一种基于前缀树的频繁模式挖掘算法*)

朱光喜　吴伟民　阮幼林　刘　干
(华中科技大学计算机科学与技术学院　武汉430074)

摘　要　挖掘频繁模式是许多数据挖掘任务的关键步骤。基于FP-Tree的挖掘算法由于无须生成候选项集效率明显高于Apriori类算法,但FP-Tree结构存在动态维护复杂、而且在挖掘过程中需要递归地创建大量的条件FP-Tree,时空效率不高。因此,本文提出一种基于前缀树的新算法。该算法通过引入一种新结构—前缀树(Prefix Tree)用来压缩存放数据所相关信息,并通过调整前缀树中节点信息和节点链直接在Prefix Tree上采用深度优先的策略挖掘频繁模式,而不需要任何附加的数据结构,从而大大提高了挖掘效率。

关键词　频繁模式,频繁项集,FP-Tree,前缀树

A Mining Algorithm for Frequent Patterns Based on Prefix Tree

ZHU Guang-Xi　Wu Wei-Min　RUAN You-lin　LIU Gan
(College of Computer Science and Technology, Huazhong University of Science and Technolgoy, Wuhan 430074)

Abstract　Mining frequent patterns is a key problem in data mining research. Although mining based on FP-Tree achieves better performance and efficiency than Apriori-like algorithms because of avoiding costly candidate generation, it still suffers from creating conditional FP-Tree separately and recursively during the mining process. In this paper, we propose a new method PTM that designs a new structure called Prefix Tree, which stores all of the information in a highly compact form. PTM mines frequent patterns in depth-first order and directly in Prefix Tree by adjusting node information and node links without using any additional data structures. Thus, it can improve performance greatly.

Keywords　Frequent pattern, Frequent itemsets, FP-tree, Prefix tree

1　引言

频繁模式的挖掘是关联规则、相关分析、序列分析等许多重要数据挖掘任务的关键步骤。近年来,人们对频繁模式的挖掘算法进行了大量的、深入的研究工作。在众多算法中,以Agrawal等人提出的Apriori算法[1]最为著名。长期以来,频繁模式的挖掘主要采用Apriori算法及其改进形式。该类算法采用候选项集生成-筛选方法,必须耗费大量处理规模巨大的候选项集,同时必须多次扫描数据库,对候选项集进行筛选。为提高Apriori算法的有效性,人们已经提出了许多Apriori算法的变形[2,3],融合了许多技术,如散列项集计数、事务压缩、划分、选样和动态项集技术等。众多Apriori类算法虽然大幅度压缩了候选集的大小,但仍然需要产生大量候选集,并可能需要重复扫描数据库。因此,针对Apriori类算法存在的问题,Han等人提出了FP-Tree和相应的FP-Growth[4]算法。FP-Growth算法是一种基于模式增长的挖掘算法,由于FP-Tree蕴涵了所有的频繁项集,其后的频繁项集的挖掘只需要在FP-Tree上进行。该算法本质上不同于Apriori算法的候选生成-筛选方法,无须生成候选项集,将挖掘长频繁项集的问题转换成递归挖掘一些短频繁项集,然后连接后缀。它使用最不频繁的项作后缀,提供了好的选择性,而且只需要两次扫描数据库。因此,该方法显著地缩小了搜索空间,有效地避免了组合爆炸,挖掘效率明显提高。对FP-Tree方法的性能研究表明,对于挖掘长的和短的频繁模式,它都是有效的和可伸缩的,并且大约比Apriori算法快一个数量级。然而,FP-Tree结构存在动态维护复杂,而且在挖掘过程中需要递归地创建大量的条件FP-Tree。特别是在支持度阈值较小或存在长模式时,即使对于不太大的数据,也会产生数以万计的条件FP-Tree。动态地创建数以万计的条件FP-Tree,将消耗大量的时间和空间。因而FP-Growth算法的时空效率不高。

