文章编号: 1001-0920(2005)08-0887-05

MAXFP-Miner: 利用 FP-tree 快速挖掘最大频繁项集

陈慧萍[1,2], 王建东[1], 叶飞跃[1]
(1. 南京航空航天大学 信息科学与技术学院, 南京 210016; 2. 河海大学 计算机信息工程学院, 江苏 常州 213022)

摘 要: 为提高频繁项集的挖掘效率, 提出了最大频繁项集树的概念和基于 FP-tree 的最大频繁项集挖掘算法 MAXFP-Miner. 首先建立了 FP-tree, 在此基础上建立最大频繁项集树 MAXFP-tree, MAXFP-tree 中包含了所有最大频繁项集, 缩小了搜索空间, 提高了算法的效率. 算法分析和实验表明, 该算法特别适合于挖掘稠密型及具有长频繁项集的数据集.

关键词: 数据挖掘; FP-tree; 频繁项集; MAXFP-tree
中图分类号: TP311 **文献标识码**: A

MAXFP-Miner: Mining Maximal Frequent Itemsets Efficiently by Using FP-tree

CHEN Hui-ping[1,2], *WANG Jian-dong*[1], *YE Fei-yue*[1]
(1. College of Information Science and Technology, Nanjing University of Aeronautics and Astronautics, Nanjing 210016, China; 2. College of Computer and Information Engineering, Hehai University, Changzhou 213022, China. Correspondent: CHEN Hui-ping, E-mail: chenhp@webmail.hhuc.edu.cn)

Abstract: In order to improve the efficiency of mining frequent itemsets, the concept of maximal frequent itemset tree and an efficient algorithm, MAXFP-Miner, based on FP-tree for mining maximal frequent itemsets are proposed. After the FP-tree is created, a maximal frequent itemset tree, MAXFP-tree, is built up to store all the maximal frequent itemsets. Therefore, this MAXFP-tree reduces the search space and improves the efficiency of the algorithm. The analysis on the algorithm and the results of experiment show that the algorithm is especially effective for mining dense datasets with long frequent itemsets.

Key words: Data mining; FP-tree; Frequent itemset; MAXFP-tree

1 引 言

数据挖掘是智能决策的重要手段, 而挖掘频繁项集是许多数据挖掘工作中的关键问题. 自 Agrawal[1] 提出频繁项集的概念后, 研究人员提出了多种频繁项集的挖掘算法, 旨在提高算法的效率和可扩展性. 频繁项集挖掘广泛应用于关联规则挖掘、序列模式挖掘、Web 日志分析和协同过滤算法中.

根据频繁项集的规模, 可将其分为完全频繁项集、闭合频繁项集和最大频繁项集. 大部分算法在挖掘过程中会产生完全频繁项集, 如 Apriori 算法[1] 及其改进算法 AprioriTid[2], AprioriHybrid[3], Partition[4], 基于 HASH 的 Apriori 算法[5] 等. Apriori 挖掘算法是基于候选集的算法, 在挖掘过程中多次扫描数据库, 并产生大量的候选集, 影响了算法的效率. FP-growth 方法[6] 基于模式增长而不产生候选集, 提高了算法的效率, 但在挖掘过程中仍会产生完全频繁项集.

挖掘完全频繁项集的算法在挖掘过程中要穷尽所有频繁项集, 算法的效率很低, 所以这类算法不适合具有长模式的数据集. 最大频繁项集可以蕴含完全频繁项集, 因此可将挖掘完全频繁项集问题转化为挖掘最大频繁项集问题. 研究人员提出了相应的

收稿日期: 2004-09-17; **修回日期**: 2004-11-05.
基金项目: 国家 973 计划项目(G1999032701); 江苏省自然科学基金项目(BK2002091).
作者简介: 陈慧萍(1964—), 女, 江苏无锡人, 副教授, 博士生, 从事人工智能、数据挖掘的研究; 王建东(1945—), 男, 南京人, 教授, 博士生导师, 从事智能决策、数据挖掘等研究.

算法用于挖掘最大频繁项集, 如MAXMINER[7], GenMax[8]等.

