表示最长序列平均事务数;I 表示最长模式平均项数。分别对数据集 D10C10T2.5N10S6I2.5 和 D5C20T2.5N5S10I2.5 进行测试,实验结果如图 6、7 所示。

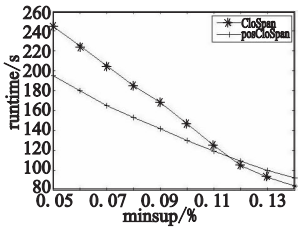

图6 D10C10T2.5N10S6I2.5 数据集执行时间比较

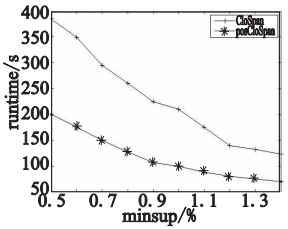

图7 D5C20T2.5N5S10I2.5 数据集执行时间比较

从图 6 中可知,当支持度较低时,相比 CloSpan 算法,posCloSpan 算法的优势较为明显,但是随着支持度的增加,优势减弱;当支持度较高时,CloSpan 算法反而优于 posCloSpan 算法。这主要是因为支持度较低时,会产生大量的投影数据库以及候选序列,CloSpan 算法需要每次扫描数据库计算大小,这其中存在着大量的重复扫描。而 posCloSpan 算法,通过对比项的位置序偶序列,就可以避免重复扫描,同时 posCloSpan 算法通过超序剪枝、子序剪枝、相同支持度剪枝等策略避免了 CloSpan 算法中对大量候选序列的维护,因此效率较明显优于 CloSpan 算法。但是随着支持度的增加,投影数据库大量减少,CloSpan 算法优势体现;当支持度较高时,其效率高于 posCloSpan。

相对于 D10C10T2.5N10S6I2.5 数据集,D5C20T2.5N5S10I2.5 数据集是行数较少、长度长、相似度较高的数据。从图 7 中可知,posCloSpan 算法在不同支持度下效率均高于 CloSpan 算法。这得益于 posCloSpan 算法避免了对大量重复投影数据库的扫描计算以及通过各种剪枝策略,无须维护候选序列,是 posCloSpan 的优势体现。通过对比两图可知 posCloSpan 算法更适合挖掘行数少、长度长,且相似度较高的数据。

实验证明,posCloSpan 算法是正确的,它能够完全挖掘出闭合序列,同时它更适合挖掘挖掘行数少、长度长、相似度较高的数据。

4 结束语

随着应用的推广,序列模式挖掘成为研究热点。本文在深入分析 CloSpan 算法的基础上,提出基于二级索引结构无候选序列闭合序列挖掘(posCloSpan)算法。通过分析与实验,证明了 posCloSpan 算法的有效性。下一步考虑如何将本算法思想与通过 2-序列连接实现序列增长方式相结合,进一步提高挖掘效率。

参考文献:

[1] AGRAWAL R, SRIKANT R. Mining sequential patterns[C]//Proc of the 11th International Conference on Data Engineering. Washington DC: IEEE Computer Society, 1995:3-14.

[2] ZAKI M J. SPADE: an efficient algorithm for mining frequent sequences [J]. Machine Learning Journal, 2001, 42(1):31-60.

[3] AYRES J, FLANNICK J, GEHRKE J, *et al*. Sequential pattern mining using a bitmap representation [C]//Proc of the 8th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. New York: ACM Press, 2002:429-435.

[4] PEI Jian, HAN Jia-wei, MORTAZAVI-ASL B, *et al*. PrefixSpan: mining sequential patterns efficiently by prefix-projected pattern growth[C]//Proc of of the 17th International Conference on Data Engineering. 2001:215-224.

[5] YAN Xi-feng, HAN Jia-wei, AFSHAR R. CloSpan: mining closed sequential patterns in large datasets[C]//Proc of the 3rd SIAM International Conference on Data Mining. 2003:166-177.

[6] WANG Jian-yong, HAN Jia-wei. BIDE: efficient mining of frequent closed sequences[C]//Proc of the 20th International Conference on Data Engineering. Washington DC: IEEE Computer Society, 2004: 79-90.

[7] 金沙,邓玉成,张翠肖,等. 闭合序列模式挖掘算法[J]. 计算机工程与设计,2006,27(3):514-517.

[8] LOEKITO E, BAILEY J, PEI J. A binary decision diagram based approach for mining frequent subsequences [J]. Knowledge and Information Systems, 2010, 24(2):235-268.

[9] 公伟,刘培玉,贾娴. 基于改进 PrefixSpan 的序列模式挖掘算法[J]. 计算机应用,2011,31(9):2405-2407.