基于排序FP-树的频繁模式高效挖掘算法*)

秦亮曦[1,2,3] 李 谦[2] 史忠植[1]

(中国科学院计算技术研究所智能信息处理重点实验室 北京100080)[1]

(中国科学院研究生院 北京100039)[2] (广西大学计算机与信息工程学院 南宁530004)[3]

摘 要 FP-growth算法是目前较高效的频繁模式挖掘算法之一。在FP-growth算法中,FP-树及条件FP-树的构造和遍历占了算法绝大部分的时间,如果能减少这方面的时间,则有望进一步改善算法的效率。本文给出了一个频繁模式挖掘算法SFP-growth。算法通过将FP-树有序化及采用高效排序算法等措施来提高FP-树构造的效率,从而使算法达到较高的效率。实验结果表明,SFP-growth是一个高效的频繁模式挖掘算法,其性能优于Apriori、Eclat和FP-growth算法。

关键词 数据挖掘,关联规则,频繁模式,排序FP-树

An Efficient Frequent Patterns Mining Algorithm Based on Sorted FP-Tree

QIN Liang-Xi[1,2,3] LI Qian[2] SHI Zhong-Zhi[1]

(Key Lab of Intelligent Information Processing, Institute of Computing Technology, Chinese Academy of Sciences, Beijing 100080)[1]

(Graduate School of Chinese Academy of Sciences, Beijing 100039)[2]

(College of Computer and Information Engineering, Guangxi University, Nanning 530004)[3]

Abstract FP-growth is a high performance algorithm for mining frequent patterns. In FP-growth algorithm, it costs most of the time in constructing and traversing the FP-tree and conditional FP-tree. If we can reduce the time consuming in tree construction and traversing, then the performance can be improved. In this paper, an improved algorithm, SFP-growth, is presented. The algorithm adopts sorted FP-trees to store the main information of the transactions. It also uses an efficient sorting algorithm and other techniques in the construction of trees. The experimental result shows that SFP-growth is an efficient algorithm, it outperforms Apriori, Eclat and FP-growth algorithm.

Keywords Data mining, Association rules, Frequent patterns, Sorted FP-tree

1 引言

自关联规则的概念于1993年[1]被提出以来,频繁模式挖掘就成为包括关联规则挖掘在内的许多数据挖掘任务的基础。频繁模式挖掘算法的效率关系到这些挖掘任务能否高效地完成。以下是关联规则和频繁模式的形式化定义:

设$I=\{i_1, i_2, \cdots, i_m\}$是项的集合,事务数据库$D=\langle T_1, T_2, \cdots, T_n\rangle$,其中的每个事务$T$是项的集合,$T\subseteq I$。如果$X\subseteq T$,则称$X$是一个项集(itemset),如果$X$中有$k$个元素,则称$X$为$k$-项集。关联规则是形如$A\Rightarrow B$的蕴含式,其中,$A\subset I$,$B\subset I$,且$A\cap B=\Phi$。规则$A\Rightarrow B$在$D$中的支持度(support),是$D$中包含$A\cup B$的事务占事务总数的百分比,即概率$P(A\cup B)$。规则$A\Rightarrow B$在$D$中的可信度(confidence),是在$D$中那些包含$A$的事务中,$B$也同时出现的概率,即条件概率$P(B|A)$。对于一个项集$X$,如果其支持度大于等于用户给定的阈值$MinSup$,则称$X$为频繁项集(FI:Frequent Itemset)或频繁模式。

关联规则的挖掘一般可分成两个步骤[2]:1)找出所有频繁模式;2)由频繁模式生成满足要求的关联规则。其中,第一步是关联规则挖掘中的关键步骤。由R.Agrawal等人提出来的Apriori算法[2]是关联规则挖掘的一个经典算法。算法的名称来源于算法中应用了频繁项集的先验知识,即:一个频繁项集的任一非空子集必定是频繁项集。因此,只要某一项集是非频繁的,则其超集就无须再检验。算法首先扫描一遍数据库计算各个1-项集的支持度,从而得到频繁1-项集L_1;然后采用迭代的方式,逐步找出频繁2-项集,3-项集,…,直至不再产生新的频繁项集为止。在计算频繁k-项集$L_k(k=2,3,\cdots)$时,先通过L_{k-1}自连接产生候选集C_k,利用一定的剪枝策略缩减C_k;再通过扫描数据库来计算候选集的出现频率,消除非频繁项,从而得到频繁k-项集L_k。许多早期的研究大都采用类似于Apriori的先产生候选集再进行测试的方法。