本文在 FP-growth 方法[6]的基础上, 提出了最大频繁项集树的概念和基于 FP-tree 的最大频繁项集挖掘算法MAXFP-Miner. 分析和实验表明, 该算法特别适用于稠密型且具有长频繁项集的数据集.

2 相关概念

给定项目集 $I = \{i_1, i_2, \cdots, i_m\}$, 项集 X, $Y \subseteq I$, 事务 $T = (\text{tid}, X)$. 如果 $Y \subseteq X$, 则称该事务包含项集 Y, 即该事务支持 Y. 事务数据库 TDB 由一组事务组成, 项集 X 在事务数据库 TDB 中的支持数为

$$\text{sup-count}(X) = \left|\{(\text{tid}, Y) \mid (\text{tid}, Y) \in \text{TDB} \wedge (X \subseteq Y)\}\right| \quad (1)$$

项集 X 在事务数据库 TDB 中的支持度是指 TDB 中包含 X 事务的百分比, 即

$$\text{sup}(X) = \frac{\left|\{\text{tid}, Y) \mid (\text{tid}, Y) \in \text{TDB} \wedge (X \subseteq Y)\}\right|}{\left|\text{TDB}\right|}. \quad (2)$$

定义 1 支持度不小于最小支持度 min-sup 的项集 X 称为频繁项集[1], 即 $\text{sup}(X) \geq \text{min-sup}$; 支持度小于 min-sup 的项集称为非频繁项集. 所有频繁项集的集合表示为 FI.

定义 2 频繁项集 X 中包含项的个数称为频繁项集的长度.

频繁项集具有如下性质[1]:

性质 1 如果 X 是频繁项集, 则 X 的任何子集都是频繁项集.

性质 2 如果 X 是非频繁项集, 则 X 的任何超集都是非频繁项集.

定义 3 给定频繁项集 X, 如果 X 的任何超集都是非频繁项集, 则 X 为最大频繁项集[7]. 所有最大频繁项集的集合表示为 MFI.

性质 3 频繁项集与最大频繁项集之间存在如下关系[8]:

$$\text{MFI} \subseteq \text{FI} \quad (3)$$

定义 4 FP-tree 中包含项集 X 所有分支的前缀项构成的事务子库称为项集 X 的条件模式库.

根据性质 1 和性质 3, 任何频繁项集都是最大频繁项集的子集, 所以挖掘所有频繁项集问题可转化为挖掘所有最大频繁项集问题, 这样可以减少候选集的数目. 特别是当挖掘具有长频繁项集的数据库时, 要挖掘出所有频繁项集是不现实的, 而挖掘最大频繁项集可以大大提高算法的效率.

3 FP-Growth 方法和 FP-tree[6, 9, 10]

FP-Growth 方法将挖掘频繁项集的过程分解为以下两步:

第 1 步 构造频繁模式树即 FP-tree. 在 FP-tree 中, 每个结点由 4 个域组成: item (项名), sup-count (结点对应项的支持数), node-link (指向相同项名下一个结点的指针), parent (指向其父结点指针). 为方便树遍历, 创建一个头表 Header table, 它由两个域组成: 项名 item 和结点链头 head of node-link (指向 FP-tree 中与之名称相同的第 1 结点). 构造频繁模式树 FP-tree 的过程 create(T) 描述如下:

算法: create(T), 构造频繁模式树 FP-tree;

输入: 事务数据库 TDB, 最小支持度 min-sup;

输出: TDB 对应的频繁模式树 T.

1) 通过扫描事务数据库 TDB, 产生所有的频繁 1- 项集及其支持数, 并按其支持数降序排列插入到 Header table.

2) 创建 FP-tree 的根结点 root, 记为 null. 对于 TDB 中的每个事务, 作如下处理: ① 去掉所有的不频繁项, 频繁项按 Header table 中的次序重新排列, 设排列后的结果为 $[p \mid P]$, 其中 p 是第 1 个项目, P 是剩余项目的列表. ② 调用 insert-tree($[p \mid P], T$), 其过程执行如下: 如果 T 有子女 N, 使得 $N_{\text{item}} = p$, 则 N 的计数增加 1; 否则创建一个新结点 N, 将其名称 item 设置为 p, 将 sup-count 设置为 1, 由 parent 链接到它的父结点 T, 并通过结点链 node-link 将其链接到具有相同 item 的结点; 如果 P 非空, 则递归调用 insert-tree(P, N).

