第 36 卷 第 4 期 计 算 机 工 程 2010 年 2 月
Vol.36 No.4 Computer Engineering February 2010
·软件技术与数据库· 文章编号：1000—3428(2010)04—0042—03 文献标识码：A 中图分类号：TP312

基于前缀树的高效频繁项集挖掘算法

才科扎西，黄景廉
(西北民族大学计算机科学与信息工程学院，兰州 730030)

摘 要：针对频繁项集挖掘时间与空间效率低的问题，提出一种基于前缀树的高效频繁项集挖掘算法，通过对事务集进行预处理，创建索引表并分配索引编号，保证前缀树中事务顺序的一致性，根据索引编号等信息创建紧凑的前缀树，采用自底向上的挖掘与投影的方式挖掘出频繁项集。实验结果表明，该算法挖掘效率高、占用空间少。

关键词：频繁项集；数据挖掘；前缀树

Efficient Frequent Item Set Mining Algorithm Based on Prefix Tree

CAIKEZAXI, HUANG Jing-lian
(College of Computer Science and Information Engineering, Northwest University for Nationalities, Lanzhou 730030)

【Abstract】Aiming at the problem of low time and space efficiencies for frequent item sets mining, an efficient frequent item sets mining algorithm based on prefix tree is proposed. To ensure the consistence of transactions sequence, the proposed algorithm pre-processes transaction sets to create index table and assign index identity. It creates compact prefix tree with the index information, and mines frequent item sets by bottom-to-up and projection methods. Experimental results show this algorithm has higher mining efficiency and expends less space.

【Key words】frequent item set; data mining; prefix tree

1 概述

近年来，关联规则挖掘逐渐成为一个具有实用意义的挖掘技术[1]。挖掘关联规则是从已知数据中得出所有满足最小支持度和最小置信度的关联规则[2]。事实上，挖掘关联规则的核心问题是求出数据库中满足最小支持度的所有频繁项集。目前，挖掘频繁项集的经典算法是 Apriori 算法[2]和 FP_growth 算法[3]，其他大部分算法都是这 2 种算法的变种[4]。这些方法在挖掘频繁项集前必须预先设置最小支持度，如何设置合理的最小支持度对挖掘的效率影响巨大。为提高挖掘效率，文献[5]对 FP_growth 树进行改进，提出一种压缩的事务序列树(Compressed and Arranged Transaction Sequences Tree, CATST)，但 CATST 结构与挖掘过程复杂，挖掘效率有待进一步提高。

本文针对频繁项集挖掘效率低的问题，提出一种基于前缀树的高效频繁项集挖掘算法。

2 提出的算法

提出的算法基于前缀树简化频繁项集的构建与挖掘过程，包括 3 个步骤：数据预处理，前缀树构建预处理与频繁项集挖掘。

2.1 数据预处理

对数据库中事务的集合 S ：

$$S=\{tr_1,tr_2,\cdots,tr_n\} \tag{1}$$

其中，tr_i 表示事务。每个事务包含多个项 t_i 。数据预处理首先对每个项 t_i 进行计数统计，然后按照其降序排列并进行索引编号，并建立索引表 T_{index} 记录 r_i ：

$$r_i=<n_i,t_i,s_i> \tag{2}$$

其中，n_i 为项 t_i 的索引编号；t_i 为项名称；s_i 为 t_i 出现的次数。根据上述统计信息，将原始事务信息转换为以索引编号为依据的事务信息。数据预处理的算法如下：

算法 1 数据预处理算法

输入 原始事务集合 S

输出 变换后的事务集合 S'

1：计算每个项 t_i 的支持度 s_i (出现的次数)；

2：创建索引表 T_{index} ；

3：根据支持值 s_i 对项进行降序排列；

4：根据项的排序分配索引号 n_i ；

5：对每个事务进行转换：用分配的索引号 n_i 取代 S 中的项名称 t_i ；

6：返回变换后的事务集合 S' 。

2.2 前缀树构建预处理

前缀树 T_{prefix} 直接对原始事务集合变换后的数据 S' 进行处理，频繁项集的挖掘在前缀树 T_{prefix} 构建后，由于 T_{prefix} 维持了原始事务的信息，无论对最小支持度进行怎样的设置，都不需要对数据进行重新扫描与重构。为了达到降低 T_{prefix} 构建时的空间消耗，T_{prefix} 的每个节点包含一个记录 rn_i ，该表包含了 2 个信息：项集的起始层号 l_i 以及该项出现的次数 s_i 。其中，每个项集起始层号 l_i 必须为树根或树的最左叶子节点，以保证不出现重复；为了便于挖掘出频繁项集，用一个数据链表记录每个数据项出现的次数 s_i ，并将相同的项串联起来。因此，前缀树 T_{prefix} 满足以下条件：

