挖掘闭合模式的高性能算法*

刘君强[1+], 孙晓莹[1], 庄越挺[2], 潘云鹤[2]

[1](杭州商学院 计算机信息工程学院,浙江 杭州 310035)

[2](浙江大学 人工智能研究所,浙江 杭州 310027)

Mining Frequent Closed Patterns by Adaptive Pruning

LIU Jun-Qiang[1+], SUN Xiao-Ying[1], ZHUANG Yue-Ting[2], PAN Yun-He[2]

[1](School of Computer Information Engineering, Hangzhou Institute of Commerce, Hangzhou 310035, China)

[2](Institute of Artificial Intelligence, Zhejiang University, Hangzhou 310027, China)

+ Corresponding author: Phn: +86-571-88835193, E-mail: liujunq@mail.hzic.edu.cn, http://www.hzic.edu.cn

Received 2002-10-24; Accepted 2003-09-05

Liu JQ, Sun XY, Zhuang YT, Pan YH. Mining frequent closed patterns by adaptive pruning. *Journal of Software*, 2004,15(1):94~102.

http://www.jos.org.cn/1000-9825/15/94.htm

Abstract: The set of frequent closed patterns determines exactly the complete set of all frequent patterns and is usually much smaller than the laster. Yet mining frequent closed patterns remains to be a memory and time consuming task. This paper tries to develop an efficient algorithm to solve this problem. The compound frequent item set tree is employed to organize the set of frequent patterns, which consumes much less memory than other structures. The tree is grown quickly by integrating depth first and breadth first search strategies, opportunistically choosing between two different structures to represent projected transaction subsets, and heuristically deciding to build unfiltered pseudo or filtered projections. Efficient pruning methods are used to reduce the search space. The balance of the efficiency and scalability of tree growth and pruning maximizes the performance. The experimental results show that the algorithm is a factor of five to three orders of magnitude more time efficient than several recently proposed algorithms, and is also the most scalable one. It can be used in the discovery of non-redundant association rules, sequence analysis, and many other data mining problems.

Key words: knowledge discovery; data mining; frequent closed pattern; association rule

摘 要: 频繁闭合模式集惟一确定频繁模式完全集并且尺寸小得多,然而挖掘频繁闭合模式仍然是时间与存

* Supported by Zhejiang Provincial Natural Science Foundation of China under Grant No.602140 (浙江省自然科学基金); the National Natural Science Foundation of China under Grant No.60272031 (国家自然科学基金); the National Research Foundation for the Doctoral Program of Higher Education of China under Grant No.20010335049 (国家教育部博士点基金); the Key Science-Technology Project of the National "Ninth Five-Year-Plan" of State Development Planning Commission of China under Grant No.JGJ-1998-2077 (国家计委"九五"重大科技攻关项目)

作者简介: **刘君强**(1962－),男,浙江杭州人,博士,教授,主要研究领域为人工智能,数据挖掘;**孙晓莹**(1963－),女,讲师,主要研究领域为管理信息系统;**庄越挺**(1965－),男,博士,教授,博士生导师,主要研究领域为计算机图形学,人工智能;**潘云鹤**(1946－),男,教授,博士生导师,主要研究领域为人工智能,计算机辅助设计.

储开销很大的任务.提出一种高性能算法来解决这一难题.采用复合型频繁模式树来组织频繁模式集,存储开销较小.通过集成深度与宽度优先策略,伺机选择基于数组或基于树的模式支持子集表示形式,启发式运用非过滤虚拟投影或过滤型投影,实现复合型频繁模式树的快速生成.局部和全局剪裁方法有效地缩小了搜索空间.通过树生成与剪裁代价的平衡实现时间效率与可伸缩性最大化.实验表明,该算法时间效率比其他算法高 5 倍到 3 个数量级,空间可伸缩性最佳.它可以进一步应用到无冗余关联规则发现、序列分析等许多数据挖掘问题.

关键词: 知识发现;数据挖掘;频繁闭合模式;关联规则

中图法分类号: TP311 **文献标识码**: A

频繁模式挖掘在许多数据挖掘任务中起着重要作用,但挖掘得到的模式与关联规则的数量往往大得惊人,难以理解与运用.一个长度为 l 的频繁模式意味着有 2^l-1 个频繁子模式需要逐一搜索.当 l 较大时,挖掘效率主要受制于 CPU,而不是 I/O 操作.然而,实际应用经常需要挖掘特别长的模式,如 DNA 序列等.

Pasquier 等人提出了频繁闭合模式的概念[1].它惟一地决定所有频繁模式的准确支持率,并且尺寸比频繁模式集小几个数量级.Pasquier 等人设计的算法 A-Close[1],以 Apriori[2]为基础,采用自底向上、宽度优先的搜索策略,通过构造“生成子”集合,逐层求出所有频繁闭合模式.通过剪裁,减少了搜索范围,但没有解决宽度优先搜索算法内在的模式匹配的高 CPU 开销和重复扫描数据库的高 I/O 开销的问题.

