第33卷 第19期 计 算 机 工 程 2007年10月
Vol.33 No.19 Computer Engineering October 2007
·软件技术与数据库· 文章编号：1000—3428(2007)19—0066—03 文献标识码：A 中图分类号：TP301.6

基于FP-tree的多层关联规则快速挖掘算法

曹洪其[1]，姜志峰[2]，孙志挥[2]
（1. 南通职业大学电子工程系，南通 226007；2. 东南大学计算机科学与工程系，南京 210096）

摘 要：研究了多层关联规则挖掘的理论和方法，提出了一种基于FP-tree的快速挖掘算法FAMML_FPT。该算法不仅实现了同层次关联规则的挖掘，也能实现跨层次关联规则的挖掘，其中引入了修补项、跨层修补项的概念，以便从低到高逐层建立FP-tree，有效减少了扫描数据库的次数，且不用产生大量的候选项集，提高了数据挖掘的效率。

关键词：数据挖掘；多层关联规则；频繁模式树；修补项

Fast Mining Algorithm for Multi-level Association Rules Based on FP-tree

CAO Hong-qi[1], JIANG Zhi-feng[2], SUN Zhi-hui[2]
（1. Department of Electronic Engineering, Nantong Vocational College, Nantong 226007;
2. Department of Computer Science and Engineering, Southeast University, Nanjing 210096）

【Abstract】 This paper focuses on the research of multi-level association rules mining, and presents a fast algorithm FAMML_FPT on frequent pattern tree. This algorithm can realize data mining not only among an identical level association rule but also among cross-level association rules. In this algorithm, conceptions of the repaired items and the cross-level repaired items are brought, which is propitious to create FP-tree from lower levels to higher levels. This algorithm can reduce the scanning times of the database and eliminate the need for generating the candidate items, which improves efficiency of data mining.

【Key words】 data mining; multi-level association rules; frequent pattern tree(FP-tree); repaired items

对于事务或关系数据库来说，一些项或属性所隐含的概念是有层次的。在许多实际应用中，由于多维数据空间的稀疏性，在低层或原始层的数据层次上很难找出强关联规则。概念分层[1]的引入使得人们能够在较高的概念层次上进行数据挖掘，发现新颖的、有价值的强关联规则。在某些情况下，有些项在同层次之间不存在关联关系，但与不同层次的项之间存在着关联关系。因此，多层关联规则挖掘的研究，不仅包括同层次关联规则挖掘的问题，有时也涉及跨层次关联规则的挖掘。

在用于多层关联规则挖掘的经典算法中，以Cumulate[2]和ML-T2L1[3]最为著名。Cumulate算法能进行多层及跨层次频繁模式的挖掘，但该算法仅是将源数据放在同一层次级别上考虑的普遍化关联规则挖掘算法；ML-T2L1算法采用自顶向下方式进行逐层挖掘，但不支持跨层次的挖掘。这2种算法都基于Apriori[4]算法思想，因此存在着与Apriori算法相同的缺陷。针对Apriori算法的不足，文献[5]提出了一种不产生候选项集的算法FP-Growth，避免了高代价的候选项集的产生，扫描次数少、效率更高。还有一些以FP-Growth算法为基础的多层次关联规则挖掘算法，这些算法不同程度地提高了挖掘效率。但其中多数算法只能实现层内的挖掘，无法发现不同概念层上数据项之间的关联性；还有一些算法虽能实现跨层次挖掘，但不能很好地体现概念层次内在的制约关系，造成FP表头关系复杂，无法有效地简化挖掘过程。

1 相关概念与定义

多层关联规则的挖掘是基于概念分层进行关联规则的挖掘。概念层次结构通常使用概念树表示，树的结点表示概念，树枝表示偏树，树中的每一层都使用一个数字来表示层数，根节点的层数为0，根节点以下节点的层数定义为其父节点层数加1。很明显，较高层的层数较低，较低层的层数较高。本文中的概念树最底层层数设为 m。