从构造 FP-tree 的过程可以看出, FP-tree 包含了事务数据库频繁项的所有信息, 在同一分支上, 结点按支持数从高到低排列, 不同的分支可以共享共同的前缀分支, 因此 FP-tree 是事务数据库中频繁项集的压缩表示.

第 2 步 调用 FP-Growth 在 FP-tree 上挖掘出所有频繁项目集. FP-Growth 方法采用分而治之的思想, 将 FP-tree 分成一些条件模式库, 然后在这些条件模式库上进行递归挖掘. FP-Growth 算法详见文献[6].

表 1 示例数据库

TID	items	重新排序后的频繁项
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

表 1 给出了示例数据库, 如果 min-sup = 60% (min-sup-count = 3), 则事务中的频繁项为: f: 4, c: 4, a: 3, b: 3, m: 3, p: 3. 图 1 给出了与表 1 数据

库相应的 FP-tree.

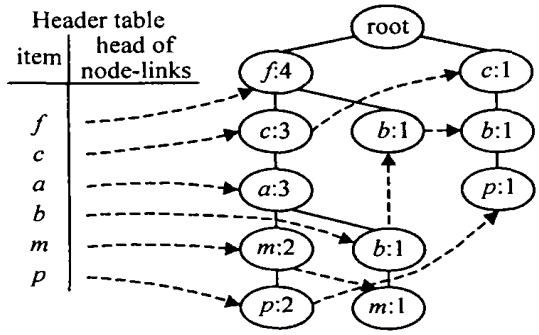

图 1　表 1 对应的 FP-tree

4　基于 FP-tree 的最大频繁项集挖掘算法 MAXFP-Miner

算法MAXFP-Miner 是建立在 FP-tree 的基础上, 在 FP-tree 上挖掘最大频繁项集树 MAXFP-tree.MAXFP-Miner 与 FP-Growth 方法类似, 但 FP-Growth 方法产生所有频繁项集, MAXFP-Miner 方法只生成最大频繁项集树.

4.1　最大频繁项集树MAXFP-tree

最大频繁项集树(最大频繁模式)MAXFP-tree, 就是树中从根结点到叶结点的每条完整路径均表示一个最大频繁项集. 在MAXFP-tree 中, 每个结点的结构与 FP-tree 中结点的结构相似, 但MAXFP-tree 中的结点没有 sup-count 域, 即每个结点有 item, node-link 和 parent 3 个域. MAXFP-tree 同样创建一个头表 Header table, 它由两个域组成: 项名 item 和结点链头 head of node-link (指向MAXFP-tree 中与之名称相同的第 1 结点). Header table 中项的次序与 FP-tree 头表中项的次序相同.

图 2 是图 1 FP-tree 所对应的最大频繁项集树. 因为树的每条从根结点到叶结点的完整路径都是最大频繁项集, 所以该例所对应的最大频繁项集为 $\{cp, fcam, b\}$.

4.2　MAXFP-tree 生成过程和MAXFP-Miner 算法

MAXFP-tree 是在 FP-tree 挖掘过程中逐步生成的, 其生成过程类似于 FP-growth 方法. 首先创建根结点, 记为 null. 如果前面生成的 FP-tree 只包含一条单一的路径, 则该路径对应的频繁项集一定是最大频繁项集, 将该路径直接插入到MAXFP-tree 中, 过程结束. 在大多数情况下, 初始的 FP-tree 不会只包含一条单一的路径, 这时采用类似于 FP-Growth 分而治之的方法, 继续生成 MAXFP-tree. 对 FP-tree 头表 Header table 中的每一项 i, 按支持数从低到高的顺序对其分别进行处理, 先构造 i 的条件模式库, 再分别在条件模式库上进行递归挖掘产生最大频繁项集.

