

# 基于循环十字链表的频繁模式挖掘算法

段仰广, 韦玉科

(广东工业大学 计算机学院, 广东 广州 510006)

**摘 要:** FP-growth 算法是当前挖掘频繁模式的有效算法之一, 但 FP 树的节点占用空间较大, 长时间占用内存不释放, 挖掘过程中需要产生大量的条件 FP 树, 因而时空效率不理想。提出了一种循环十字链表结构用作存储事务数据库, 而不生成 FP 树, 在挖掘频繁项集的过程中, 这种链表结构逐步缩小, 减少了内存的使用率, 通过构建排序的条件频繁模式树挖掘频繁项集。理论分析和实验表明基于这种结构的排序条件频繁模式树挖掘频繁项集具有较好的时空效率。

**关键词:** 频繁模式; 循环十字链表; 排序条件频繁模式树

**中图分类号:** TP391.4

**文献标识码:** A

**文章编号:** 1673-629X(2009)10-0073-04

## Algorithm for Mining Frequent Patterns Based on Circular Orthogonal Linked List

DUAN Yang-guang, WEI Yu-ke

(Faculty of Computer, Guangdong University of Technology, Guangzhou 510006, China)

**Abstract:** FP-growth is one of efficient algorithms for mining frequent patterns, however, every node of FP-tree occupies a large memory space, and doesn't release for a long time, and importantly it needs to generate numerous re-construction of intermediate FP-trees during the mining process, so the efficiency of FP-growth remains unsatisfactory. Proposes a circular orthogonal linked list structure named Cir-Orthogonallist which used to store transactional database, so don't need to generate FP-tree. During the mining process the dimension of the new structure will be reduced gradually and the memory requirement lessens too. At last by building the sort condition FP-tree to get frequent patterns. Theoretical analysis and experimental results show that the algorithms based on the new structure proposed in this paper have good efficiency.

**Key words:** frequent pattern; circular orthogonal linked list; sort condition FP-tree

## 0 引言

发现频繁模式集是挖掘关联规则、相关分析、序列模式等许多数据挖掘任务的关键技术和基础步骤, 自从 Agrawal 提出频繁项集挖掘的 Apriori<sup>[1]</sup> 算法之后, 产生了各种改进的衍生算法, 然而这些算法在挖掘过程中产生大量的候选项集, 并且需要多次扫描数据库, 严重地影响了算法的效率<sup>[2,3]</sup>。J. Han 等人提出的 FP-growth 算法<sup>[4]</sup> 是一种在本质上不同于 Apriori 算法的有效算法。FP-growth 算法采用分治的策略, 无需产生候选项集, 只需扫描数据库两次, 提高了挖掘效率。然而它在时间和空间效率还不够高, 仍可以进一步改进。FP-growth 算法, 绝大部分时间花费在复杂

的 FP-树及条件 FP-树的构建与遍历上, 如果在这方面提高效率, 无疑将对算法效率提高有很大帮助<sup>[5]</sup>。基于这方面的分析, 提出了用带有循环的十字链表结构来替代复杂的 FP-树的措施, 并用排序的条件频繁模式树<sup>[6]</sup> 挖掘频繁项集。

每个循环十字链表节点占用空间比树节点少, 使在查找前缀条件模式集方面更快速, 循环结构使前缀按序号排序更方便快捷, 而不用像经典的 FP-growth 算法那样得到的前缀是升序的, 此外采用排序的条件频繁模式树挖掘频繁项集使该算法明显地优于 FP-growth 算法。

## 1 基本概念

### 1.1 频繁项集

设  $I = \{i_1, i_2, i_3, \dots, i_n\}$  是项的集合,  $D$  是事务数据库的集合, 其中每个事务  $T$  是项的集合, 使得  $T \subseteq I$ 。对于任何  $X \subseteq I$ , 事务  $T$  包含  $X$  当且仅当  $X \subseteq T$ 。  $X$

收稿日期: 2008-12-22; 修回日期: 2009-03-19

基金项目: 国家科技支撑计划课题(2006BAI08B01-03)

作者简介: 段仰广(1983-), 男, 山东济宁人, 硕士研究生, 研究方向为数据挖掘、计算智能与智能工程; 韦玉科, 副教授, 硕士生导师, 研究方向为智能信息处理、智能控制与监测。