定义1 j 层 k-项集 $A[j, k]$在事务数据库DB中的支持数称为 $A[j, k]$的支持数，用 $A[j, k].Sup$ 表示。设 $minsup.j$ 为 j 层的最小支持度，$|D|$为DB中事务数的个数。若 $A[j, k].Sup \geq minsup.j \times |D|$，则 $A[j, k]$是 j 层的频繁 k-项集，记为 $L[j, k]$。

定义2 在基于FP-tree的多层关联规则挖掘环境中，将DB对应的 j 层频繁模式树称为 j 层频繁模式树，记为 $Fptree[j]$。

定义3 $\lambda(p)$是一个函数，其功能为返回项 p 上一层的父节点项 q。如果 j 层的项 p 在DB中为非 j 层频繁项，但 q（即 $\lambda(p)$）在DB中为 j-1层频繁项，则 p 称为项 q 的修补项，记为 $Lre[j, 1]$，p 编码中 j-1层后编码变为#。

由概念分层和定义3可得到：j-1层项 q 的支持数为 j 层中所有父节点为 q 的 j 层频繁项的支持数与 q 的所有修补项的支持数之和。

定义4 频繁项集中所含的项目至少存在2个项目所在的层次不同，且它们在概念树中不属于同一颗子树，则称这

基金项目：国家自然科学基金资助项目(70371015)
作者简介：曹洪其(1957－)，男，副教授，主研方向：数据库，数据挖掘；姜志峰，硕士研究生；孙志挥，教授、博士生导师
收稿日期：2007-02-06 **E-mail：**chq@mail.ntvc.edu.cn

样的频繁项集为跨层次频繁项集。

定义 5 若项 $A[j, 1]$不是 j 层频繁项，但 $A[j, 1]$在跨层次挖掘时要跨到 i 层$(j<i)$，在 i 层是频繁项，则称 $A[j, 1]$是跨 i 层频繁项，记为 $Lk[j,1, i]$，项 $A[j,1]$编码中 j 层后编码变为@。

定义 6 $\lambda(r)$是一个函数，其功能为返回项 r 上一层的父节点项 s。如果 j 层的项 r 在 DB 中不为 j 层频繁项和其他层的跨层次频繁项，s(即$\lambda(r)$)在 DB 中也不为 j-1 层频繁项，但 s 在 DB 中是跨 i 层频繁项，则称 r 为项 s 的跨 i 层修补项，记为 $Lkre[j, 1]$。r 编码中，j-1 层后的编码变为@，i 层后编码变为$。

同样，由概念分层和定义 6 可得到：在进行跨层次挖掘时，j-1 层项 s 的支持数为 j 层中所有父节点为 s 的跨层次频繁项的支持数与 s 的所有跨层次修补项的支持数之和。

挖掘跨层次频繁项集最简单的方式是把所有不同层项目同等对待，即不考虑项目的层次关系进行挖掘。但按照这种方式挖掘速度太慢，并且会有许多无用规则产生。产生跨层次规则冗余的主要原因是：层中的项 A 是频繁项或跨层频繁项，可得出项 A 与其他相关项的关联关系，若再跨层次挖掘得出项 A 的父项和子项与这些项的关联关系，则所包含的新信息成分不多；而如果项 A 是非频繁的，那么研究它的跨层次挖掘是有价值的。所以，本文并不是挖掘所有层间关联规则，在生成跨层次的候选项集时，去除具有子孙关系的项集，缩减了候选项目集。

2 FAMML_FPT 算法

下面结合表 1 所示的编码事务表介绍 FAMML_FPT 算法。该编码事务表中的编码表示的概念树共 3 层，在较低层使用递减的最小支持度（支持数），1 层的最小支持数 $SP1=5$，2 层的最小支持数 $SP2=3$，3 层的最小支持数 $SP3=2$。

