基于先验位运算的频繁项集挖掘*

张　岳[1]，王洪国[1]，邵增珍[1]，赵建秀[2]

(1. 山东师范大学 信息科学与工程学院，济南 250014；2. 山东省分布式计算机软件新技术重点实验室，济南 250014)

摘　要： 为提高频繁项集的产生效率，提出一种在垂直数据表示下，基于先验位运算的频繁项集挖掘算法(A-FIMBII)。该算法建立从项集合到事务的索引，利用先验性质减少候选集的产生，通过位运算计算支持度。与 Apriori、Eclat 算法进行了比较，实验表明，A-FIMBII 具有更高的效率。

关键词： 频繁项集；Eclat 算法；先验；索引表

中图分类号： TP18；TP301　　**文献标志码：** A　　**文章编号：** 1001-3695(2013)09-2610-03

doi:10.3969/j.issn.1001-3695.2013.09.012

Frequent itemsets mining based on Apriori-bit

ZHANG Yue[1], WANG Hong-guo[1], SHAO Zeng-zhen[1], ZHAO Jian-xiu[2]

(1. *College of Information Science & Engineering, Shandong Normal University, Jinan* 250014, *China*; 2. *Shandong Provincial Key Laboratory for Distributed Computer Software Novel Technology, Jinan* 250014, *China*)

Abstract: This paper proposed an Apriori frequent itemset mining based on bittable and inverted index(A-FIMBII). A-FIMBII built the index from the collection of items to the transaction, and used priori nature to reduce the generation of candidate sets, calculating the support counts of two items through the bit operations. It compared to Apriori, ECLAT in four datasets. The experiment results show that A-FIMBII has a higher efficiency.

Key words: frequent itemset; Eclat algorithm; Apriori; inverted index

0　引言

关联规则挖掘是数据挖掘的关键问题之一，而关联规则挖掘就是要发现所有满足最小支持度和最小置信度的关联规则。关联规则挖掘一般分为两步：a)频繁项集的挖掘；b)由频繁项集挖掘隐含的关联规则。其中频繁项集的挖掘是非常耗时的，也是挖掘关联规则的关键步骤，它决定了关联规则挖掘的总体性能是否满足用户的需求。关联规则挖掘有两个基本性质：

a)任何频繁项集的子集一定也是频繁项集。

b)任何非频繁项集的超集一定也是非频繁项集。

最经典的关联规则算法是 Agrawal 等人[1]提出的 Apriori 算法，它采用层次迭代扫描事务数据库的方法挖掘关联规则，可以挖掘出所有满足阈值的关联规则，但是它存在两个严重缺陷：a)重复多次扫描数据库；b)产生的候选集数据过大。FP-growth 算法[2]虽然只需要扫描数据库两次，但是它也有不足，动态地生成和释放数以万计的 FP-数，将耗费大量的时间和空间。除此之外，还有 Park 等人[3]提出的 DHP 算法、Brin 等人[4]提出的 DIC 算法等，这些算法在对大型数据库进行关联规则挖掘过程中，通常无法将驻留磁盘的数据一次性读入内存，造成大量的磁盘 I/O 操作和严重的 CPU 开销。

以上算法采用的是传统的水平数据格式，即从“事务→项集合”格式的事务集挖掘频繁项集，但随着数据规模越来越大，应用环境越来越复杂，传统改进算法已经不适应时代的发展要求。为此，提出了一种垂直数据格式，即“项集合→事务”的格式。Zaki[5]较早提出的 Eclat 算法仅需要扫描一次事务数据库建立项目到事务的垂直数据结构，然后通过求交集的方法计算各个项目集合的支持度计数，生成频繁项集。但是 Eclat 算法在对庞大的事务数据库求交集时效率低下，如何快速求交集就成为解决 Eclat 算法瓶颈的关键问题。

为此，Shenoy 等人[6]提出了 VIPER 算法，Burdick 等人[7]提出了 MAFIA 算法，都对 Eclat 垂直数据格式的算法进行了改进。国内也有部分学者对垂直数据格式的算法进行了研究改进。熊忠阳等人[8]通过散列表与布尔矩阵相结合，提出了一种基于散列的布尔矩阵的 Eclat 改进算法，提高了求交集的速度。宋长新[9]和张玉芳[10]等人分别在文献中对 Eclat 算法进行了研究，并对 Eclat 进行了改进，使改进后的算法对数据集较小的稠密数据库具有较好的挖掘性能。傅向华等人[11]基于 Eclat 算法提出了一种 DF-FIMBII 算法，在数据规模不是很大且支持度较小的情况下，DF-FIMBII 具有较好的时间优越性，但是当数据规模非常大时，基于位运算的方法性能下降。