现将MAXFP-Miner 算法描述如下:

```
Algorithm: MAXFP-Miner(T)
  //根据FP-tree挖掘最大频繁模式树MAXT
  //输入: T 为当前 FP-tree
  //全局变量: Head 为当前链接项列表
      MAXT 为最大频繁模式树
  //输出: 最大频繁模式树MAXT
1) If T 只包含单一路径 then
2)     将 T ∪ Head 直接加入到MAXT 中
3) else 对 Header-table 中每一项 i
       (以支持数递增的次序) 作如下处理:
4)   将 i 添加到 Head 中
5)   构造 Head 的条件模式库 H-base
6)   if H-base 不为空 then
7)     item-list = {H-base 中的频繁项}
8)     if item-list ∪ Head 没有包含
         在当前MAXT 的路径中 then
9)       由模式库 H-base 构建条件 FP-tree T'
10)      递归调用MAXFP-Miner(T',MAXT)
11)  从 Head 中删除 i
```

该算法为递归算法, 初始调用时 T 是由调用 create(T) 后产生的 FP-tree, Head 为空集 ∅, MAXT 为只有一个根结点 null 的树. 当递归调用结束时,MAXT 包含了所有最大频繁项集, 树中从根结点到叶结点的每条完整的路径都是一个最大频繁项集.

算法第 2 行将 T ∪ Head 直接加入到MAXT 中, 其过程与前面描述的 insert-tree 过程类似, 但不需考虑支持度的计算. 第 3 行按支持数从低到高的顺序处理 Header table 中的每一项 i, 这样可保证发现最大频繁项集总是先于其任一频繁子集. 第 5 行构造Head 条件模式库H-base 的过程, 以及第 9 行构建条件 FP-tree 的过程, 与 FP-Growth 方法中这两部分算法相同. 第 7 行进行子集检测, 用来判断当前最大频繁模式树MAXT 的某条路径中, 是否已包含待测试的频繁项集 item-list ∪ Head, 如果包含, 说明 item-list ∪ Head 不可能是新的最大频繁项集, 则不需递归调用; 如果没有包含, 则进行递归调用. 子集检测中采用 HASH 方法进行优化, 以提高算法的效率.

为更好地说明算法的思想, 对图 2 最大频繁模式树MAXT 的产生过程加以说明. 开始MAXT 只包含一个根结点 null, 算法的输入为图 1 中的 FP-tree. 这时 FP-tree 包含多条路径, 找到 Header

table 中支持数最低的项 p, Head = $\{p\}$, 建立其条件模式库 H-base 为 $\{(fcam: 2), (cb: 1)\}$, 则 item-list = $\{c: 3\}$. 因为 item-list ∪ Head = cp 没有包含在当前MAXT 的路径中, 所以建立 p 的条件 FP-tree 为 $c: 3$. 然后进行递归调用, 这时 FP-tree 只包含单条路径, 所以将 cp 插入到MAXT 中. 再找到项 m, Head = $\{m\}$, 建立其条件模式库 H-base 为 $\{(fca: 2), (fcab: 1)\}$, item-list = $\{f: 3, c: 3, a: 3\}$. 因为 item-list ∪ Head = fcam 没有包含在当前 MAXT 的路径中, 所以建立 m 的条件 FP-tree 为 $fca: 3$. 最后进行递归调用, 这时 FP-tree 只包含单条路径, 所以将 $fcam$ 插入到MAXT 中. 对于项 b, a, c 和 f, 以此类推. 于是得到最大频繁项集树如图 2 所示. 表 2 给出了各项的条件模式库及条件 FP-tree.

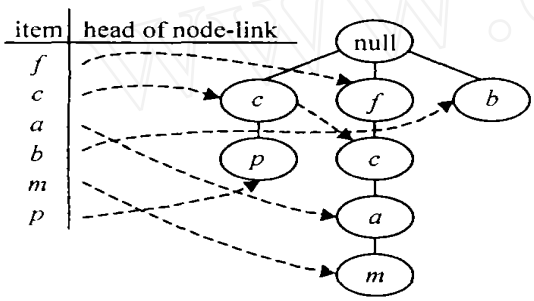

图 2 图 1 EP-tree 对应的最大频繁项集树

表 2 各项对应的条件模式库和条件 EP-tree

item	条件模式库	条件 FP-tree
p	$\{(fcam: 2), (cb: 1)\}$	$\{(c: 3)\}\|p$
m	$\{(fca: 2), (fcab: 1)\}$	$\{(f: 3, c: 3, a: 3)\}\|m$
b	$\{(fca: 1), (f: 1), (c: 1)\}$	∅
a	$\{(fc: 3)\}$	可以跳过
c	$\{(f: 3)\}$	可以跳过
f	∅	∅