Zaki等人提出的Eclat算法[3]采用了纵向(vertical)数据表示方法,并通过网格(lattice)和项集聚簇技术来挖掘频繁项集。J.Han等人[4]提出了一种利用频繁模式树(FP-树)进行频繁模式挖掘的FP-growth算法。与Apriori算法相比,该算法具有以下的特点:(1)采用FP-树存放数据库的主要信息。算法只需扫描数据库两次,然后将关键信息以FP-树的形式存放在内存中,避免了因多次扫描数据库而带来的大量的I/O时间。(2)不需要产生候选集,从而减少了由于产生和测试候选集需要耗费的大量时间。(3)采用分而治之的方式对数据库进行挖掘,从而在挖掘过程中,大大地减少了搜索空间。实验结果表明[4],FP-growth算法的性能比Apriori算法快了一个数量级。

在FP-growth算法中,绝大部分时间主要是消耗在FP-

*)基金项目:国家自然科学基金(90104021,60173017)。秦亮曦 副教授,博士研究生,研究方向:数据挖掘、进化计算。李 谦 硕士研究生,研究方向:计算机应用。史忠植 研究员,博士生导师,主要研究方向:知识工程、机器学习、智能主体等。

树及条件 FP-树的构造与遍历上，如果能提高这方面的效率，将对提高算法的效率有较大的帮助。基于这样的分析，我们提出了对 FP-growth 算法的改进措施。首先，将 FP-树有序化（采用排序 FP-树），使得树中的每一个结点的子结点按照项的序号从小到大排列。这样，加入新结点时需要比较的结点数平均可减少一半，从而构造一棵树的平均时间可以减少一半。其次，算法中的排序操作采用高效的 QuickSort 排序算法。由于在构造最初的 FP-树及条件 FP-树时，都需要将待插入的项集先按照频度的降序排列，然后再插入 FP-树中，因此排序算法的效率对于整个算法的效率有较大的影响。此外，还采取了其它的优化措施，如将 item-no 按照 item-name 的次序排成一个列表，在将 item-name 转换为 item-no 时，通过列表可直接找到对应的序号。

本文第2节给出 SFP-树的定义及构造算法；第3节给出基于 SFP-树的挖掘算法；第4节是算法的性能比较实验结果及分析；最后是本文的结论。

2 SFP-树的定义与构造

排序频繁模式树是由文[4]的 FP-树进行改进而得。以下是排序频繁模式树的定义：

定义1 排序频繁模式树（SFP-树）是一种树结构，定义如下：

（1）它由以下三个部分组成：一个标记为“null”的树根，一棵以项前缀子树集作为树根的孩子所组成的树，以及一个频繁项头表。

（2）项前缀子树中的每个结点由5个域组成：*item-no*，*count*，*parent-link*，*child-link* 和 *node-link*。其中，*item-no* 记录该结点所代表的项在项头表中的序号，*count* 记录从根结点到该结点的路径上所代表的模式在所有数据库事务中出现的次数，*parent-link* 是指向父结点的指针，*child-link* 是指向第一个子结点的指针，而 *node-link* 则连接到 FP-树中与该结点具有相同 *item-no* 的下一个结点，如果没有下一结点，则为 null。具有相图父结点的结点按照 *item-no* 从小到大的次序排列。

（3）频繁项头表中的每个项由两个域组成：*item-name*（结点所代表的项名）和 *node-link* 的头指针（指向 FP-树中具有 *item-name* 对应 *item-no* 的第一个结点）。项头表中的项按照其出现频度的降序排列。

SFP-树与 FP-树不同之处主要在于：(1)FP-树中的结点保存的是 *item-name*，而 SFP-树中的结点保存的是 *item-no*，在输出模式时才将 *item-no* 换成 *item-name*。(2)FP-树中的结点是无序的，而 SFP-树中的结点是按照 *item-no* 从小到大的次序排列的。

例1 设事务数据库中的事务如表1所示，最小支持度阈值为3。

表1 一个事务数据库示例

TID	事务中的所有项	其中的频繁项（按频率降序排列）
100	f, a, c, d, g, i, m, p	f, c, a, m, p
200	a, b, c, f, l, m, o	f, c, a, b, m
300	b, f, h, j, o	f, b
400	b, c, k, s, p	c, b, p
500	a, f, c, e, l, p, m, n	f, c, a, m, p