Pei 等人提出的算法 CLOSET[3]以 FP-Growth 为基础,采用 FP-Tree 来表示模式支持集,通过深度优先搜索来挖掘频繁闭合模式.其困难是,递归构造“条件 FP-Tree”的 CPU 开销和存储开销很大.同时,CLOSET 采用分割法处理大型数据集,检查局部频繁闭合模式的全局闭合性和频繁性的代价非常高.Burdick 等人提出的深度优先搜索算法 MAFIA[4]采用垂直二进制位图表示模式支持集,当项目平均支持率低于 1/32 时,存储效率远低于数组表示法,并且重建、计数和预处理开销很高.Zaki 等人提出的 CHARM[5]是现有算法中性能最好的.CHARM 采用垂直格式的事务标识集来表示模式支持集,对项目集与事务标识集进行双向搜索,剪裁效率很高.CHARM 的困难在于事务标识集无论采用 tidset 还是 diffset 表示,存储开销都非常大,并且投影操作效率不高,对于密集型或存在特别长模式的数据集,CHARM 效率与可伸缩性较差.

本文提出了效率高与可伸缩性好的频繁闭合模式挖掘新算法.它采用新颖的复合型频繁模式树组织频繁模式集,支持高效的局部与全局剪裁,继承 OpportuneProject[6]伺机投影的思想,以深度优先搜索为主、辅以宽度优先搜索,混合使用基于数组和基于树的模式支持集表示法,投影效率高、存储开销低,并较好地平衡了树生成与剪裁的代价.

1 问题的描述

定义 1. 数据源 $\boldsymbol{D}=(\boldsymbol{O},\boldsymbol{I},\boldsymbol{R})$,其中 $\boldsymbol{O}$ 是数据对象的有限集合,$\boldsymbol{I}$ 是数据属性的有限集合,$\boldsymbol{R}\subseteq\boldsymbol{O}\times\boldsymbol{I}$ 是对象与属性间的二元关系.二元组 $(o,i)\in\boldsymbol{R}$ 表示对象 $o\in\boldsymbol{O}$ 具有属性 $i\in\boldsymbol{I}$.

数据属性(值)又称为项目,因而 $\boldsymbol{I}=\{i_1,i_2,\ldots,i_m\}$,其中文字 i_k 称为一个项目,表示一个属性值.如图 1 所示数据源,$\boldsymbol{I}=\{a,b,c,d,e,f,g,h,i,k,l,m,n,o,p,s\}$.数据对象又称为事务,可以用标识符和组成属性值(项目)表示,因而 $\boldsymbol{O}=\{(tid_1,t_1),\ldots,(tid_n,t_n)\}$,其中 (tid_k,t_k)表示一个对象(事务),tid_k 是对象(事务)标识符,$t_k\subseteq\boldsymbol{I}$ 是对象(事务)的属性值(项目)集合.如图 1 所示数据源,$\boldsymbol{O}$ 由 5 个事务组成.

TID	Items
01	*a c d f g l m p*
02	*a b c f l m o*
03	*b c h m o*
04	*b f k p s*
05	*a c e f l m n p*

Fig.1 Example dataset

图 1 数据集示例

定义 2. 公共模式映射 $f:2^{\boldsymbol{O}}\to2^{\boldsymbol{I}}$,$f(O)=\{i\in\boldsymbol{I}\mid\forall o\in O\subseteq\boldsymbol{O},(o,i)\in\boldsymbol{R}\}$,称 $f(O)$是 $O\subseteq\boldsymbol{O}$ 的公共模式.

模式 I 是 $\boldsymbol{I}$ 的一个子集,即 $I\subseteq\boldsymbol{I}$,因而模式又称为项目集.如果模式 I 由 k 个项目组成,则称 I 为 k-模式,或 k-项目集.而公共模式则是数据对象子集 $O\subseteq\boldsymbol{O}$ 中普遍存在的属性值集合.如图 1 所示,事务 01,02 和 03 组成子集的公共模式是 $\{c,m\}$.

定义 3. 投影映射 $g:2^{\boldsymbol{I}}\to2^{\boldsymbol{O}}$,$g(I)=\{o\in\boldsymbol{O}\mid\forall i\in I\subseteq\boldsymbol{I},(o,i)\in\boldsymbol{R}\}$,称 $g(I)$为模式 $I\subseteq\boldsymbol{I}$ 的支持集.

若 $I\subseteq t_k$,则称模式 I 被事务(tid_k,t_k)所支持,$g(I)$是支持 I 的事务集合.如图 1 所示,模式$\{c,m\}$支持集由事务 01,02,03 和 05 组成.

定义 4. $2^{\boldsymbol{I}}$ 上闭包算子 $h=f\circ g,h(I)=f(g(I))$.模式 $C\subseteq\boldsymbol{I}$ 是闭合的,当且仅当 $h(C)=C$.

如图 1 所示,有 $h(\{c,m\})=\{c,m\}$,所以$\{c,m\}$是闭合模式.

定义 5. $I\subseteq\boldsymbol{I}$ 的绝对支持率为 $support(I)=\|g(I)\|$,相对支持率为$\|g(I)\|$除以$\|\boldsymbol{O}\|$的值.

模式 I 的绝对支持率是包含 I 的事务的数目,相对支持率是包含 I 的事务的百分比.如图 1 所示,$\{c,m\}$绝对支持率为 4,相对支持率为 80%.