表 1 编码事务表

TID	项目编码集
$T1$	{111, 211, 221, 312}
$T2$	{111, 211, 222, 323 }
$T3$	{112, 121, 221, 411}
$T4$	{111, 721}
$T5$	{111, 122, 211, 221, 413}
$T6$	{211, 323, 524}

2.1 算法思路

如何快速找出频繁项目集是挖掘关联规则算法的关键步骤和研究重点。因此，FAMML_FPT 算法的关键在于建立起各概念层的条件模式树 Fptree，形成各层条件模式基，从而获得各层中的频繁项目集和跨层次频繁项目集。FAMML_FPT 算法采用层交叉单项过滤搜索策略，即一个第 i 层上的项被考察，当且仅当它在第 i-1 层上的父节点是频繁的。该算法主要有以下步骤：

(1)扫描事务数据库 DB，统计出 1 层 1-项集 $A[1, 1]$的支持数，产生 1 层频繁 1-项集 $L[1, 1]$及跨层次频繁项 $Lk[1, 1, i]$。例如：表 1 中 {1**} ，{2**} 的支持数分别为 7 和 8，是频繁项；而 {3**} ，{4**} ，{5**} ，{7**} 的支持数分别为 3, 2, 1, 1，是非频繁项，但 {3**} ，{4**} 分别是跨第 2 层和第 3 层的跨层次频繁项。因此，经本步骤可得 $L[1, 1]$为 {1**} 7, {2**} 8，同时也得到跨层次频繁项 $Lk[1,1,2]$和 $Lk[1, 1, 3]$分别为 {3**} 3 和 {4**} 2。

(2)第 2 遍扫描数据库 DB，根据 $L[1,1]$和 $Lk[1,1,i]$对 DB 进行层间裁减，统计出最底层即 m 层 1-项集 $A[m,1]$的支持数。在此基础上，按层统计各层 1-项集的支持数，产生各层的频繁 1-项集及修补项、各层的跨层次频繁 1-项集及跨层次修补项。例如：由表 1 及相应的 $L[1, 1]$经本步骤产生的各层频繁 1-项集及修补项如图 1 所示。

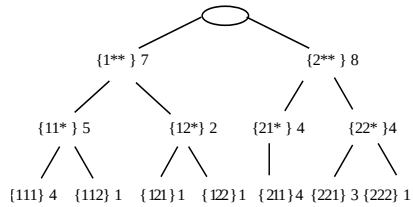

图 1 产生各层频繁 1-项集及修补项

在图 1 中，2 层 {11*} 支持数为 5，是频繁项，3 层中 {111} 支持数为 4，是频繁项，而 {112} 支持数为 1，是非频繁项。由于 {11*} 是频繁项，因此 {112} 为 {11*} 的修补项，该修补项用编码 {11#} 表示（11#由 112 在第 3 位变为#而得到），{112} 的支持数记入修补项的支持数。同理，{22*} ，{221} 为频繁项，{222} 是非频繁项，产生修补项为 {22#}；{1**} 为频繁项，{12*} ，{121} ，{122} 都为非频繁项，产生修补项为 {1##}。

由表 1 产生的跨层次信息如表 2 所示。为了缩减候选项目集，根据关联规则的冗余规则，表 2 中去除了同一层中具有子孙关系的冗余跨层次频繁项。表中 {3**}的支持数是 3，是跨第 2 层的频繁项，用{3@@}表示；{4**}的支持数是 2，是跨第 3 层的频繁项，用{4@@}表示；{32*}的支数为 2，是跨第 3 层的频繁项，用{32@}表示；{31*}的支持数为 1，是不跨层的频繁项，但由于{3**}是跨第 2 层的频繁项，则{31*}为{3**}跨第 2 层频繁项的修补项，用{3@$}表示。