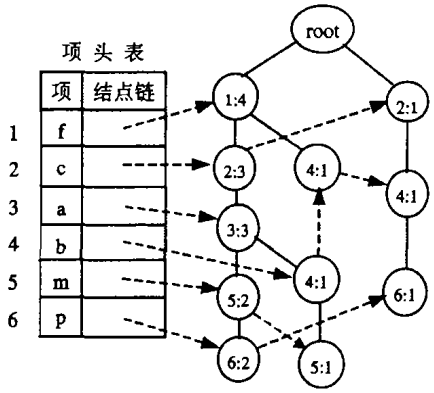

图1 表1的事务数据库对应的 SFP-树

算法1 SFP-tree Build

输入：一个事务数据库 DB 及最小支持度阈值 minsup
输出：建立后的排序频繁模式树 SFP-tree
方法：SFP_tree 按照以下步骤来建立。
1) 扫描事务数据库 DB 一遍，获得频繁1-项集及其支持度信息。将频繁1-项集按照支持度的递减顺序排列，记为 L。
2) 创建 SFP-树的根结点 T，标记为“null”。对于 DB 中的每个事务 Trans，执行以下操作：
将 Trans 中的每个频繁项用它在 L 中的序号代替，并按照从小到大的次序排列。记排序后的序列为[p|P]，p 为第一个元素，P 为其它元素。调用 inert_tree(p|P, T)。

Insert_tree(p|P, T)的实现方法为：

1) 如果 T 没有子结点，则 N. item-no=p，N. count=1，N 的父结点链指向 T；否则，在 T 的子结点中查找 p 的插入位置，如果找到与 p 相同的结点 N，则 N. count 加1；如果找不到相同结点，则将创建一个新结点 N，设置 N 的各个域的值（同前），将 N 插入到比 p 大的第一个结点之前。
2) 如果新加入了结点 N，则将 N 插入到项头表中第 p 个元素的相同结点链表的末尾。
3) 如果 P 非空，则递归调用 insert_tree(P, N)。

需要指出的是：在 SFP-树中，由于相同父结点的子结点是有序的，在加入新结点时只需要比较小于新的 item-no 的结点，而 FP-树则需要比较所有结点。所以，SFP-树加入一个新结点的平均时间是 FP-树的一半。而且，*item-no* 就是该项在项头表中的位置，不需要进行查找。在频繁1-项集和每个事务的频繁项集的排序中，采用了快速排序算法 QuickSort。

3 SFP-树挖掘算法

以下是 SFP-树的挖掘算法。

算法2 SFP-growth

输入：构造好的 SFP-树 T 及 minsup
输出：所有的频繁模式及支持度
方法：调用 SFP-growth (SFP-tree, null)

```
Procedure SFP-growth (T, α) {
(1) if 树 T 包含单一路径 P
(2) then 对路径 P 中的任一模式组合 β
      输出模式 β∪α(转换为 item-name)，模式支持度取 β 中的结点的
      最小支持度
(3) else {
(4)       for (i=n; i>=0; i--) { // n 为项头表的长度减1
(5)         β= i ∪α 且 sup(β) = sup(i) ;
(6)         构造 β 的条件模式库和条件 FP-树 Tβ;
(7)        if Tβ≠φ then call SFP-growth(Tβ, β);
(8) }}}
```

图2 SFP-growth 算法伪代码

当 SFP-树只包含单一路径时，算法输出路径上的所有模式组合与后缀的并集，再输出时需要将序号转换为项目名。否则，对项目号从 *n* 到0循环，分别求出以该项目号为后缀的频繁模式。在条件模式库和条件 FP-树的构造过程中，采用了快速排序算法和其它优化措施。

4 性能测试

在这一节中，我们将把 SFP-growth 算法与 Apriori、E-

clat 及 FP-growth 算法进行性能比较。测试环境为：Pentium Ⅲ 主频1GHz，内存384M，运行 Windows 2000 Professional 操作系统，SFP-growth 算法采用 Microsoft Visual C++6.0编写。Apriori 和 Eclat 算法的可执行代码是由 C. Borgelt[5]提供的。其中，Apriori 算法代码是为 SPSS 公司的 Clementine 数据挖掘平台编写的代码。算法中采用了许多优化措施，如算法不再重复地扫描数据库，采用事务树来计算候选项集的支持度等。该算法的性能已大大优于最初的 Apriori 算法，是公认的 Apriori 算法的最佳实现。FP-growth 算法的代码是由 B. Goethals[6]提供的。图中给出的运行时间是从读数据开始，一直到求出频繁模式的时间，但不包括输出结果的时间。比较的算法在同一数据集和相同支持度下得到的结果集完全相同。

本实验中采用了两个测试数据集：T25I20D100K 是利用文[2]中给出的人造数据生成算法生成的数据，具有100k 个事务，10k 个不同的项，每个事务平均项数为25。Connect-4是一个稠密数据集，来自机器学习数据资源库[7]。它具有67557个事务，130个不同的项，每个事务平均项数为43。图3中给出了在数据集 T25I20D100K 上的比较情况，性能大致为 SFP-growth＞Apriori＞FP-growth＞Eclat。其中 SFP-growth 的效率是 FP-growth 效率的5倍以上。图4中给出了在数据集 Connect-4上的比较情况（其中 Y 轴坐标是取了 log 后的值），性能大致为 SFP-growth＞FP-growth＞Apriori＞Eclat（其中，当 sup＞70%时，Eclat＞Apriori）。SFP-growth 的效率是 FP-growth 效率的2～5倍，两者的效率都优于 Apriori 和 Eclat 算法。