定义 6. (闭合)模式 $C\subseteq\boldsymbol{I}$ 是频繁的,如果 $support(C)\geq\min sup$.

如图 1 所示的数据源,取 minsup 为 3,则有 18 个频繁模式,5 个频繁闭合模式.

2 复合型频繁模式树及其生成

2.1 复合型频繁模式树

定义 7. 给定数据源 $\boldsymbol{D}=(\boldsymbol{O},\boldsymbol{I},\boldsymbol{R})$,项目排序 $\prec$,最小支持率 minsup,复合型频繁模式树 CFIST(compound frequent item set tree)由结点集 V 和有向边集 E 组成.每个结点 $v\in V$ 记作五元组(i,w,A,O,I),其中 $i\in\boldsymbol{I}$ 是用于标识该结点的项目,记作 $v.item$;w 是根到 v 路径所表示模式的支持数,称为权重,记作 $v.weight$;$A\subset\boldsymbol{I}$ 是附加项目集(auxiliary item set)用于表征复合结点,记作 $v.AIS$;$O\subseteq\boldsymbol{O}$ 是根到 v 路径所表示模式的支持集,称为投影事务子集(projected transaction subset),记作 $v.PTS$;$I\subseteq\boldsymbol{I}$ 是后代结点中出现的项目的集合,称为局部项目子集(item list),记作 $v.IL$.结点 p 到 c 的有向边记为$\langle p,c\rangle\in E$,p 是父母,c 是子女.任意结点各子女标识项目排列和任意路径上标识项目排列符合 $\prec$.

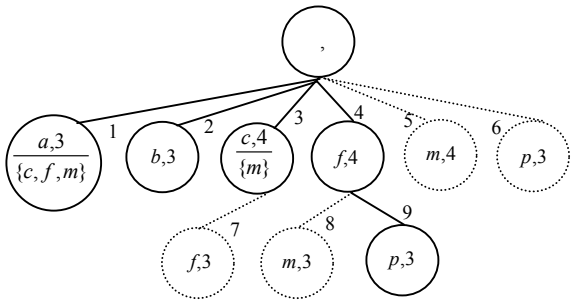

Fig.2 Compound frequent item set tree
图 2 复合型频繁模式树

推论 1. 给定 CFIST 结点 $p=(i_p,w_p,A_p,O_p,I_p)$是 $c=(i_c,w_c,A_c,O_c,I_c)$ 的父母,则 $i_c\in I_p,w_c=\|O_c\|\geq\min sup$,$A_c\subset I_p,O_c=g(\{i_c\})\cap O_p=g(\{i_c\}\cup A_c)\cap O_p,I_c=\{i\in I_p|i_c\prec i\wedge i\notin A_p\}$.

复合型频繁模式树是有序树,以一种压缩表示方法完整地表达了频繁模式集.如图 1 所示的数据源,取 minsup 为 3,取 $\prec$ 为字典序,则频繁模式完全集的 CFIST 如图 2 所示.

定义 8. 在从根结点到任意结点 v 的路径上,各结点标识项目的集合称为特征项目集,记作 $kis(v).kis$ 并上各结点附加项目集称为完全项目集,记作 $fis(v)$.

对于结点 v,$kis(v)$并上附加项目集的任意子集表示一个频繁模式,v 的权重是模式支持数.一条具有复合结点的路径压缩表达了多个频繁模式.例如,图 2 中的结点 3 表达了两个频繁模式:$\{c\}$和$\{c,m\}$,结点 3 和结点 7 结成的路径表达了两个频繁模式:$\{c,f\}$和$\{c,f,m\}$,$fis(v)$是候选的频繁闭合模式.

2.2 复合结点的形成

CFIST 复合结点是通过合并得到的.如果不引入复合结点,则图 2 中的结点 1 需要用 9 个结点来表示.

定理 1. 给定 CFIST 结点 p 是结点 c 的父母,若 $p.weight=c.weight$,则可将 c 并入 p.

证明:设 $p=(i_p,w_p,A_p,O_p,I_p)$,$c=(i_c,w_c,A_c,O_c,I_c)$.因为 $O_c=g(\{i_c\})\cap O_p\subseteq O_p$,已知 $w_c=w_p$,即 $\|O_c\|=\|O_p\|$,所以有 $O_p=O_c$.将 c 并入 p 得到新的复合结点 p',它与 p 和 c 具有相同的闭包.又由于 $I_c=\{i\in I_p|i_c\prec i\wedge i\notin A_p\}\subseteq I_p,i_c\notin I_c$,故 $I_c\subseteq I_p-\{i_c\}=I_{p'}$,所以 p'包含 p 和 c 代表的模式. □

CFIST 的生成过程实际上是确定各结点所表示的模式支持集的过程.任意结点 p 支持集 $p.PTS$ 由所有支持 $fis(p)$的事务组成.根结点 PTS 就是 $\boldsymbol{O}$,其他结点 PTS 由父亲 PTS 投影而来.

3 复合型频繁模式树的剪裁与包含关系的检查

3.1 局部剪裁

定理 2. 给定 CFIST 结点 p 和 n 是兄弟结点,若 $p.weight=n.weight$ 并且 $n.item\in p.AIS$,则 n 可以剪裁掉.