基金项目：国家“863”计划基金资助项目(AA2006010101)；教育部科学技术研究基金资助重点项目(105172)

作者简介：才科扎西(1964－)，男，副教授、硕士，主研方向：数据挖掘；黄景廉，高级工程师

收稿日期：2009-11-05 **E-mail：**caikezaxi@163.com

(1) T_{prefix} 中的父节点的索引编号小于子节点的索引编号。

(2) T_{prefix} 中的每个节点包含一个记录 rn_i，包括{起始层编号 l_i，事务出现的次数 s_i }。

(3) T_{prefix} 中的最左节点构成的左斜树(left skew tree)，其节点数等于索引表中的索引编号个数。

前缀树 T_{prefix} 构建预处理算法主要包括 2 步：首先根据索引编号的顺序创建左斜树；然后将数据预处理生成的转换事务集加入左斜树，形成前缀树 T_{prefix}。

在每个事务记录 tr_i 加入左斜树时，首先确定该记录的起始层号 l_0，l_0 为 tr_i 中第 1 个索引编号 n_1 对应倒左斜树中相同节点编号 n_1 的起始层号。若 n_1 为 0，则 l_0 为 0。然后将 l_1 设为当前节点 cn，把 tr_i 中下一个索引编号 l_2 加入树中，对比 cn 的子节点中是否有节点编号与 l_2 相同，若有，cn 等于此节点编号，继续找 tr_i 中下一个索引编号 l_3，对比 cn 的子节点是否有节点编号与 l_3 相同，依此类推，并有以下 2 种情况：

(1)若可找到 tr_i 中最后一个索引编号 l_{last} 所对应的节点编号，则判断此节点的记录表以存在的信息中起始层号是否与 l_0 相等，若有则将该信息的 s_i 值加 1，否则，在该记录表中加入信息(l_0 ,1)。

(2)若 tr_i 中某一索引编号 l_j 不属于 cn 的子节点，则将 tr_i 中 l_j 之后的索引编号依次序利用节点串起来成为 cn 的子树，并在此子树中的最后一个节点的记录表中加入信息(l_0 ,1)。

前缀树 T 构建预处理算法如下：

算法 2 前缀树构建预处理算法

输入 转换后的事务集合 S'

输出 前缀树 T_{prefix}

```
1：  If  T_prefix == NULL
2：      创建左斜树 T_left ;
3：  else for(i=0; i<=|S'|; ++i)
4：      if(i==0){
5：          将索引为 i 的节点设置为当前节点 cn ;}
6:       else {
7：          if( t_i  ==  当前节点 cn 的子节点索引)
8：              将子节点设为当前节点 cn ；
9：          else{
10：             创建新节点；
11：             将新节点的索引值设置为 t_i ；
12：             将新节点设为当前节点 cn ；}
13：     }
14： }
15： if(i = = |S'| - 1)
16：     修改当前节点 cn 的记录；
17： return  T_prefix ；
```

2.3 频繁项集挖掘

基于前缀树 T_{prefix} 的频繁项集挖掘包括 3 个步骤：向上累加，投影与自底向上挖掘。由于 T_{prefix} 中只有在每个事务 tr_1' 的最后一个节点处存放了信息记录 rn_i，因此在挖掘的过程中采用向上累加的方式。考虑到系统内存资源的限制，采用 2 种不同的方法：

(1)内存资源充足时，向上累加将在前缀树 T_{prefix} 构建完成后，自下而上从索引链表中搜索每个节点，当前节点信息中的起始层号小于父节点的起始层号时，并将当前节点的信息向上加到其父节点。

(2)当内存资源有限时，向上累加将在挖掘时完成，并采用暂存的方式实现下层的信息累加，当节点挖掘完成后，删除暂存信息以提高内存资源的利用率。

为了降低挖掘过程中的空间消耗，将前缀树进行投影。考虑到采用自底向上的挖掘方法中，当挖掘编号为 l_j 时，只需要对前缀树中所有编号为 l_j 的节点的子节点，因此，可对前缀树中搜索节点部分进行投影，得到子树。这种投影方式并不需要额外的节点来暂存一个新的投影树，不需占用任何额外的空间；同时由于投影过程简单，提高了挖掘的速度。