5 算法分析与实现

算法MAXFP-Miner 建立在FP-tree 的基础上, 而 FP-tree 是数据库的压缩表示, 所以可节省存储空间; 而且MAXFP-Miner 在FP-tree 基础上只产生最大频繁项集, 因此大大缩小了搜索空间. 数据集可分为稠密型数据集和稀疏型数据集. FP-tree 对稠密型数据集具有优越性, 因为有许多可共享的前缀, 所以可节省内存, 并能加快挖掘效率.

衡量数据集的主要指标有: 数据集中包含的事务数、项目数、平均事务长度、最大事务长度. 如果平均事务长度接近最大事务长度, 则为稠密型数据集. 当数据集具有较短的平均事务长度且其最大频繁项集长度接近平均事务长度时, 如果要挖掘所有的频繁项集, 则搜索空间很大. 算法MAXFP-Miner 只挖掘最大频繁项集, 因此性能大大提高.

本文在 PⅢ 797/256M 计算机上用 C++ 编程, 对算法MAXFP-Miner 和算法 GenMax[8] 的性能进行测试. 将它们应用于典型的机器学习数据集 mushroom 和 chess, 这二个数据集均属于稠密型数据集, 其特征如表 3 所示. 挖掘时所产生的平均最大频繁项集长度不同, mushroom 既有较短的平均事务长度又有与平均事务长度接近的最大频繁项集, chess 既有较长的平均事务长度又有相对较短的最大频繁项集.

表 3 数据集的特征

数据集	事务数	项目数	平均事务长度	最大事务长度
mushroom	8 124	120	23	23
chess	3 196	76	37	37

在不同的最小支持度下, 用两种算法对两个数据集分别进行挖掘. 实验结果表明, 最小支持度 min-sup 越低, 同一算法挖掘出最大频繁项集所用的时间越长. 因为最小支持度 min-sup 越低, 频繁项集越长, 所产生的最大频繁项集也越多. 两种算法的性能对不同的数据集呈现出不同的结果: 对于 mushroom 数据集, MAXFP-Miner 的性能优于 GenMax, 如在 min-sup = 10% 时, MAXFP-Miner 的运行时间为 0.4 s, GenMax 的运行时间为 5 s, 且最小支持度越低, MAXFP-Miner 的性能越优于 GenMax, 最小支持度越高, 两种算法的性能越接近. 对于 chess 数据集, GenMax 的性能优于 MAXFP-Miner, 如在 min-sup = 40% 时, MAXFP-Miner 的运行时间为 70 s, GenMax 的运行时间为 17 s, 且最小支持度越高, GenMax 的性能越优于MAXFP-Miner, 最小支持度越低, 两种算法的性能越接近, 正好与 mushroom 数据集得到的结果相反. 实验结果表明, MAXFP-Miner 特别适合于稠密型并有较短平均事务长度和较长频繁项集的数据集.

该算法还有待在更多的数据集上进行测试. 对于新的数据集, 可以通过扫描数据集的方法, 确定数据集的特性(即数据集中包含的事务数、项目数、平均事务长度、最大事务长度), 也可确定潜在的最大频繁项集长度等. 如果属于稠密型数据集, 且其平均事务长度和潜在的最大频繁项集长度较接近, 则可采用MAXFP-Miner 算法.

6 结 语

本文在 FP-growth 方法的基础上, 提出一种基于 FP-tree 的最大频繁项集挖掘算法 MAXFP-

Miner. 实验结果表明,该算法特别适合于稠密型并有较短平均事务长度和较长频繁项集的数据集. 本算法还有待在更多的数据集上进行测试,有待进一步优化以提高算法的性能. 特别是实验中发现,算法第 8 行进行子集检测这一步花费的时间较长,所以需要进一步改进.

参考文献(References)

[1] Agrawal R, Imielinski T, Swami A. Mining Association Rules between Sets of Items in Large Databases [A]. *Proc of the 1993 ACM SIGMOD Int Conf on Management of Data* [C]. Washington: ACM Press, 1993: 207-216.

