文章编号:1001 - 9081(2011)01 - 0101 - 03 doi:10.3724/SP. J. 1087. 2011. 00101

基于改进的 FP-tree 的频繁模式挖掘算法

李也白[1],唐 辉[1],张 淳[2],贺玉明[1]
(1. 北方工业大学 信息工程学院,北京 100144; 2. 北京地坛医院 器械科 北京 100015)
(tanghuiluck@ gmail. com)

摘 要:FP-growth 算法是一种基于 FP-tree 数据结构的高效的频繁模式挖掘算法,它不产生候选集。构造频繁模式树 FP-tree 需扫描数据库两次,在第二遍扫描中还扫描了那些仅包含了非频繁项的事务,针对此问题,在深入分析了 FP-tree 特性的基础上,改进了 FP-tree 构造过程,同时用一种基于 Hash 表的辅助存储结构,节省了项目查找时间,提高了挖掘效率。

关键词:数据挖掘;关联规则;频繁模式;FP-growth 算法;FP-tree
中图分类号: TP311.13 **文献标志码**:A

Frequent pattern mining algorithm based on improved FP-tree

LI Ye-bai[1], TANG Hui[1], ZHANG Chun[2], HE Yu-ming[1]
(1. *College of Information Engineering, North China University of Technology, Beijing* 100144, *China*;
2. *Instruments Division, Beijing Ditan Hospital, Beijing* 100015, *China*)

Abstract: FP-growth is an efficient frequent pattern mining algorithm based on the data structure of FP-tree, which does not generate candidate sets. Constructing frequent pattern tree TP-tree requires to scanning data twice. What's more, transactions which only contain non-frequent items are also scanned during the second scanning. In order to solve this problem, after analyzing particularity of FP-tree deeply, this paper improved construction process of FP-tree and employed an auxiliary storage structure that bases on hash table, which saves time of searching items and enhances mining efficiency.

Key words: data mining; association rule; frequent pattern; FP-growth algorithm; FP-tree

0 引言

关联规则挖掘[1-2]是数据挖掘的一个重要研究课题,是一种从大型数据库或数据仓库中发现隐藏在其中有用信息的一种技术。在关联规则挖掘研究中,频繁项集的挖掘是关联规则挖掘应用中的关键技术和基本步骤。目前通常采用的一种策略是将关联规则的挖掘过程分为两个步骤:第一步生成频繁项集,其任务就是根据所设定的最小支持度阈值生成全部的频繁项集;第二步生成关联规则,其任务是从上一步生成的所有频繁项集中根据设定的最小的置信度阈值提取所有满足条件的规则。这两步中,第二步相对容易实现,因此挖掘关联规则的总体性能基本由第一步决定[3]。

为解决频繁模式挖掘问题,由 R. Agrawal 等人于 1993 年首次提出了 Apriori 算法[2]。它是一种宽度优先算法,采用逐层迭代的候选项集生成策略,在挖掘过程中会产生大量的候选项集,并且需要反复扫描数据库,当数据库较大时,效率较低。针对 Apriori 算法的缺陷,Han Jiawei 等人提出了一种基于频繁模式树(FP-tree)[3-4]进行频繁模式挖掘的 FP-growth 算法[5]。与 Apriori 算法相比,该算法具有以下的特点:不产生候选项集,并且只需要扫描数据库两次,这大大提高了频繁项集挖掘的效率。然而 FP-growth 算法在挖掘过程中要创建复杂的数据结构,绝大部分时间主要消耗在构造和遍历 FP-tree及条件 FP-tree 上。研究者们还提出了许多其他的改进算法,Smart-Miner 算法[6]使用了一种动态减枝技术,在 FP-tree中快速地剪掉那些不频繁的候选频繁项集;FP-Growth* 算法[7]使用二维数组来存储 FP-Tree 中任意两个项目同时出现的次数,以提高时间效率;PatriciaMine 算法[8]使用一种类似于 FP-tree 的 Patricia Tries 数据结构表示频繁项集信息。本文提出了一种改进的 FP-tree 算法,并采用一个基于 Hash 表的数据结构辅助存储频繁项集,节省了 FP-tree 的创建时间。

1 改进的 FP-tree 构造算法

1.1 FP-growth 算法

FP-growth 算法采用分而治之的思想,把数据库中的所有频繁项目集都压缩到一棵频繁模式树(FP-tree)上,形成一个投影数据库,同时依然保留项集之间的关联信息,然后将这棵频繁模式树分成一些条件子树,再分别对这些条件子树进行挖掘。整个挖掘过程分为两步:1)扫描事务数据库取得频繁 1-项集,根据频繁 1-项集构造项目头表,再次扫描事务数据库,并根据项目头表构造频繁模式树;2)挖掘频繁模式树中的所有频繁项集。以上简单介绍了 FP-growth 算法的基本思想,更详细的内容请参见文献[9]。