表 2 跨层次信息

层数	跨层次的项
2	{3**}
3	{32*}, {4**}

(3)建立各层 Fptree。1)建立头表。依次将各层频繁项、修补项及跨层次频繁项、跨层次修补项按降序排列，由这些项作为相应层 FP-tree 的项集。2)第 3 次扫描数据库，进行层间裁减，减去不属于 $L[1, 1]$及 $Lk[1, 1, i]$的项，建立 m 层 FP-tree 树。3)遍历 FP-tree 树，一方面用不包含修补项、跨层次修补项的路径产生仅包含频繁项和跨层次修补项的该层的 FP-tree’树，从而得到该层的频繁项目集及跨层次频繁项目集；另一方面用路径产生上一层的 FP-tree 树。

例如：图 2~图 5 表示了由表 1、表 2 和图 1 产生的第 3 层 FP-tree 树，及通过遍历该树形成该层的跨层次 FP-tree’树以及上一层（第 2 层）FP-tree 树。

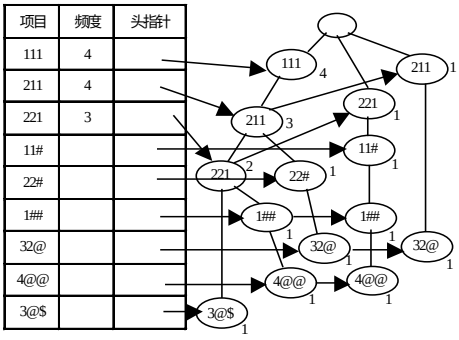

图 2 第 3 层 FP-tree 树

{111, 211, 221, 3@$}1 =>{111, 211, 221}1	{111, 211, 221, 3@$}1 =>{11*, 21*, 22*, 3@@}1
{111,211, 221,1##, 4@@}1 =>{111, 211, 221,4**}1	{111, 211, 221,1##,4@@}1 =>{11* , 21* , 22* , 1##}1
{111, 211, 22#, 32@}1 =>{111, 211, 32*}1	{111, 211, 22#, 32@}1 =>{11*, 21*, 22*, 3@@}1
{111}1 => {111}1	{111}1 =>{11*}1
{221 , 11# , 1## , 4@@}1 => {221,4**}1	{221 , 11# , 1## , 4@@}1 =>{22* , 11* , 1## }1
{211 , 32@} 1=>{211 , 32*}1	{211 ,32@}1 =>{21* ,3@@}1

(a) 形成第 3 层 FP-tree'树的路径　　(b) 形成第 2 层 FP-tree 树的路径

图 3　生成的路径

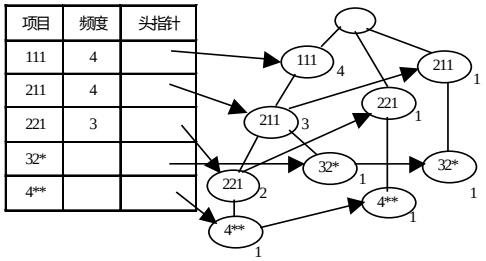

图 4　第 3 层 FP-tree'树

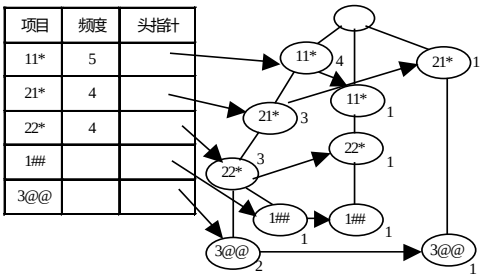

图 5　第 2 层 FP-tree 树

2.2　算法

输入　数据库 DB；各层的最小支持度 *minsup.j*(*j*=1, 2,…,*m*)。

输出　各层的频繁项集

方法：

(1)L[1, 1]={1 层频繁 1-项集}；

Lk[1, 1, i]={ 所有 A[1, 1]跨层次频繁 1-项集}；

//第 1 遍扫描数据库，产生 *L*[1,1]和 *Lk*[1,1,*i*]