[2] Agrawal R, Srikant R. Fast Algorithms for Mining Association Rules[A]. *Proc 20th Int Conf on Very Large Data Bases* [C]. Santiago: Morgan Kaufmann, 1994: 487-499.

[3] Agrawal R, Mannila H, Srikant R, et al. *Fast Discovery of Association Rules: Advances in Know ledge Discovery and Data Mining* [M]. Boston: MIT Press, 1996: 307-328.

[4] Ashoka S, Edward O. An Efficient Algorithm for Mining Association Rules in Large Databases [A]. *VLDB* 1995[C]. Zurich, 1995: 432-444.

[5] Park J S, Chen M S, Yu P S. An Effective Hash-based Algorithm for Mining Association Rules [A]. *SIGMOD* '95[C]. San Jose, 1995: 175-186.

[6] Han J, Pei J, Yin Y. Mining Frequent Patterns without Candidate Generation: A Frequent-pattern Tree Approach [J]. *Data Mining and Know ledge Discovery*, 2004, 8(1): 53-87.

[7] Bayardo R J. Efficiently Mining Long Patterns from Databases[A]. *SIGMOD* '98[C]. Seattle, 1998: 85-93.

[8] Gouda K, Zaki M J. Efficiently Mining Maximal Frequent Itemsets[A]. *1st IEEE Int Conf on Data Mining* [C]. San Jose, 2001: 163-170.

[9] Han J, Micheline K. *Data Mining: Concepts and Techniques* [M]. Beijing: China Machine Press, 2001: 225-278.

[10] 朱玉全,孙志挥,季小俊. 基于频繁模式树的关联规则增量式更新算法[J]. *计算机学报*, 2003, (1): 91-96.
(Zhu Y Q, Sun Z H, Ji X J. Incremental Updating Algorithm Based on Frequent Pattern Tree for Mining Association Rules [J]. *Chinese J of Computers*, 2003, (1): 91-96.)

(上接第 886 页)

参考文献(References)

[1] 吴麒,高黛陵. *控制系统的智能设计*[M]. 北京: 机械工业出版社, 2003.
(Wu Q, Gao D L. *Intelligeut Design Methods for Control Systems* [M]. Beijing: Engineering Industry Press, 2003.)

[2] Mao W J, Chu J. Input-output Decoupling of Linear Time-invariable Sytems [J]. *Control Theory and Applications*, 2002, 19(1): 146-148.

[3] Hu Z J, Shi S J, Wen Z X. Application and Development of Linear Matrix Inequality in Control Theory[J]. *J of Shanghai Jiaotong University*, 1999, 33(11): 1458-1467.

[4] Jeremy G V, Richard D B. A Tutorial on Linear and Bilinear Matrix Inequalities [J]. *J of Process Control*, 2000, 10(5): 363-385.

[5] Li J, Wang O H, Niemann D. *Relationship between LMI and ARE with Their Applications to Absolute Stability Criteria, Robustness Analysis and Optimal Control* [R]. Durham: Duke University, 1998.

[6] Safonov M G, Chiang R Y. CACSD Using the State Space $L\infty$ Theory — A Desigen Example [J]. *IEEE Trans on Autamatic Control*, 1988, 33(5): 477-479.

[7] 郑大钟. *线性系统理论*[M]. 北京: 清华大学出版社, 2002.
(Zheng D Z. *Linear System Theory* [M]. Beijing: Tsinghua University Press, 2002.)

[8] 俞立. *鲁棒控制——线性矩阵不等式处理方法*[M]. 北京: 清华大学出版社, 2002.
(Yu L. *Robust Control — Linear Matrix Inequalities Method* [M]. Beijing: Tsinghua University Press, 2002.)

[9] Fu J, Yang W D, Liu T L, et al. Polynomial Approach Algorithm in Real Time Calculation of Looper Tension Moment [J]. *Control Engineering of China*, 2004, 11(3): 226-228.

[10] An B J, Park S H, Kim B Y, et al. *Tension Control System for Hot Strip Mill* [M]. Isie: Pusan Korean, 2001: 1452-1457.