为此,本文提出一种新结构—前缀树(Prefix Tree)和直接在Prefix Tree上挖掘频繁模式的新算法PTM。前缀树用来压缩存放数据的相关信息,PTM算法通过调整前缀树中相关节点信息和节点链直接在Prefix Tree上采用深度优先的策略挖掘频繁模式,而不需要任何附加的数据结构,极大提高了频繁模式的挖掘效率。

2　相关概念

2.1　频繁模式

设$I=\{i_1,i_2,\cdots,i_m\}$是m个不同项目的集合。给定事务数据库DB,对于项目集$X\subseteq I$,X在DB中的支持数是指D中包含X的事务数,记为$X.count$。X在DB中的支持度是指DB

*)基金项目:国家自然科学基金重大项目(603905405);国家自然科学基金项目(60273075)资助项目;国家863计划课题(2001AA123014)。朱光喜　教授,博士生导师,研究兴趣为宽带无线通信与多媒体系统,数据挖掘,并行与分布式计算。吴伟民　讲师,博士研究生,研究兴趣为宽带无线通信与多媒体系统,数据挖掘,并行与分布式计算。阮幼林　博士研究生,研究兴趣为数据挖掘,并行与分布式计算,宽带无线通信与多媒体系统。刘　干　博士研究生,讲师,研究兴趣为宽带无线通信与多媒体系统,数据挖掘,并行与分布式计算。

中包含 X 事务的百分比，记为 $X.sup$。如果 X 的支持度不小于用户给定的最小支持度阈值 *minsup*，则称 X 为 DB 中的频繁项目集。如果 X 包含 k 个项目，那么又称为频繁 k-项目集。项目集中项目的个数称为项目集的维数或长度，频繁1-项目集简称频繁项目。对于给定的支持度阈值 *minsup*，挖掘频繁模式就是找出事务数据库 DB 中的全部频繁项集。

2.2 前缀树 Prefix Tree

前缀树 Prefix-Tree 与频繁模式树 FP-Tree 非常类似，记录的都是 DB 中的每个事务 Trans 的频繁项。频繁模式树 FP-Tree 中事务中的项按支持度计数降序排列，而前缀树 Prefix-Tree 中出现的项组成一个偏序集($\leqslant$)，事务中的项严格按字典顺序排列。因此，两者只是记录项的顺序不同。在 Prefix-Tree 中，每个节点有4个域组成：节点名称 item-name、节点计数 count、节点链 node-link 和父节点指针 parent。图1所示的是有4个频繁项的完全前缀树，表示4个频繁项所构成的所有可能模式。从根节点 *root* 到其它节点表示每种模式，如1234表示长度为4的模式。每个子树表示以该子树的根为前缀的所有模式，如1-prefix 表示包含1的所有模式，2-prefix 表示包含2但不包含1的所有模式，依次类推。下面，i_j 和 i_k 分别用来表示单个前缀项，而 α，β 和 γ 表示通用前缀项。

性质1 事务数据库 DB 的每次事务记录在前缀树 Prefix-Tree 的某条路径中。

性质2 前缀树 Prefix-Tree 中每条路径的计数表示事务数据库 DB 中相同事务的个数。

性质3 前缀树 Prefix-Tree 的深度为相应的事务数据库 DB 中事务的最大长度。

性质4 给定一个满足偏序关系($\leqslant$)的项集 $I=\{i_1,i_2,\cdots,i_m\}$，如果 $\alpha.i_j$-prefix 包含的模式是频繁的，而且满足 $i_j\leqslant i_k$，则 $\alpha.i_k$-prefix 包含的模式才可能是频繁的。如果 $\alpha.i_j$-prefix 包含的模式不频繁，则 $\alpha.i_j.i_k$-prefix 包含的模式一定不频繁。

性质4提供了一个很好的挖掘策略：如果该节点的计数小于支持度阈值 *minsup*，即该项不频繁，则以该节点为根的整个子树不需考虑。因此，利用该性质可以显著地缩小搜索空间，有效地避免了组合爆炸，极大提高挖掘效率。