1.2 改进的策略

主要是从以下两个方面对 FP-tree 构造算法进行改进。

收稿日期:2010 - 06 - 08;**修回日期**:2010 - 08 - 10。

作者简介:李也白(1957 -),男,辽宁沈阳人,教授,主要研究方向:数据挖掘、数据仓库、电子商务; 唐辉(1977 -),男,广西桂林人,硕士研究生,主要研究方向:数据挖掘; 张淳(1968 -),女,北京人,硕士研究生,主要研究方向:数据挖掘; 贺玉明(1985 -),女,河北唐山人,硕士研究生,主要研究方向:数据挖掘。

1)用一个基于Hash表的数据结构辅助存储项目头表中的项目及对应的位置,提高项目的查找效率。在构造FP-tree的过程中会频繁地在项目头表中查询项目,由于项目头表需要按项目支持数降序顺序存储,因此只能采用基于数组这样的顺序存储结构。顺序存储结构的查询效率较低,其查找时间复杂度为$O(n)$,但可以进行随机读取。

哈希表是一种根据元素关键字的值,通过哈希函数(一种将关键字映射到存储空间中特定位置的函数关系)计算得出该元素所存储的位置,它的查找时间复杂度为$O(1)$。设哈希函数为$f(key)$,其中key表示关键字,当要访问关键字为$k1$的项时,只要计算$f(k1)$即可得到$k1$的存储地址。本文先用一个数组依次存储排序后的全部1-频繁项集作为项目头表,再用一个基于Hash表的数据结构(键—值)辅助存储项目及其在项目头表中对应的位置。这种数据结构是一个数组和链表的结合体。当往这个结构插入项目时,先根据项目计算出它的Hash值,再根据这个Hash值得到这个项目在数组中的位置,如果数组该位置上已经存放有其他项目了,那么在这个位置上的项目将以链表的形式存放,新加入的放在链头,最先加入的放在链尾。如果没有项目,就直接将该项目放到此数组中的该位置上。

当需要从项目头表中查询项目时,先以这个项目为键在辅助存储结构中查找,取出该项目的值,这个值也就是该项目在项目头表中的位置,根据这个位置再从项目头表中随机读出。通过这样的改进,既保证了项目头表的顺序结构,又利用了Hash表的优点,可以快速查找频繁项目,从而提高了FP-tree的创建效率。

2)减少第二遍扫描事务数据库时的事务数量。在传统的FP-tree构造算法中,两遍扫描事务数据库的过程都是独立的,第二遍也扫描了全部的事务,这些事务中很多只包含了非频繁项目,对于这样的事务可以不用再扫描。因此可以根据第一遍扫描的结果,取得那些只包含频繁项目的事务,在第二遍扫描时,则不需扫描那些仅包含非频繁项目的事务。随着最小支持度的增加,频繁项集变得越来越少,需要扫描的事务相应也变得越来越少,从而可以减少大量不必要的扫描,节省FP-tree的构建时间。

1.3 改进的步骤

第一遍扫描事务数据库,计算每条事务中每个项的支持数。用一个与项目头表类似的数据结构来存储扫描结果,记为L,L中含三个域:项目名、支持数和事务ID集,其中事务ID集用于存储包含该项的所有事务ID集。第一遍扫描结束后,L中存储了事务数据库中所有单个项目的支持数及每个项目的事务ID集。

根据设定的最小支持度阈值,遍历L生成频繁1-项目集F。在得到频繁1-项集的同时,也可以得到有效事务ID集,其实现过程是:创建一个初始为空的集合S,在遍历结果L时,如果某项是频繁项,则将其事务ID集取出添加到S中,否则不作处理。这样当遍历完结果L时,S中就是所有包含了频繁项的事务ID集,第二遍扫描只需要扫描S中的事务即可。

在构造项目头表之前,还需要对F按支持数降序排序,然后根据排序后的F创建项目头表。在构造项目头表的同时创建一个基于哈希表的辅助存储结构H,在H中保存每个项目及其在项目头表中的位置。以后所有基于项目头表的查询,都转向在H中进行查询,找到后再从项目头表中读取项目。