称作项集。项集  $X$  的支持度计数是  $D$  中包含  $X$  的事务数目。项集  $X$  的支持度是  $D$  中包含  $X$  的事务的比例,如果  $D$  中的事务总数为  $n$ ,那么  $X$  的支持度是  $X/n$  得到的百分比。如果项集  $X$  的支持度大于等于某个给定的支持度,称  $X$  为频繁项集,该给定的支持度为最小支持度。

1.2 FP-tree 和 FP-growth 算法

频繁模式树即 FP-tree 中,每个节点由 3 个域组成:项名 item、节点支持度计数 sup\_count 及节点链 node\_link。为方便遍历,创建一个项头表 Header table,它由 2 个域组成:项名 item 和节点链头 head of node\_link,其中节点链头指向 FP-tree 中与之名称相同的第一个节点。FP-growth 算法<sup>[7]</sup>主要是 FP-tree 的构造过程,需要扫描两次数据集:

(1) 第一次扫描数据集  $D$ ,产生所有频繁 1-项集及其支持度计数,按其支持度降序排列插入到项头表。

(2) 创建 FP-树  $T$  的根节点,用“null”标记,对  $D$  中每个事务做如下处理:①按项头表中的次序排列第一次扫描得到的频繁项集,设排列后的结果为  $[p \mid P]$ ,其中  $p$  是第 1 个项目,而  $P$  是剩余项目的列表;②调用 insert\_tree( $[p \mid P]$ ),如果  $T$  有子女  $N$  使得  $N.item = p$ ,则  $N$  的计数增加 1,否则创建一个新节点  $N$ ,将其名称 item 设置为  $p$ ,将 sup\_count 设置为 1,链接到它的父节点,并通过节点链 node\_link 链接到具有相同项名的节点,如果  $P$  非空,递归调用 insert\_tree( $[P \mid N]$ )。

2 替代 FP-树的循环十字链表

FP-树的创建,是一个递归调用 insert\_tree( $P \mid N$ )的过程,比较耗费内存资源,在每次调用自己时,都会在内存中生成一个函数副本;多次用到堆栈,效率较低。改用循环结构和尽力减少 FP-树的构建和遍历就可以提高算法的速度。使用带有循环的十字链表结构 Cir-OrthogonalList 来代替 FP-树。它在快速查找项的前缀,并插入生成条件 FP-树方面有极大的优越性。Cir-OrthogonalList 结构中的每个项节点有 3 个域: item\_name, node\_link, node\_down。其中 item\_node 记录节点项在头表中的位置,即节点项序号, node\_link 指向按频繁 1 项集降序排列的每条事务中的下一项, node\_down 指向另外的链表中具有 item\_name 的节点,每条事务的最后一个节点的 node\_link 指向该事务的第一个节点,从而形成环路。保留了 FP-树的项头表结构,舍弃了原项头表中的支持度计数

项,但依旧有三个域:项的名称 item\_name,项的序号 item\_node,链接同名节点的 node\_down,项头表中的项按频繁 1 项集的顺序降序排列。

利用循环十字链表来存储事务数据库,构造步骤如下:扫描事务数据库  $DB$  一遍,产生频繁 1-项集,记下每个频繁项的支持度计数,各项按支持度计数降序排列,在项后标记项序号,存储在项头表  $L$  中;第二次扫描  $DB$ ,删除事务数据库  $DB$  的每一个事务序列中的不频繁项,余下用项序号代替,并按照序号从小到大排列,得到一个新序列,连接成链表,并为其设置项头表的同名节点指针 node\_down,序列的最后一项的 node\_link 指向序列的第一个节点。

例 1 设事务数据库中的事务如表 1 所示,最小支持度阈值 minsupport 为 2。

统计 1-项集及各项的支持度计数,得到 1-候选项集:  $\{I_1:4, I_2:5, I_3:3, I_4:2, I_5:2\}$ ,删去其中的不频繁项,并对其按支持度计数降序排列得到 1-频繁项集  $\{I_2:5, I_1:4, I_3:3, I_4:2, I_5:2\}$ ,在各项后标记项序号,存储在项头表  $L$  中。