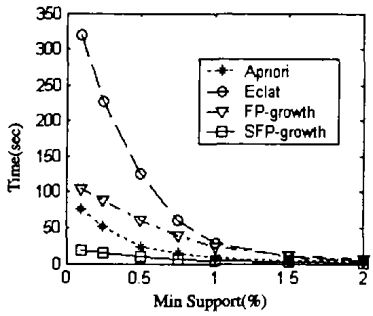

图3 在 T25I20D100K 上的性能比较

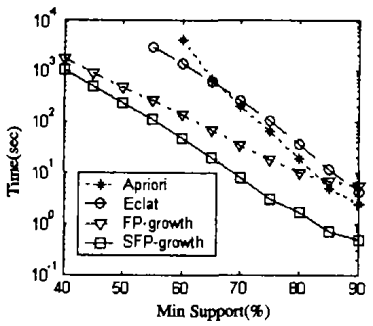

图4 在 Connect-4上的性能比较

讨论：从测试结果可以看出，SFP-growth 的性能对于两个数据集都是效率最高的，其它的算法在不同的数据集上有不同的表现。其原因主要在于：首先，是数据集的特性不同，T25I20D100K 的项集较大，频繁项集较分散，是一个稀疏型的数据集；而 connect-4数据集的项集较小，数据非常稠密。其次，是各算法采用的数据表示模式及挖掘策略不同。采用优化措施后的 Apriori 算法，对于非稠密数据已经具有较高的效率，其性能甚至优于 FP-growth 算法；但由于其采用的是广度优先的挖掘策略，对稠密数据效率仍较差。而 Eclat 算法采用的纵向表示法，对数据集较小的稠密数据，效率相对较高；但对于数据集较大的稀疏数据，效率较低。FP-树浓缩了数据库的主要信息，分而治之的挖掘策略也使挖掘问题的复杂程度有所降低。SFP-growth 算法通过将 FP-树有序化，使构造 SFP-树的时间比 FP-树有所减少；其中新结点加入 FP-树的平均时间可减少一半，并且新结点连接到项头表的相同结点链时，可通过序号直接找到链头，减少了查找链头的时间。此外，采用高效排序算法也使得项集排序的时间大为减少，所以 SFP-growth 算法能达到较高的效率。

结论 本文经过分析认为，通过减少 FP-growth 算法中 FP-树及条件 FP-树的构造和遍历的时间，将可较大地提高算法的效率。基于这样的分析，我们给出了对 FP-growth 算法的改进方法，包括：将 FP-树有序化，采用高效的 QuickSort 排序算法等。我们给出了改进后的 SFP-growth 算法，并将算法与 Apriori、Eclat 及 FP-growth 算法进行了性能比较测试。测试结果表明，SFP-growth 算法是一个高效的频繁模式挖掘算法，其性能优于 Apriori、Eclat 和 FP-growth 算法。

参 考 文 献

1 Agrawal R, Imielinski T, Swami A. Mining association rules between sets of items in large database. In: P Buneman, S Jajodia eds. Proc. of 1993 ACM SIGMOD Conf. on Management of Data. Washington DC: ACM Press, 1993. 207～216

2 Agrawal R, Srikant R. Fast algorithms for mining association rules. In: J Bocca, M Jarke, C Zaniolo eds. Proc. of the 20th Int'l Conf. on Very Large DataBases (VLDB'94). Santiago: Morgan Kaufmann, 1994. 487～499

3 Zaki M, Parthasarathy S, Ogihara M, Li W. New algorithms for fast discovery of association rules. In: D Heckerman, et al eds. Proc of the Third Intl. Conf. on Knowledge Discovery and Data Mining (KDD'97). AAAI Press, 1997. 283

4 Han J, Pei J, Yin Y. Mining frequent patterns without candidate generation. In: M Dunham, J Naughton, W Chen eds. Proc. of 2000 ACM-SIGMOD Int'l Conf on Management of Data (SIGMOD'00). Dallas, TX, New York: ACM Press, 2000. 1～12

5 http://fuzzy.cs.uni-magdeburg.de/～borgelt/

6 http://www.cs.helsinki.fi/u/goethals/

7 http://www.ics.uci.edu/～mlearn/MLRepository.html