1.4 改进算法伪代码

改进后的算法创建项目头表的伪代码表示如下。

```
BuildItemHeadTable(D,min_sup)
输入:事务数据库 D,最小支持度阈值 min_sup
输出:项目头表 H,有效事务 ID 集 valid_idset,辅助存储项目头表的 Hash 表 hash_ItemHead
1)  创建用于存储扫描结果的 L;
2)  for each(Transaction Tr in D)
3)    for each(Item e in Tr)
4)      在 L 中查找 ID 为 e 的项;
5)      if( 不存在项目 e)
6)        将 e 插入到 L 中,并将 e 的支持数设置 1;
7)      else
8)        将 L 中的 e 项目的支持计数增加 1;
9)      把 Tr 对应的事务 ID 添加到 L 中 e 项目对应的事务 ID 集中;
10)   end for
11) end for
12) for each(Item e in L)
13)   if( e 的支持数大于或等于 min_sup)
14)     把 e 添加到 H 中;
15)   把 e 对应的事务 ID 集添加到 valid_idset 中;
16) end for
17) 对 H 按项目的支持数降序排序;
18) for each(Item e in H)
19)   以项目 e 为键,把 e 在 H 中的索引存储到 hash_ItemHead;
20) end for
```

与传统的构建算法不同的是,改进算法在第二遍扫描事务数据库D时不再扫描全部的事务,而是先根据valid_idset对事务数据库D进行过滤,使D中只包含valid_idset中的所有事务,记为D'。构建FP-tree伪代码表示如下。

```
BuildFPTree(D', H, hash_ItemHead)
输入:事务数据库 D',项目头表 H, 辅助存储结构 hash_ItemHead
输出:FP-Tree T
1)  创建一棵只包含根节点的 FP-Tree T, 并把根节点置为当前父节点;
2)  for each(Transaction Tr in D')
3)    在 hash_ItemHead 查找 Tr 中的每一项, 选出 Tr 中包含在 hash_ItemHead 中的所有项, 并按照项目在 H 中的顺序排列, 形成只包含频繁项的 Tr';
4)    for each(Item e in Tr')
5)      if( T 的当前父节点的孩子节点中存在项目 e)
6)        将 T 中的节点名为 e 的节点的支持数增加 1;
7)        将 T 中的节点名为 e 的节点置为当前父节点;
8)      else
9)        将 e 插入 T 中;
10)     在 hash_ItemHead 找到项目 e 的位置, 把 T 中新插入的节点 e 连接到 H 中 e 的同名节点链上;
11)     将 T 中新插入的节点 e 置为当前父节点;
12)   end for
13) end for
```

2 实验及性能分析

实验的运行环境如下:CPU为Intel Core2 2.8 GHz,内存为2 GB,操作系统为Windows XP Professional SP2,使用的编程语言为Java,编程环境为MyEclipse8.0。实验使用的数据

集为 T10I4D100K 和 retail,从文献[10]下载。这两个数据集都属于稀疏型数据集,其中 T10I4D100K 总共有 100 000 条记录,包含 870 个项目,这些项目的支持数分布也较均匀,最大项目支持计数为 7 828;而 retail 是更稀疏型数据集,它的记录数是 88 162,包含项目 8 708 个,项目的支持数跨度较大,最大项目支持计数为 50 675。实验分别对这两个数据集从两个方面对改进前后的算法进行对比分析。一个方面是测试使用基于哈希表的辅助存储结构后对 FP-tree 创建时间的影响,对两个数据集都选择支持度为 0.1 的条件下,每次加载不同的事务数量进行测试,实验结果如图 1 和图 2 所示;另一个方面是测试过滤掉无效事务后对 FP-tree 创建时间的影响,对两个数据集都加载全部事务,每次选择不同的支持度,实验结果如图 3 和图 4 所示。