表 1 一个事务数据库示例

TID	item_set	frequent itemset (ordered by count)
$T_1$	$I_1, I_2, I_5$	$I_2, I_1, I_5$
$T_2$	$I_2, I_4$	$I_2, I_4$
$T_3$	$I_2, I_3$	$I_2, I_3$
$T_4$	$I_1, I_2, I_4$	$I_2, I_1, I_4$
$T_5$	$I_1, I_3$	$I_1, I_3$
$T_6$	$I_1, I_2, I_3, I_5$	$I_2, I_1, I_3, I_5$

对例 1 中的每一个事务,删除其中的不频繁项后,余下的项用项序号代替,并按项序号从小到大排列,在排列后的事务序列构造循环十字链表。如图 1 所示。

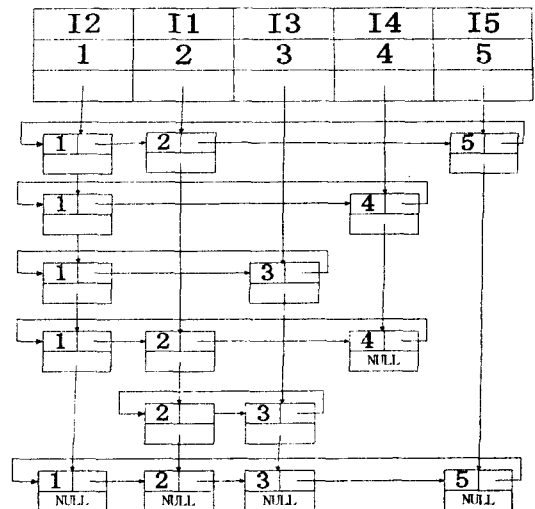


图 1 存储事务数据库信息的循环十字链表

### 3 挖掘算法

循环十字链表储存了所有有效的数据信息(各个项的支持度默认为 1),可由它构造排序条件频繁模式树 SConFP-tree 挖掘频繁项集。基本思想:遍历项头表的每一项,从最后一个项 item 开始,找到含有该项的所有单向循环链表,求其前缀,去除计数小于最小支持度的项,余下从小到大排列,形成条件模式基数据库 CondDB,构建项头表,并按照算法 1<sup>[6]</sup>构造排序条件频繁模式树。然后按照算法 2<sup>[6]</sup>对该树挖掘频繁模式。

算法 1: SConFP-tree Build。

输入:一个条件事务数据库 CondDB 及最小支持度阈值 minsupport;

输出:排序条件频繁模式树 SConFP-tree。

创建根节点  $T$ , 标记为“NULL”, 对于 condDB 中的每个事务 Trans(已排序) 中的频繁项列表为  $[p | P]$ , 其中  $p$  为第一个元素,  $P$  为其它元素。调用 insert\_tree( $p | P, T$ )。

Insert\_tree( $p | P, T$ ) 的实现方法为:

1) 如果  $T$  没有子节点则  $N.item\_no = p, N.count = 1, N$  的父节点链指向  $T$ ; 否则, 在  $T$  的子节点中查找  $p$  的插入位置, 如果找到与  $p$  相同的节点  $N$ , 则  $N.count$  加 1; 如果找不到相同节点, 则将创建一个新节点  $N$ , 设置  $N$  的各个域的值, 将  $N$  插入到比  $p$  大的第一个节点之前。

2) 如果新加入了节点  $N$ , 则将  $N$  插入到项头表中第  $p$  个元素的相同节点链表的末尾。

3) 如果  $P$  非空, 则递归调用 insert\_tree( $P, N$ )。

算法 2: 挖掘算法。

输入: 已构造的 SConFP-tree  $T$  及 minsupport;

输出: 所有以项 item 为后缀的频繁项集及其支持度( $\alpha = item$ )。

方法: 调用 Ming(SConFP-tree,  $\alpha$ )

Ming(SConFP-tree, item){

If 树  $T$  仅有单一路径  $P$

Then 对路径  $P$  中的任一模式组合  $\beta$

输出模式  $\beta \cup \alpha$  (再转换为对应的 item\_name), 模式支持度取  $\beta$  中节点的最小支持度

Else{

For( $i = n, i \geq 0; i--$ ) { //  $n$  为项头表的长度  
减 1

$\beta = i \cup \alpha$  且  $\sup(\beta) = \sup(i)$ ;

构造  $\beta$  的条件模式库和条件 FP 树  $T_\beta$ ;