算法 3 频繁项集挖掘算法

输入 前缀树 T_{prefix}，索引链表 L，支持度阈值 ms

输出 频繁项集 FS

```
1：  对 L 中的每个记录 L[i]，由最后一个记录 tr_n' 开始；
2：  if( tr_i'.s_i >= ms ){
3：        将 L[i]对应的项 t_i 加入 FS；
4：        对 L[i]对应的项 t_i 构建投影树 T_prefix' ；
5：        为该树 T_prefix' 创建索引链表 L' ；
6：      根据投影树 T_prefix' 与其索引链表 L' 从最后一个记录
开始挖掘投影树；
7：  }
8： return FS；
```

3 实验结果

实验的硬件运行环境：CPU 为 Pentium 4，内存为 512 MB，硬盘为 120 GB 的计算机；软件运行环境：Windows2000 Server 操作系统，VC++6.0；实验数据采用 KDD 2005 中的 2 个数据集：BMS-WebView-1.dat 与 BMS-WebView-2.dat，其中，BMS-WebView-1.dat 包含 120 124 个事务，最大事务长度为 278；BMS-WebView-2.dat 包含 178 367 个事务，最大事务长度为 186。

为验证本文算法的有效性，将与文献[5]中的压缩事务序列树 CATST 进行对比。图 1 为最小支持度与事务数量变化时 2 种算法的挖掘时间。可以看出，随着最小支持度的增加，2 种算法的挖掘时间逐渐减少。

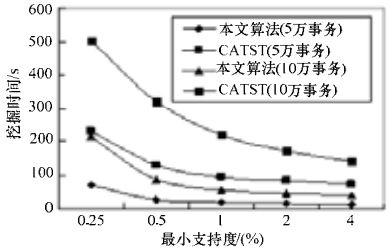

图 1　2 种算法的挖掘时间

对于 CATST 算法，事务数量为 5 万时，挖掘时间始终大于 73 s(最小支持度为 4%)，而提出的算法在最小支持度为 0.25%时只有 70 s，当最小支持度为 4%时，挖掘时间仅有 13 s，比 CATST 算法少 60 s，即提出的算法的挖掘时间仅为 CATST 算法的 1/5。当事务数量增加到 10 万时，CATST 算法的挖掘时间从最小支持度为 0.25%时的 503 s 降低到最小支持度为 4%的 140 s，而提出的算法在最小支持度为 0.25%时只有 214 s，当最小支持度为 4%时，挖掘时间仅有 40 s，远小于 CATST 算法。其原因在于 CATST 算法在挖掘的过程中，只要改变了最小支持度的值，CATST 算法需要重构条件树，

并进行节点的合并与移动操作，因此，增加了时间消耗。而提出的算法通过预处理，保证了创建的前缀树中事务的顺序的一致性，无论最小支持度如何改变，无需重新扫描事务与创建前缀树，减少了处理时间，另外，提出的算法在挖掘过程中采用投影的方式，简化了挖掘步骤，提高了频繁项挖掘的时间效率。

图 2 为事务数量变化时，2 种算法的创建树所需的时间性能。随着事务数量的增加，2 种算法的创建树所需的时间逐渐增加。对于 CATST 算法，创建时间从事务数量为 5 万时的 53 s 增加到 20 万时的 693 s；而提出的算法的创建时间从事务数量为 5 万时 50 s 增加到 20 万时的 620 s。2 种算法的创建时间差值随事务数量的增加而加大，其原因在于 CATST 算法构建压缩树的过程中不断地对高频项进行合并、向上调整以及对子节点进行互换，因此，构建树消耗的时间长。而提出的算法通过预处理，保证了创建的前缀树中事务的顺序的一致性，同时采用静态的前缀树构建方法，减少了节点合并与调整的处理时间，因此，构建所需时间更少。