定义1 当满足 $i_j\leqslant i_k$ 时，如果 $i_j.\alpha$-prefix 包含在 $i_k.\alpha$-prefix 中，就记作 $i_j.\alpha\text{-prefix}\subseteq i_k.\alpha\text{-prefix}$。

定义2 α-prefix 的覆盖是指包含 α-prefix 的所有 β-prefix，即 $\alpha\text{-prefix}\subseteq\beta\text{-prefix}$。

如 $b.c.d$-prefix 的覆盖包括 $b.c.d$-prefix 和 $a.b.c.d$-prefix；$c.d$-prefix 的覆盖包括 $c.d$-prefix，$b.c.d$-prefix，$a.c.d$-prefix 和 $a.b.c.d$-prefix。

性质5 挖掘 α-prefix 包含的频繁模式就转化为挖掘 α-prefix 的覆盖包含的频繁模式。

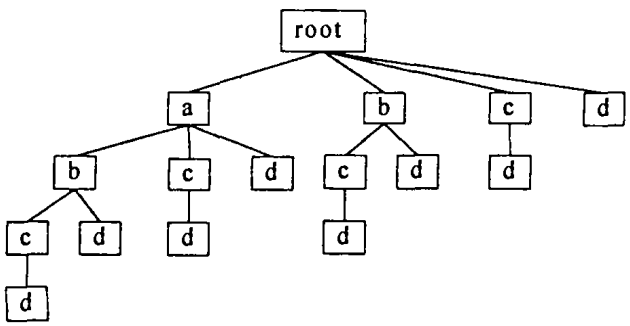

图1 项 a，b，c，d 的完全前缀树 Prefix-Tree

因此，要想挖掘 α-prefix 包含的频繁模式，只需得到 α-prefix 的覆盖即可。如要确定 $c.d$-prefix 包含的模式是否频繁只需对以上四个子树进行挖掘即可。由于相同模式可能分布在不同子树中，当分别对每个子树进行挖掘时，无法判断其是否频繁，必须递归结构造其条件模式树才能判断，就如同 FP-Tree 和 FP-Growth 算法。但如果把这四个子树合并成一个树(α-prefix 的覆盖)，就不需要构造条件模式树，基于性质4采用深度优先的策略即可判断该模式是否频繁。

3 频繁模式挖掘算法

通常基于 FP-Tree 的挖掘算法的主要步骤为：(1)扫描 DB 一遍得到各项目的频度，根据最小支持度 *minsup* 得到频繁项集；对频繁项目按其频率由大到小排列成表 L，形成头表；(2)再次扫描 DB，对每一条交易中的所有频繁项目，按表 L 中的次序插入到 FP-Tree 中；(3)调用 FP_Growth 算法对 FP-Tree 进行挖掘。在 FP-Tree 中，相同模式可能存在不同的子树中；当对某个子树进行挖掘时，无法判断该模式是否频繁，必须对出现该模式的所有子树进行挖掘，构造其条件模式树。由于本算法采用前缀树 Prefix-Tree 存放事务，并把不同子树中的相同模式并入一个子树中，故而可以直接判断该模式是否频繁而无须构造其条件模式树。因此，本算法在性质4和性质5的基础上由顶向下采用深度优先搜索进行挖掘。因此，挖掘处 PTM 也分为以下三个步骤：