If  $T_\beta \neq \text{NULL}$  then call Ming( $T_\beta, \beta$ );

|||

例 2 取  $I_5$  的前缀, 并生成排序条件频繁模式树, 如图 2 所示。

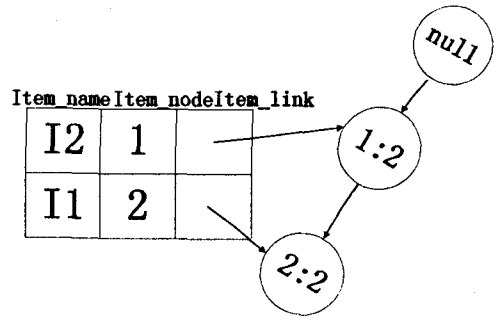


图 2  $I_5$  的条件频繁模式树

此时的排序条件频繁模式树仅有单一路径  $\{1\ 2\}$ , 生成子集  $\{1:2\}, \{2:2\}, \{1\ 2:2\}$ , 因为这是  $I_5$  的条件模式树, 所以要加上后缀 5, 并转换为 item\_name, 输出频繁项集及支持度为  $\{\{I_1, I_5:2\}, \{I_2, I_5:2\}, \{I_2, I_1, I_5:2\}\}$ 。同理可以求出其他的频繁项集。

删除  $I_5$  后的循环十字链表数据库如图 3 所示, 可见随着程序的运行, 循环十字链表数据库的规模在逐步减小, 内存消耗在减少。

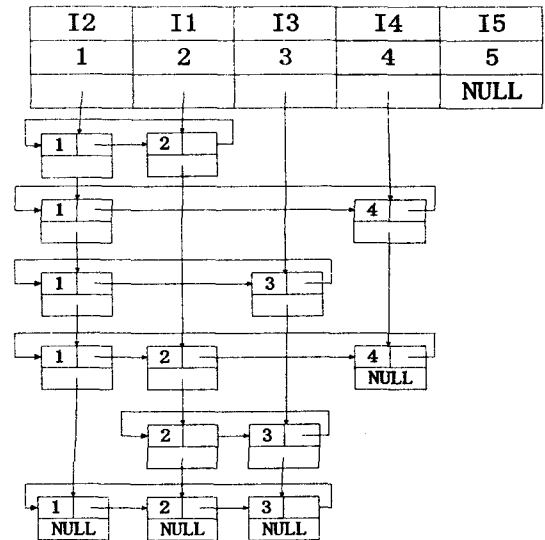


图 3 取  $I_5$  的前缀后的循环十字链表

### 4 试验结果及算法性能分析

为了更好地说明本算法的效率, 在同样的实验环境下将本算法与 FP-growth 算法进行比较。实验在 Pentium 4.0G CPU 的 PC 机上运行, 内存为 512MB, 运行环境为 Windows XP。程序代码均用 C++ 实现, 编译器是 Visual C++ 6.0。FP-growth 算法依据 Bart Goethals 提供的代码。实验数据集来源于 <http://firmics.helsinki.fi/data>, 采用 T40I10D100K.dat 的一部分作为测试数据集, 事务数据库大小为 500 k, 事务平均长度为 35。分别对两个算法运行时间和内存消耗情