证明:设 $p=(i_p,w_p,A_p,O_p,I_p)$,$n=(i_n,w_n,A_n,O_n,I_n)$,父母 $a=(i_a,w_a,A_a,O_a,I_a)$.因为 $\{i_n\}\subseteq A_p\subseteq\{i_p\}\cup A_p$,有 $O_n=g(\{i_n\})\cap O_a\supseteq g(\{i_p\}\cup A_p)\cap O_a=O_p$.已知 $w_p=w_n$,即 $\|O_p\|=\|O_n\|$,所以有 $O_p=O_n$,p 和 n 具有相同闭包.又因为 $i_n\in A_p$,$i_p\prec i_n$,得 $I_n\subseteq I_p$,所以 p 分枝必定包含 n 分枝. □

如图 2 所示,结点 3 和结点 5 是兄弟,权重都为 4,而结点 5 的标识项目 m 包含于结点 3 的附加项目集,所以剪裁掉结点 5.

3.2 分枝包容关系检查

定理 3. 给定 CFIST 结点 p 和 $q\neq p$,若 $p.weight=q.weight$,并且 $kis(p)\subset fis(q)$,则 p 可以剪裁掉.

证明:已知 $kis(p)\subset fis(q)$,得 $g(kis(p))\supset g(fis(q))$.显然,$g(kis(p))=g(fis(p))$,所以 $g(fis(p))\supset g(fis(q))$.又因为 $p.weight=q.weight$,即 $\|g(fis(p))\|=\|g(fis(q))\|$,得 $g(fis(p))=g(fis(q))$.又 $kis(p)\subset fis(q)$,有 $fis(q)\prec kis(p)$,得 $fis(q)\prec fis(p)$,即 q 先于 p 生成.必存在先于 p 生成的结点 r,有 $fis(r)=fis(p)\cup fis(q)$,显然 $g(fis(r))=g(fis(p))\cup g(fis(q))$.即 r 分枝必定包容 p 分枝,$fis(p)$不是闭合的,所以 p 可以剪裁掉.

如图 2 所示,结点 1 和结点 7 的权重都为 3,而结点 7 的特征项目集 kis(结点 7)=$\{c,f\}$被结点 1 的完全项目集 fis(结点 1)=$\{a,c,f,m\}$所包含,所以剪裁掉结点 7.由于 $kis(p)$是 $fis(p)$的子集,所以用定理 3 作分枝包容关系检查比直接检查的效率要高.

3.3 快速杂凑法

应用定理 3 检查分枝包容关系的直接方式是将已生成模式链入用支持率作键值的 hash 链表中.但是,很多无关模式可以具有相等的支持率,显然会降低应用 hash 链表的效果.根据闭合模式定义,X_p 被 X_q 所包含是指 $X_p\subset X_q$ 并且 $g(X_p)=g(X_q)$.然而,直接检查模式支持集是否相同的代价非常高.

本文采用一种折衷方法,就是以 $g(X_p)$的某个线性函数,比如 $g(X_p)$所有事务标识符的简单和(SumOfTids),作为 hash 键值.这样,检查 X_p 是否被 X_q 所包容的步骤依次为判定 X_p 和 X_q 是否在同一 hash 链中,X_p 和 X_q 的 hash 键值是否相等,X_p 和 X_q 的支持率是否相等,$kis(X_p)\subset X_q$ 是否成立.

图 2 中的结点 1、7 和 8 键值为 8,且结点 7、8 被 1 包容;结点 2 键值为 9;结点 6 和 9 键值为 10,且结点 6 被 9 包容.图中虚线表示的是被剪裁掉的结点,最终剪裁后得到所有频繁闭合模式(5 个).

3.4 复合结点的分解

由于复合结点的存在使得各结点的完全项目集排列不一定符合给定次序 $\prec$,造成挖掘结果的应用,比如规则抽取,需要进行高 CPU 开销的模式匹配.因此,必须分解复合结点,方法如下:

设 n 是 CFIST 复合结点,其辅助项目集为 $n.AIS$ 非空,通过周游以 n 为根的子树,扩散 $n.AIS$ 中的每个项目 i,即对于周游遇到的每个结点 p 执行如下操作:

(1) 如果 $i\prec p.item$,即项目 i 排列在 p 的标识项目之前,则

(1.1) 如果 p 有兄弟结点的标识项目也是 i,则令 s 表示该兄弟结点;否则,创建 p 的子女结点 s,令 $s=(i,p.weight,\varnothing,\text{false})$.

(1.2) 将以 p 为根的子树作为结点 s 的一个分枝,则使 p 成为 s 的一个子女结点.

(1.3) 停止周游 p 以下的子树.

(2) 如果 $p.item\prec i$,则

(2.1) 如果 $p.tag$ 为真,即 $fis(p)$是一个闭合模式,则创建 p 的一个新子女 c,令 $c=(i,p.weight,\varnothing,\text{true})$,并置 $p.tag$ 成假.

(2.2) 继续周游 p 以下的子树.

(3) 如果 *p.item* 就是 *i*,则停止周游 *p* 以下的子树.

最终,复合结点 *n* 变成为原子结点(*n.item*,*n.weight*,∅,false).图 3 说明了按此算法分解复合结点的过程.图 4 是分解图 2 的结果(已剪裁完毕).