1. 扫描 DB 一遍得到各项目的频度，根据最小支持度 *minsup* 得到频繁项集；对频繁项目按其字典顺序排列成表 L，形成头表；

2. 再次扫描 DB，对每一条交易中的所有频繁项目，按表 L 中的次序插入到前缀树 Prefix-Tree 中；

3. 基于前缀树 Prefix-Tree 采用深度优先搜索进行挖掘；

3.1 前缀树 Prefix-Tree 的构造算法

输入：事务数据库 DB

输出：前缀树 Prefix-Tree 和项集合 F 及其支持数

前缀树 Prefix-Tree 的构造算法如下：

(1)创建前缀树 Prefix-Tree 的根节点，以"*root*"标记它。扫描事务数据库 DB 一次，构造前缀树 Prefix-Tree。

(2)对于 DB 中的每个事务 Trans 作如下处理：

①事务 Trans 中的频繁项按字典顺序排列。设排序后的项列表为 $[p|P]$，其中 p 是第1个项目，而 P 是剩余项目的列表；

②调用 *insert_tree*($[p|P]$,T)：如果 T 有子女 N 使得 $N.node\text{-}name=p.item\text{-}name$，则 N 的计数应增加1；否则创建一个新节点 N，将其名称 *node-name*、计数 *node-count* 应分别设置为 p 和1，由父节点指针 *node-parent* 链接到它的父节点 T，并通过节点链 *node-link* 将其链接到具有相同名称 *node-name* 的节点。如果 P 非空，递归调用 *insert_tree*(P,N)。

3.2 基于前缀树 Prefix-Tree 的挖掘算法 PTM

Algorithm PT-Mine(α,minsup)

```
Input:
α is the prefix path of α-prefix tree;
minsup is the minimum support threshold;
Description:
1:for all children a_i of α do
2: if a_i.count≥minsup then
3:     add α.a_i and α.a_i.count to the set of frequent pattern;
4:     PT-Mine(α.a_i,minsup);
5:   end if
6:   for all children subroot of a_i do
7:       sibroot = the right sibling of whose item equal to subroot.
           item;
8:     merge(subroot,sibroot);
9:   end for
```

10:end for

4 应用实例

对表1所示的交易数据库 DB,数据荐集 $I=\{a,b,c,d,e,f,g,h,i,k\}$,设最小支持度计数为2,则由前缀树构造算法可得 DB 的前缀树 Prefix-Tree,如图2(a)所示,并有频繁项集 $L=[a:3,c:3,d:3,e:3,g:2]$。当 a-prefix 挖掘完,ac-prefix 子树应与 c-prefix 子树合并,ad-prefix 子树应成为树根 *root* 的新子树 d-prefix,结果如图2(b)所示。同理,当包含项 c(c-prefix)的频繁模式挖掘完,其 cd-prefix 子树和 d-prefix 子树合并,结果如图2(c)所示。最后可得所有的频繁模式,如表2所示。

表1 交易数据库 DB

Trans ID	items	Frequent items
T1	c,d,e,f,g,i	c,d,e,g
T2	a,c,d,e,m	a,c,d,e
T3	a,b,d,e,g,k	a,d,e,g
T4	a,c,h	a,c