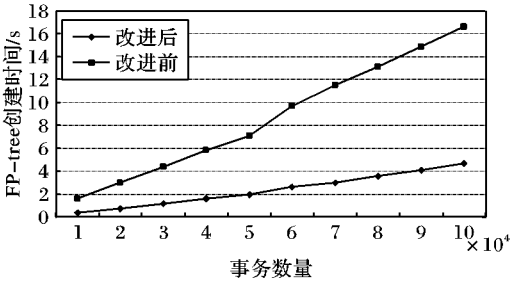

图 1　对 T10I4D100K 不同事务数量效率比较

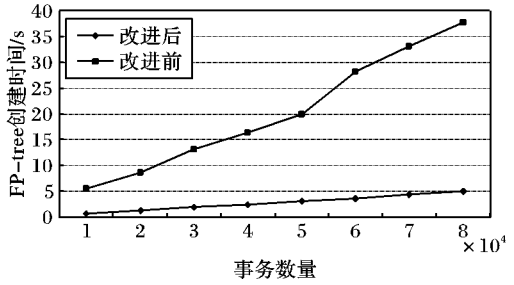

图 2　对 retail 不同事务数量效率比较

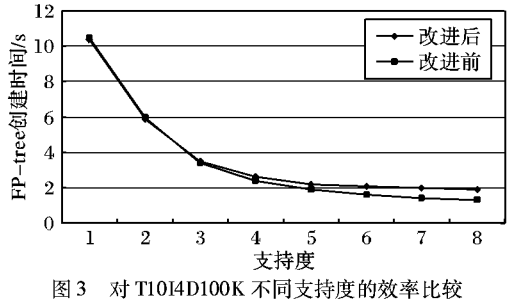

图 3　对 T10I4D100K 不同支持度的效率比较

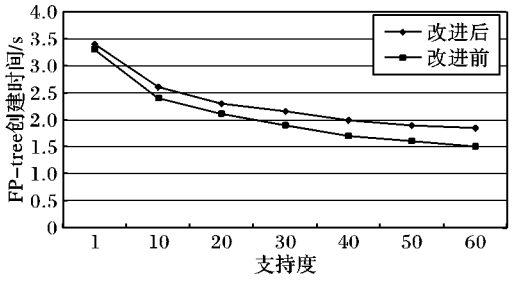

图 4　对 retail 不同支持度的效率比较

从测试结果可以看出,采用基于 Hash 表的辅助存储结构改进后,FP-tree 创建算法效率有了很大的提高,特别是在 1-频繁项数量较大时,效果更加显著。其主要原因是由于哈希表的快速查找性能,因此在频繁项较多、事务数量越大的情况下,节省的时间也就越多。从图 3 和图 4 中可以看出,随着支持度的增加,有效事务变得越来越少,改进后算法的效率也变得更加明显,当支持度增加到一定程度,FP-tree 的创建时间趋向一个稳定值,这部分时间中的绝大部分时间成本都消耗在第一遍扫描事务数据库及创建项目头表上,而这部分时间不会发生变化。实验结果证明了改进算法的有效性。

3　结语

本文对 FP-Tree 的创建算法进行了探讨,通过分析 FP-Tree 的建立过程,对 FP-Tree 的项目头表存储方式进行了改进,增加了一个辅助存储结构,虽然在一定程度上增加了空间开销,但是显著地减少了 FP-tree 的创建时间;再利用第一次扫描事务库的结果,减少了第二次扫描的事务数量,更进一步减少了 FP-tree 的创建时间。对于第二方面的改进,在稀疏型的数据集上效果比较明显,如果是稠密型的数据集,FP-tree 的创建时间没有太大改进。在对 FP-growth 算法进一步仔细分析与研究的基础上,发现该算法中的条件模式基的生成,以及在条件模式基之上生成条件子树,然后递归挖掘频繁项集部分,将产生很大的时间开销,在将来的工作中,将进一步研究对于遍历 FP-tree 挖掘频繁项集的复杂性的改进。

参考文献:

[1] AGRAWAL R, IMIELINSKI T, SWAMI A. Mining association rules between sets of items in large databases[C]// Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data. New York: ACM, 1993: 207 - 216.

[2] AGRAWAL R, SRIKANT R. Fast algorithms for mining association rules[C]// VLDB 1994: Proceedings of the 20th International Conference on Very Large Database. [S.l.]: Morgan Kaufmann, 1994: 478 - 499.

[3] HAN JIAWEI, KAMBER M. Data mining: Concepts and techniques [M]. 影印版. 北京: 高等教育出版社, 2001.

[4] 马丽生, 邓辉文, 齐逸. 基于 FP-tree 的最大频繁项目集挖掘算法[J]. 计算机工程与设计, 2008, 29(2): 385 - 388.

[5] HAN JIAWEI, PEI JIAN, YIN YIWEN. Mining frequent patterns without candidate generation[J]. ACM SIGMOD Record, 2000, 29(2): 1 - 12.

[6] ZHOU QINGHUA, CHU W W, LU BAOJING. SmartMiner: A depth first algorithm guided by tail information for mining maximal frequent itemsets[C]// ICDM 2002: Proceedings of IEEE International Conference on Data Mining. Washington, DC: IEEE, 2002: 570 - 577.

[7] GRAHNE G, ZHU JIANFEI. Fast algorithms for frequent itemset mining using FP-trees[J]. IEEE Transactions on Knowledge and Data Engineering, 2005, 17(10): 1347 - 1362.

[8] PIETRACAPRINA A, ZANDOLIN D. Mining frequent itemsets using patricia tries[C] // FIMI '03: Proceedings of the 1st Workshop on Frequent Itemset Mining Implementations. Melbourne, Florida, USA: [s.n.], 2003: 204 - 208.

[9] 朱明. 数据挖掘[M]. 2 版. 合肥: 中国科学技术大学出版社, 2008.

[10] Frequent itemset mining implementations repository[EB/OL]. [2010 - 01 - 25]. http://fimi.cs.helsinki.fi.