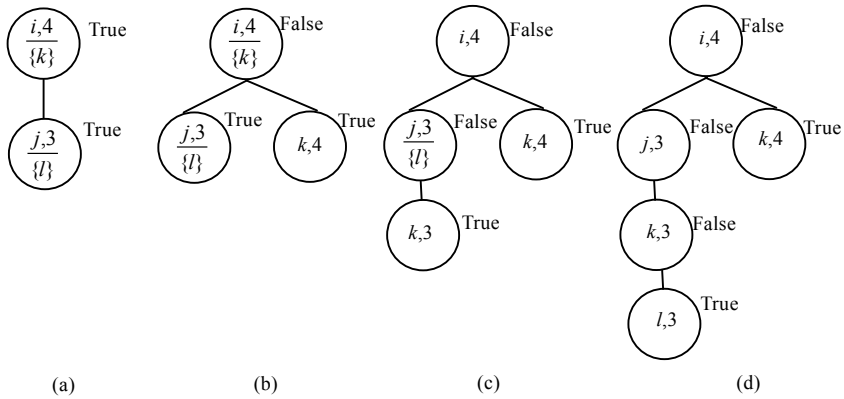

Fig.3 Decomposing compound nodes

图 3 分解复合结点的过程

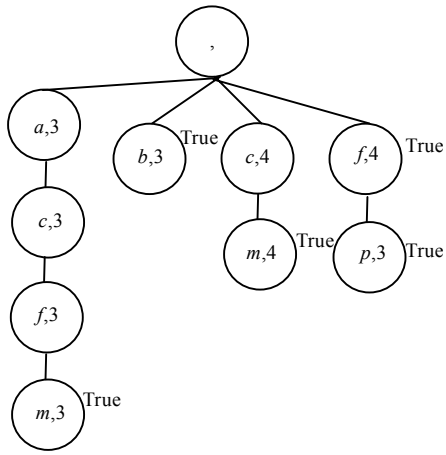

Fig.4 Decomposed CFIST

图 4 分解后的 CFIST

4 挖掘闭合模式的高性能算法

本文提出了一个基于完全集挖掘算法 OpportuneProject,采用适应性剪裁策略、挖掘频繁闭合模式的新算法 CROP(mining closed pattern by adaptive pruning based on opportuneproject).

4.1 混合搜索策略

深度优先搜索具有较高的投影效率,通过在内存中保留当前结点所在路径上所有结点的投影事务子集(模式支持子集),避免了重建投影事务子集的高 CPU 开销.CROP 以深度优先搜索为主.

对于无法直接在内存中保存的大型数据源,CROP 首先通过宽度优先搜索建造 CFIST 上半部.当所有 *k* 层结点投影事务子集的并集缩减到可用基于内存的结构表示时,*k* 层以下通过深度优先搜索建造.

4.2 模式支持子集的表示与投影

基于数组的稀疏型投影事务子集表示.如图 5 所示,TVLA(threaded varied length arrays)是一种适用于稀疏型 PTS 的基于数组的表示.事务用数组表示,第 1 个项目(自身不必表示)相同的事务由一个链队列 LQ 串在一起,并系于局部频繁项目列表 IL 中相同项目的条目.项目 *a* 的条目所链 LQ(*a*)已将所有支持结点 1(项目 *a*)的事务

串起来,而 LQ(*b*)穿线支持项目 *b* 的部分事务.通过逐步执行转移操作,可从父亲 TVLA 投影出子女 TVLA.首先,第 1 个 IL 条目所链 LQ 穿线了支持第 1 个子女的所有事务.将这些事务按其第 2 个项目分别转移到相应的 LQ 链后,第 2 个条目所链 LQ 穿线了支持第 2 个子女的所有事务,依此类推.子女 TVLA 可共享父亲 TVLA 的事务数组(非过滤型)或创建单独的事务数组(过滤型).

基于树的密集型投影事务子集表示.TTF(threaded transaction forest)是一种基于树的 PTS 表示,由项目列表 IL 和事务树林组成.每个结点用(*i*,*w*)标注,*i* 是标注项目,权重 *w* 是从根起始至该结点路径所表示事务的数目.IL 条目将标注同一项目的所有结点穿在一起.如图 6 所示,是图 2 根结点 PTS(原始数据源)的过滤型 TTF.路径(*a*,3)−(*c*,2)−(*f*,2)−(*m*,2)−(*p*,2)表示事务 01 和 05.IL 第 3 个条目的项目是 *c*,其支持数为 4,链指针(虚线箭头)将结点(*c*,2),(*c*,1)和(*c*,1)穿线在一起.当 TTF 压缩表示密集型 PTS 时,存储开销较 TVLA 小.通过自底向上或自顶向下的虚拟投影,可从父亲 TTF 投影出子女 TTF,子女 TTF(IL 和事务树林)不需要单独存储,共享父亲 TTF 同一内存空间.