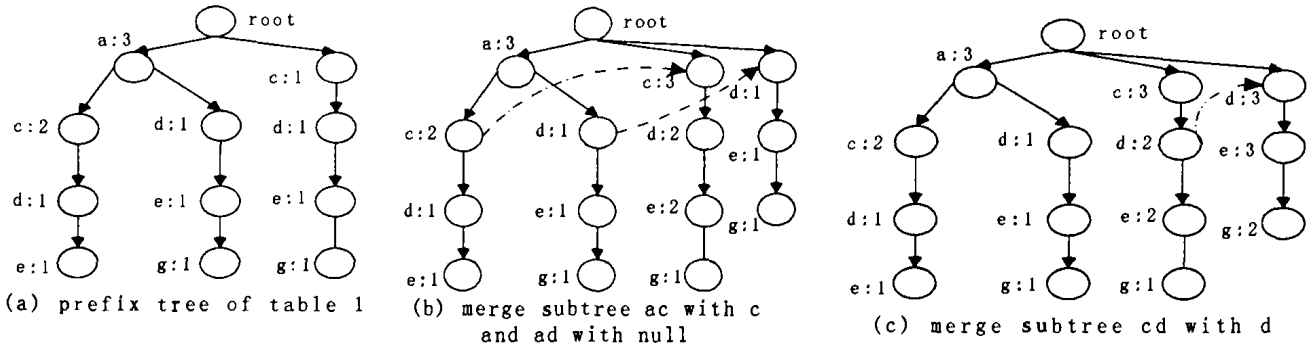

图2 An example

表2 交易数据库 DB 的频繁模式

Prefix	Frequent patterns
null(root)	a:3 c:3 d:3
a	ac:2 ad:2
ad	ade:2
c	cd:2
cd	cde:2
d	de:3
de	deg:2

5 算法实现与比较

我们用 VC++6.0在内存256M,CPU 为 Pentium Ⅲ-1GHZ,操作系统为 Windows 2000的机上实现了 PTM 算法并进行了性能测试。利用 http://www.ics.uci.edu/~mlearn/ML Summary.html 上提供的蘑菇数据库(mushroom database)来进行实验。该数据库有8124条记录,记录了蘑菇的23种属性。图3显示了在不同的最小支持度下算法的性能比较。由于 PTM 算法无须递归地创建大量的条件 FP-Tree,仅仅通过调整前缀树中节点信息和节点链直接在 Prefix Tree 上采用深度优先的策略挖掘频繁模式,而不需要任何附加的数据结构,因此 PTM 算法的执行时间比 FP-Growth 算法要少得多,如图3所示。

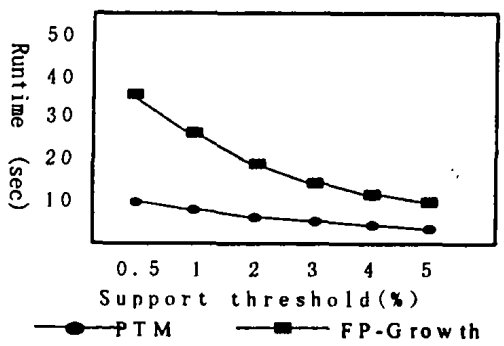

图3 Runtimes with some support thresholds

结束语 针对 FP-Tree 结构存在动态维护复杂、在挖掘过程中需要递归地创建大量的条件 FP-Tree,导致时空效率不高等缺点,本文提出一种基于前缀树的新算法。该算法通过引入一种新结构—前缀树(Prefix Tree)用来压缩存放数据的相关信息,通过调整前缀树中节点信息和节点链直接在 Prefix Tree 上采用深度优先的策略挖掘频繁模式,而不需要任何附加的数据结构,从而大大提高了挖掘效率。

参考文献

1 Agrawal R, Srikant R. Fast Algorithm for Mining Association Rules. In VLDB'94, Sep. 1994. 489～499

2 Park J S, Chen M-S, Yu P S. An Effective Hash Based Algorithm for Mining Association Rules. In: Proc. ACM SIGMOD Int. Conf. on Management of Data, 1995. 175～186

3 Chen M Y, Han J, Yu P. Data Mining: An Overview from a Database Perspective. IEEE Transactions on Knowledge and Data Engineering. 1996, 8(6): 866～883

4 Han J, Pei J, Yin Y. Mining Frequent Patterns Without Candidate Generation. In: Proc. ACM-SIGMOD, Dallas, TX, May 2000

5 Agarwal R, Aggrawal C, Prasad V V V. A Tree Projection Algorithm for Generation of Frequent Itemsets, Journal of parallel and Distributed Computing, 2000. 427～434

6 Huang H, Wu X, Relue R. Association Analysis with One Scan of Databases. In: Proc. of Pacific-Asia Conference, PAKDD, 2002. 334～334

7 Liu Guimei, Lu Hongjun, Xu Yabo, Yu J Xu. Ascending Frequency Ordered Prefix-tree: Efficient Mining of Frequent Patterns. In: Proc. of 8th Database Systems for Advanced Applications (DASFAA'03)