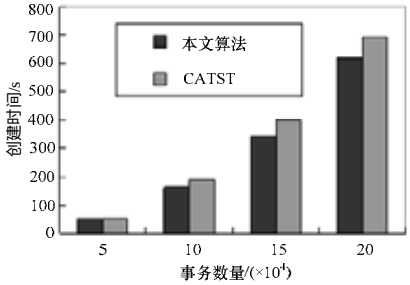

图 2　2 种算法创建树所需时间

图 3 为最小支持度变化时，2 种算法挖掘与创建树占用的空间的大小。随着最小支持度的增加，2 种算法挖掘时所需的空间将逐渐减少。对于 CATST 算法，最小支持度为 0.25% 时，挖掘空间为 9.1 MB，比相同条件下提出的算法所需空间大 2.3 MB；当最小支持度为 4%时，2 种算法挖掘所需空间差为 2.5 MB。其原因在于，当支持度增加时，符合条件的事务越少，创建的树所包含的节点数量越少，因此，所占空间越小；而 CATST 算法在挖掘过程中需要重复扫描与创建树，因此，所需空间大于提出的算法。另外，从创建树所占的空间看，2 种算法的空间大小随最小支持度的增加略有增加，CATST 算法创建树所占的空间略大于提出的算法。原因在于提出的算法通过预处理，保证创建的前缀树中事务的顺序的一致性；另外，提出的算法在挖掘过程中采用投影的方式，无需占用额外的空间，降低创建过程中的空间消耗。

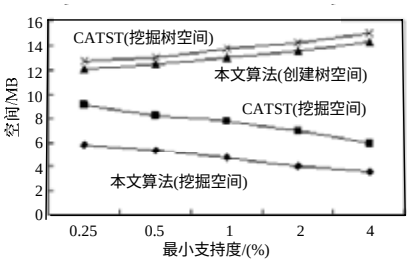

图 3　2 种算法的空间性能

4　结束语

本文提出频繁项集挖掘算法，通过对原始数据的预处理，构建索引表与索引编号确保了前缀树中事务顺序的一致性，降低了前缀树构建与挖掘的复杂性，基于紧凑的前缀树，采用自底向上的挖掘以及投影的方式挖掘出频繁项集，在降低空间消耗的同时提高挖掘的时间效率。实验结果表明，与 CATST 算法相比，本文算法不仅挖掘效率更高，而且所需空间更小。

参考文献

[1] Lee Wan-Jui, Jiang Jung-Yi, Lee Shie-Jue. Mining Fuzzy Periodic Association Rules[J]. Data Knowledge Engineering, 2008, 65(3): 442-462.

[2] Agrawal R, Imielinski T, Swami A. Mining Association Rules Between Sets of Items in Large Databases[C]//Proceedings of the ACM SIGMOD'93. Washington D. C., USA: ACM Press, 1993.

[3] Han Jiawei, Pei Jian, Yin Yiwen. Mining Frequent Patterns Without Candidate Generation[C]//Proceedings of the ACM SIGMOD'00. Dallas, TX, USA: ACM Press, 2000.

[4] 曹洪其, 姜志峰, 孙志挥. 基于 FP-tree 的多层关联规则快速挖掘算法[J]. 计算机工程, 2007, 33(19): 66-69.

[5] Zaiane R. Incremental Mining of Frequent Patterns Without Candidate Generation or Support Constraint[C]//Proc. of the 7th Int'l Conf. on Database Engineering and Applications. [S. l.]: IEEE Press, 2003.

编辑　陈　文

(上接第 41 页)

实验结果都随着样本长度的增长而提高，抽准率会逐步趋于一个值，样本长度继续增长时两者结果差别不大。

以上分析说明，当样本长度不大时，建立页面信息本体描述模型的信息抽取所做的改进效果明显。虽然随着样本的增长抽准率的差别不大，但实际应用中，用户一般不会有耐心去操作多个样本，因此，在小样本集(<7)时，提高了数据项的抽准率。

5　结束语

本文在 Web 页面信息项本体的基础上，结合 DOM 树技术，归纳学习页面集中稳定出现的信息块结构，对样本页面进行预处理。经过预处理后的页面能减少噪声信息对抽取精度的影响。该方法适用于信息项内容的结构变化不是很大的 Web 页面，对于信息项的结构变化较大的情况准确率还有待提高；该方法中划分页面信息抽取区域和抽取规则构建时都用到归纳学习的方法，2 次归纳学习的使用必然会增大程序处理的时间复杂度和空间复杂度。这是下一步需要继续研究的问题。

参考文献

[1] Berners L T. The Semantic Web[J]. Scientific American, 2001, 284(5): 34-43.

[2] 刘　耀, 穗志方. 领域 Ontology 概念描述体系构建方法探析[J]. 大学图书馆学报, 2006, 24(5): 28-33.

[3] 周明健, 高　济, 李　飞. 基于本体论的 Web 信息抽取[J]. 计算机辅助设计与图形学学报, 2004, 16(4): 535-541.

[4] 于　琨, 蔡　智, 糜仲春, 等. 基于路径学习的信息自动抽取方法[J]. 小型微型计算机系统, 2003, 24(12): 2147-2149.

编辑　陈　文