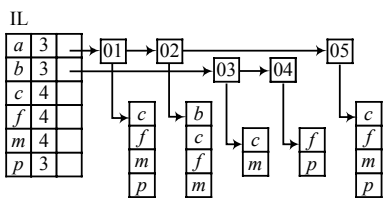

Fig.5 Array based PTS representation

图 5 基于数组的投影事务子集表示

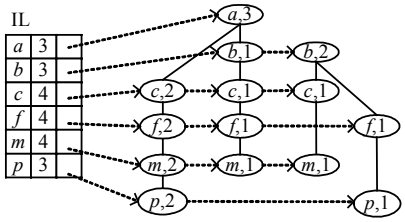

Fig.6 Tree based PTS representation

图 6 基于树的投影事务子集表示

投影事务子集表示形式与投影方法的选择.CROP 首先使用 TVLA 表示 PTS,因为 CFIST 高层结点或稀疏分枝的 PTS 中的事务往往是多样的、随机分布的,难以压缩表达,TVLA 存储开销相对较低.当 PTS 的密集程度达到给定值时,CROP 转而使用 TTF.在 CFIST 低层结点或较密集分枝中,PTS 内局部频繁项目较少并且相对支持率较高,用 TTF 压缩表达密集型 PTS 的效果非常好.在 PTS 收缩较快时,CROP 实际创建子女 PTS,进一步计数与投影操作的效率更高.在 PTS 收缩较慢时,CROP 采用虚拟投影或非过滤投影,既节省存储空间又避免递归建造 PTS 的高 CPU 开销,而且对投影与计数操作的效率影响不大.

4.3 复合型频繁模式树生成与剪裁代价的平衡

复合型频繁模式树的变型.定义 7 给出的 CFIST 称为标准型,路径结点自顶向下、子女结点自左向右的排列符合给定序 $\prec$,故也称为 TL 型(top down left to right).实际上,结点也可按逆 $\prec$ 序排列,得出 CFIST 的 3 种变型,见表 1.

Table 1 Variants of compound frequent item set tree

表 1 复合型频繁模式树变型

	Item ordering of nodes along any path ($\prec$)	Item ordering of children of any node ($\prec$)
TL	Top down	Left to right
TR	Top down	Right to left
BL	Bottom up	Left to right
BR	Bottom up	Right to left

定理 4. 按自左向右、自顶向下方式生成 CFIST,则:(1) 对于 TL 型或 BR 型 CFIST,若模式 P_1 先于 P_2 生成,则 $P_1 \not\subset P_2$;(2) 对于 TR 型或 BL 型 CFIST,若模式 P_1 先于 P_2 生成,则 $P_2 \not\subset P_1$.

复合型频繁模式树生成与剪裁效率平衡.TR 型或 BL 型 CFIST,可应用虚拟或非过滤型投影,生成效率较高,但发现被包容非闭合分枝较晚,剪裁效率较低.TL 型或 BR 型 CFIST 能及时发现被包容的非闭合分枝,剪裁效率高,但生成效率较低.CROP 采用混合型 CFIST:高层 TL 型或 BR 型,低层 TR 型或 BL 型.

4.4 算法描述

挖掘频繁闭合模式新算法 CROP 描述如下.主函数 CROP 首先创建复合型频繁模式树 CFIST 的根结点 *T*,

其标识项目及附加项目集均为空,即代表空模式,其投影事务子集 PTS 是原始数据集;然后调用 ClosedBF 按宽度优先搜索建立 CFIST 上半部;再调用 ClosedDF 按深度优先搜索建立 CFIST 下半部.

CROP(*O*,*I*,≺,min*sup*)

create CFIST root *T* and let *T.PTS*=***O*** and *T.IL*=***I***;

ClosedBF(*T*,0,≺,min*sup*);

ClosedGDF(*T*,≺,min*sup*);

函数 ClosedBF 首先对根结点投影事务子集 *T.PTS* 中的每个事务 *t* 作投影操作,若 *t* 可投影到当前层 *L*,则增加相应结点的局部项目的支持数,否则从 *T.PTS* 中删除 *t*(缩减);然后,对于 *L* 层的每个结点的支持数超过阈值的局部项目生成其对应子女,并完成相应的合并与剪裁操作;如果缩减后的 *T.PTS* 可用内存结构表示则返回,否则继续宽度优先搜索.

ClosedBF(*T*,*L*,≺,min*sup*)

for each $t=(t.tid,t.items)\in T.PTS$

for each *v* at level *L* reached by projecting *t*

for each $i\in v.IL\cap t.items$

$support(i)$++; $tidsum(i)$+= *t.tid*;

if (*t* cannot project to *L*) then remove *t* from *T.PTS*;

for each *v* at level *L*

for each $i\in v.IL$ with $(w=support(i))\geq min sup$

$MLPSC(v,i,w,tidsum(i),\varnothing,\{j\in v.IL|i\prec j\})$;

If ($NoMem(T.PTS)$) then $ClosedBF(T,L+1,\prec,min sup)$

函数 ClosedGDF 对于当前结点 *v* 的局部项目集中的每个项目 *i*,确定对应的模式支持集等;若支持数超过阈值,则生成 *v* 的子女 *c* 并完成必要的合并与剪裁操作,对于子女 *c* 递归执行此过程.