况进行对比。

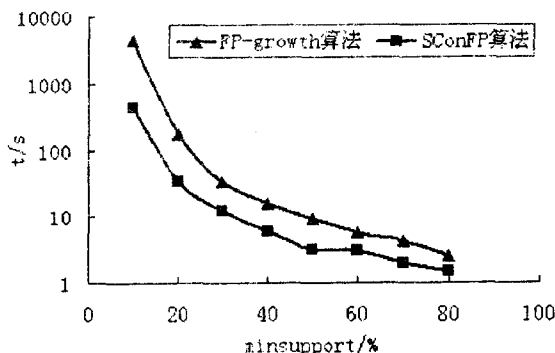


图 4 算法运行时间随支持度变化情况

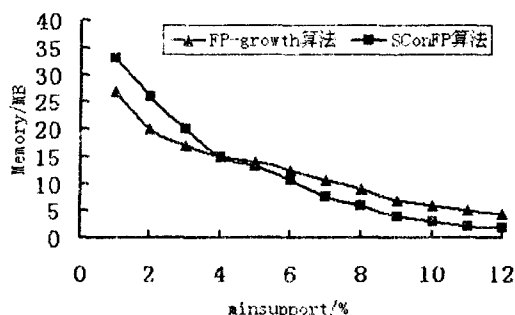


图 5 算法占用内存随支持度变化情况

图 4 和图 5 表明,基于循环十字链表结构的排序 FP 树挖掘算法在时间和空间上优于 FP-growth 算法,当支持度较小时算法时间效率是 FP-growth 算法的 1 倍以上,内存占用空间比 FP-growth 节省了一半以上,并且随着支持度的增加基于循环十字链表结构的排序 FP 树挖掘算法相比 FP-growth 的时空效率更高。这是因为:①采用带有循环的十字链表结构查找速度快,每个节点占用空间小,随着挖掘的进行,十字链表中的规模在逐步减小;②排序条件 FP 树的构造过程中,同一个节点的子节点是有序的,新节点  $N$  加入时,只要将  $N.item\_node$  与比其值小的节点比较,而 FP-tree 则需要比较所有节点<sup>[8]</sup>。遍历树时,节点可通过  $item\_node$  直接连接到头表的同名节点链。与 FP-tree 相比,有序的条件模式树的建树时间和遍历时间都较少。

笔者做过多次试验,该算法在稀疏数据库频繁项集的挖掘效果会更好。该方法存在一点瑕疵在于在设

定的支持度较小时,建立的链表数据库占用内存比 FP-tree 大,但是随着挖掘的进行,链表不断有节点被删除,规模在缩小,速度在加快,使得总时间和空间优于 FP-growth 算法。

## 5 结束语

频繁模式挖掘是数据挖掘中的一个重要课题。提出了一种新的基于循环十字链表的结构替代 FP-tree,这种结构存储的事务集数据库在程序运行中规模逐步减小,并利用排序条件频繁模式树挖掘频繁项集。理论分析和试验表明,与 FP-growth 算法相比,这种方法总的时空效率都较优。

## 参考文献:

- [1] Agrawal R, Srikant R. Fast algorithms for mining association rules in large databases[M]. Santiago: IBM Almaden Research Center, 1994:487-499.
- [2] Aly H H, Taha Y, Amr A A. Fast mining of association rules in large scale problems[C]// Abdel-wahwa H, Jeffay K. Proc. of the 6th IEEE Symp. on Computers and Communications (ISCC 2001). New York: IEEE Computer Society Press, 2001:107-113.
- [3] Tsai Cheng-Fa, Lin Yi-Chau, Chen Chi-Pin. A new fast algorithms for mining association rules in large databases[C]// Kamelae, Mellouluk, Borne P. Proc. of the 2002 IEEE Int'l Conf. on Systems, Man and Cybernetics (SMC 2002). [s. l.]: IEEE Computer Society Press, 2002:251-256.
- [4] Han Jia-wei, Pei Ji-an, Yin Yi-wen. Mining frequent patterns without candidate generation[C]//Proceedings of ACM SIGMOD'00 International Conference on Management of Data. New York: ACM Press, 2000:1-12.
- [5] 冯洁,陶宏才. 典型关联规则挖掘算法的分析与比较[J]. 计算机技术与发展, 2007, 17(3):121-124
- [6] 秦亮曦,李谦,史忠植. 基于排序 FP-树的频繁模式高效挖掘算法[J]. 计算机科学, 2005(4):31-33.
- [7] Han Jiawei, Kamber A. 数据挖掘:概念与技术[M]. 范明,孟小峰等译. 北京:机械工业出版社, 2008:157-159.
- [8] 秦亮曦,史忠植. SFP-Max——基于排序 FP-树的最大频繁模式挖掘算法[J]. 计算机研究与发展, 2005(2):17-22.

(上接第 72 页)

动目标检测[J]. 计算机技术与发展, 2008, 18(1):136-139.

- [8] 韩颖婕,张海,李琳怡. 基于混合高斯背景建模的阴影抑制算法[C]//第十四届全国图象图形学学术会议论文集.

北京:清华大学出版社, 2008:313-316.

- [9] Liu Hong, Li Jintao, Liu Qun, et al. shadow elimination in traffic video segmentation[C]//IAPR Conference on Machine Vision Applications 2007. Tokyo, Japan:[s. n.], 2007.