(2)For each Transaction t in DB do

//第 2 遍扫描数据库

{ t'=Filter Transaction(t)；

//根据 *L*[1,1]和 *Lk*[1,1,*i*]，进行层间裁剪

For each item in t' do

TransItem [item].sup++；

//统计 *m* 层 1-项集各自的支持数}

(3)Get _each_level_items(TransItem，minsup)；

//处理 *m* 层 1-项集，统计各层 1-项集的支持数，产生各层的频
//繁 1-项集及修补项、各层的跨层次频繁 1-项集及跨层次修补项（除
//1 层）

(4)For(j=m；j≥1；j--)

sort L[j, 1]、Lre[j, 1]、 Lk[k, 1, j] and Lkre[j, 1] in descending order；//建立头表

(5)For each Transaction t in DB do

//第 3 遍扫描数据库

{t'=Filter Transaction(t)；

inset the t' to the Fptree[m]；

//产生 *m* 层 FP-tree　　}

(6)For(j=m；j＞1；j--)

{Fptree'[j]=Get_FP-tree1(Fptree[j])；

//产生该层的 FP-tree'树

Get _ Frequent_Itemsets (Fptree'[j])；

//建立条件模式基，得出频繁项集

Fptree[j-1]=Get_FP-tree2(Fptree[j])；

//产生上一层的 FP-tree 树　　}

Fptree'[1]=Get_FP-tree1(Fptree[1])；

//产生 1 层的 FP-tree'树

Get _ Frequent_Itemsets (Fptree'[1])；

3　算法及实验结果分析

3.1　算法分析

FAMML_FPT 算法是以 FP-Growth 算法为基础的多层关联规则挖掘算法，通过引入修补项、跨层修补项的概念改进原算法的性能。该算法只须扫描数据库 3 遍，便能产生各层的频繁 1-项集及修补项、各层的跨层次频繁 1-项集及跨层次修补项，并建立 *m* 层（最低层）FP-tree 树。然后从 *m* 层 FP-tree 树起，逐层遍历 FP-tree 树，用路径产生上一层 FP-tree 树，从低到高建立起每一层的 FP-tree 树，实现同层次关联规则的挖掘和跨层次关联规则的挖掘。所以，FAMML_FPT 算法用于任何层次数的多层关联规则挖掘时，最多只须扫描数据库 3 遍。而对于基于 Apriori 的多层关联规则挖掘算法，在挖掘每一层的频繁项集和跨层次频繁项集时，需要扫描数据库多遍。因此，FAMML_FPT 算法与基于 Apriori 的多层关联规则挖掘算法相比，有效地减少了扫描数据库的次数，且不用产生大量的候选项集，提高了数据挖掘的效率。

3.2　实验结果分析

实验的硬件运行环境：CPU 为 Pentium Ⅲ、内存为 256MB、硬盘为 60GB 的计算机；软件运行环境：Windows2000 Server 操作系统，VC++6.0；并采用文献[4]的合成数据集：T5I2D100K，*N*=100，|*L*|=200，*maxlevel*=3 作为测试数据，实验结果如图 6 所示。

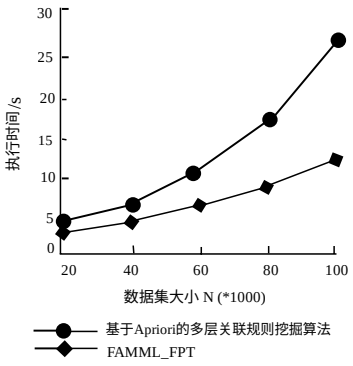

图 6　算法执行时间与数据集大小的关系

图 6 表示了各层在特定的最小支持度阈值(最高层到最低层分别为 20%, 12%, 8%)下，从合成数据集中抽取不同大小的数据集时，FAMML_FPT 算法与基于 Apriori 的多层关联规则挖掘算法执行时间的比较关系。实验结果说明 FAMML_FPT 算法的执行效率优于后者，并且当数据集增大时，FAMML_FPT 算法性能更好。这是因为当数据集增大时，虽然 2 种算法的执行时间都会增加，但由于基于 Apriori 的多层关联规则挖掘算法会产生相对较多的候选项集，而

（下转第 71 页）