ClosedGDF(*v*,≺,min*sup*)

for each $i\in v.IL$

$O=g(\{i\})\cap v.PTS$; $ts=SumOfTids(O)$; $I=\{j\in v.IL|i\prec j\}$;

If $((w=\|O\|))\geq min sup$ &&$(c=MLPSC(v,i,w,ts,O,I))$ then $ClosedGDF(c,\prec,min sup)$;

函数 MLPSC 完成 CFIST 结点的生成、合并、局部剪裁和包容关系检查操作,包括结点 PTS 的表示形式选择和创建.

5 性能测评

性能测评的硬件平台是 Pentium IV 800MHz CPU,512MB Memory 和 20GB HD 的 PC 机,操作系统是 Windows 2000 Server.对比算法:最新版 CHARM[5]采用 tidsets,diffsets(2、3 阶段起)选项,记作 charm-t,charm-d2 和 charm-d3,CLOSET[3]是优化版,MAFIA[4]采用闭合模式选项,所有执行代码由原作者提供.

5.1 数据集的特性

Table 2 Characteristics of datasets

表 2 数据集的特性

Data set	# Items	Avg. length	# Trans.	Max. pattern length (support)
T25I20D100k	20K	28.4	100K	33 (0.10%)
CONNECT	150	43.0	67 557	29 (10.0%)
BMS-POS	1 657	6.5	515 597	14 (0.01%)
Gazelle	497	2.5	59 602	154 (0.01%)

数据集基本特性见表 2,不同支持率下频繁模式和闭合模式数如图 7 所示,纵轴为模式数(千个),横轴为最小支持率(百分比).T25I20D100k 是介于稀疏型与密集型之间的人工数据集,用 IBM Almaden 生成器[7]生成.CONNECT[8]是非常密集的数据集.BMS-POS 和 Gazelle 是 Blue Martini 公司提供的实际数据集.

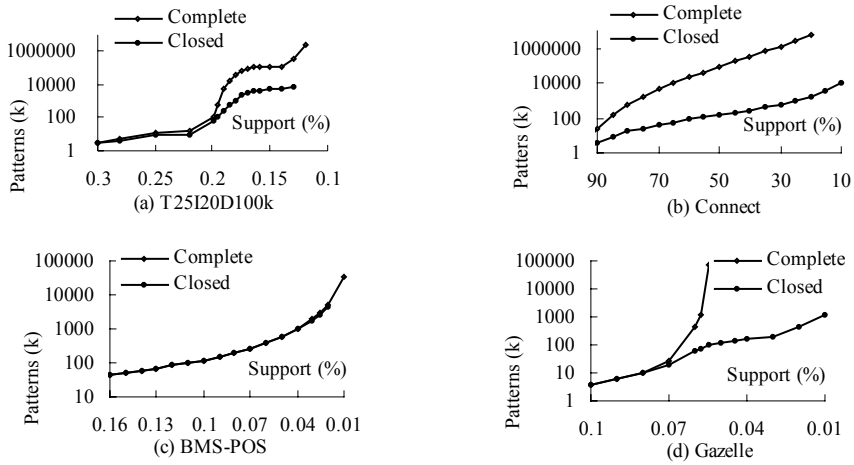

Fig.7 Complete and closed frequent item sets with different supports
图 7 不同支持率下的完全与闭合频繁模式

5.2 实验结果

性能对比分析指标是各算法在不同数据集、不同最小支持率下的运行时间和存储开销.如图 8 所示为运行时间指标,纵轴(时间)采用对数坐标,横轴(最小支持率)采用百分比.

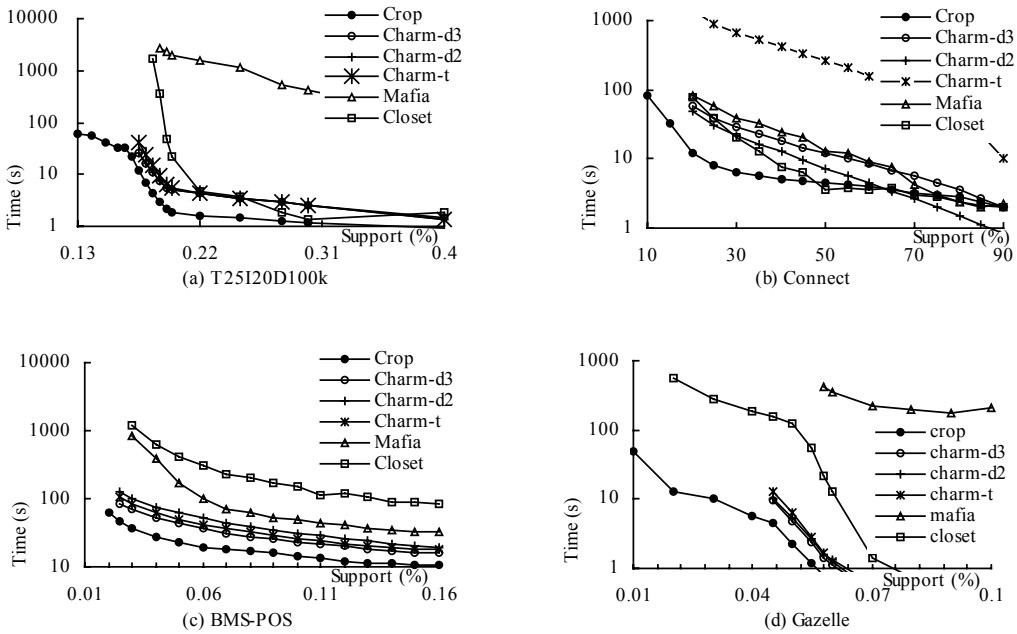

Fig.8 Runtimes of different algorithms on four datasets
图 8 不同算法挖掘 4 个数据集的运行时间

CROP 时间效率是 CHARM 的 2~5 倍.挖掘 Gazelle,最小支持率低于 0.06%,CROP 效率是 charm-d2 的 2~3 倍;挖掘 CONNECT,低于 50%,是 charm-d2 的 2~5 倍;挖掘 BMS-POS,低于 0.16%,是 charm-d2 的 2~3 倍;挖掘 T25I20D100k,低于 0.4%,是 charm-d2 的 2~3 倍.charm-d3 与 charm-d2 挖掘 T25I20D100k 效率相同,挖掘 Gazelle, BOS-POS,前者略高,挖掘 CONNECT 后者高得多.charm-t 效率总比 charm-d2 和 charm-d3 要低.CROP 效率比 charm-t 高 1~2 个数量级.

CROP 时间效率比 CLOSET 高 1 个数量级.挖掘 Gazelle,最小支持率低于 0.06%,CROP 效率比 CLOSET 高 1 个数量级;挖掘 CONNECT,低于 40%,CROP 效率比 CLOSET 高 2~7 倍;挖掘 BMS-POS,低于 0.16%,CROP 效率比 CLOSET 高 1 个数量级;挖掘 T25I20D100k,低于 0.2%,CROP 效率比 CLOSET 高 1 个数量级.

CROP 时间效率比 MAFIA 高 2~3 个数量级.几乎在所有数据集所有最小支持率水平都如此.

CROP 存储开销最低,可在更低的最小支持率下进行挖掘.挖掘 Gazelle,最小支持率低于 0.045%,CHARM 和 MAFIA 因耗尽物理内存而无法运行,低于 0.02%,CLOSET 也无法运行,而 CROP 可在 0.01%以下运行;挖掘 CONNECT,低于 20%时,其他 3 种算法无法运行,而 CROP 可在 10%以下运行;挖掘 BMS-POS,低于 0.025%,其他 3 种算法无法运行,而 CROP 可在 0.02%以下运行;挖掘 T25I20D100k,低于 0.175%时,其他 3 种算法无法运行,而 CROP 可在 0.13%以下运行.

6 结束语

本文在我们提出的挖掘频繁模式完全集算法 OpportuneProject[6]基础上,提出了挖掘频繁闭合模式的高性能算法 CROP.CROP 采用复合型频繁模式树 CFIST 组织频繁模式集,存储开销小,易于实现高效的局部和全局剪裁.CROP 以深度优先搜索为主、辅以宽度优先搜索,混合使用基于数组和基于树的模式支持集表示法,投影效率高,并且较好地平衡了树生成与剪裁代价.实验表明,CROP 的效率与可伸缩性是最佳的.

References:

[1] Pasquier N, Bastide Y, Taouil R, Lakhal L. Discovering frequent closed itemsets for association rules. In: Beeri C, *et al*, eds. Proc. of the 7th Int'l. Conf. on Database Theory. Jerusalem: Springer-Verlag, 1999. 398~416.

[2] Agrawal R, Srikant R. Fast algorithms for mining association rules. In: Beeri C, *et al*, eds. Proc. of the 20th Int'l. Conf. on Very Large Databases. Santiago: Morgan Kaufmann Publishers, 1994. 487~499.

[3] Pei J, Han J, Mao R. CLOSET: An efficient algorithm for mining frequent closed itemsets. In: Gunopulos D, *et al*, eds. Proc. of the 2000 ACM SIGMOD Int'l. Workshop on Data Mining and Knowledge Discovery. Dallas: ACM Press, 2000. 21~30.

[4] Burdick D, Calimlim M, Gehrke J. MAFIA: A maximal frequent itemset algorithm for transactional databases. In: Georgakopoulos D, *et al*, eds. Proc. of the 17th Int'l. Conf. on Data Engineering. Heidelberg: IEEE Press, 2001. 443~452.

[5] Zaki MJ, Hsiao CJ. CHARM: An efficient algorithm for closed itemset mining. In: Grossman R, *et al*, eds. Proc. of the 2nd SIAM Int'l. Conf. on Data Mining. Arlington: SIAM, 2002. 12~28.

[6] Liu JQ, Pan YH, Wang K, Han J. Mining frequent item sets by opportunistic projection. In: Hand D, *et al*, eds. Proc. of the 8th ACM SIGKDD Int'l. Conf. on Knowledge Discovery and Data Mining. Alberta: ACM Press, 2002. 229~238.

[7] Srikant R. Quest synthetic data generation code. San Jose: IBM Almaden Research Center, 1994. http://www.almaden.ibm.com/software/quest/Resources/index.shtml

[8] Blake C, Merz C. UCI Repository of machine learning. Irvine: University of California, Department of Information and Computer Science, 1998. http://www.ics.uci.edu/~mlearn/MLRepository.html