本文提出了一种改进的基于垂直数据格式的关联规则算法，实验表明，改进算法能获得较好的时间优越性。

1　Eclat 算法

Eclat 算法采用垂直数据格式表示事务数据集，只需扫描一次数据库，且通过交集运算计算频繁项集的支持度。

收稿日期： 2013-01-13；**修回日期：** 2013-03-04　　**基金项目：** 山东省自然科学基金资助项目(ZR2011FQ029,ZR2011FL026)；山东省科技发展计划资助项目(2011YD01099,2011YD01100)；山东省高等学校科技计划资助项目(J11LG32)

作者简介： 张岳(1988-)，男，山东济南人，硕士研究生，主要研究方向为数据挖掘等(fanslinjunjie@126.com)；王洪国(1966-)，男，教授，博导，主要研究方向为电子政务、智能算法；邵增珍(1976-)，副教授，硕士；赵建秀，硕士研究生，主要研究方向为数据挖掘等.

1.1　垂直数据表示

计算支持度需要访问数据库,采用垂直数据表示,从概念上讲,这样的数据库可以用一个二进制的二维矩阵来表示,矩阵的每一行代表数据库的项目,每一列代表事务。

定义 1　索引项集 Tidset。

设事务集 $T=\{T_1,T_2,T_3,\cdots,T_{|T|}\}$,项目集 $I=\{I_1,I_2,I_3,\cdots,I_{|T|}\}$,数据库中的每一条记录由一个项目及其所出现过的所有事务记录的列表(即 Tidset 表)构成。这样使得任何一个频繁项集的支持度都可以通过 Tidset 交集运算求得。

例如,在这种数据表示方法中,假设包含项目 I_1 的事务有 $\{T_1,T_2,T_4,T_7\}$,则 $\text{Tidset}(I_1)=\{T_1,T_2,T_4,T_7\}$。

1.2　支持度计算

Eclat 算法的支持度可以通过两项目的索引项集的交集得到,关联规则 $x\rightarrow y$ 的支持度定义为

$$\text{support}(x,y)=\frac{|\text{Tidset}(x)\cap\text{Tidset}(y)|}{|T|}$$

关联规则 $x\rightarrow y$ 的置信度定义为

$$\text{confidence}(x,y)=\frac{(\text{Tidset}(x)\cap\text{Tidset}(y))}{\text{Tidset}(x)}$$

因此,不难看出,在 Eclat 算法中,在算法性能上起决定性作用的是交集运算,如果要得到一个高效的算法性能,必须有一个高效的求交集的方法。

1.3　Eclat 算法的缺点

Eclat 与 Apriori 算法最大的不同之处是,对于 Eclat 算法而言,候选集的生成过程和支持度的计算过程没有明显的界限。在求交集的过程中,若候选集的支持度小于给定的最小支持度 Minsupp,则将其删除。

由算法的过程可知,算法并没有充分利用 Apriori 算法的先验性质(性质 1 和 2),因此 Eclat 算法产生的候选集数目远远大于 Apriori 算法,消耗系统大量内存资源。

2　基于先验位运算的频繁项集挖掘

首先为了获得高效的求交集的方法,本文采用二进制数据存储索引表的结构,基于位运算进行项目的求交,然后统计结果二进制数组中 1 的个数,即可求得支持度。根据 Eclat 不能充分利用 Apriori 先验的性质,本文提出一种基于先验位运算的频繁项集挖掘算法 A-FIMBII。

2.1　建立索引表

扫描完一遍事务数据库,可以建立每个项目的索引项集,可以称之为索引表。这时,可以申请一个无符号整型二维数组 $a[m][n]$ 来存储索引表,m 为项目的个数,n 为事务集中事务的总数 $|T|$ 除以长整型变量所包含的位数,如在 32 位机器上,长整型数据分配了 32 位,则数组大小为 $\lceil |T|/32\rceil$。例如,设 $|T|=32$,$\text{Tidset}(I_1)=\{T_1,T_2,T_3,T_4\}$,$\text{Tidset}(I_2)=\{T_1,T_3,T_4,T_5\}$。若事务 k 出现,则内存中第 K 位为 1,否则为 0,则 $\text{Tidset}(I_1)$ 在计算机内存中存储为如图 1 形式。

1	1	1	1	0	0	…	…	0	0	0
第 0 位	第 1 位	第 2 位	第 3 位							第 |T| 位

图 1　$\text{Tidset}(I_1)$

如果将图 1 内存中的二进制数转换为十进制数字即为 240,则可以设长整型数组 $a[0][0]=240$。$\text{Tidset}(I_2)$ 在计算机内存中存储为如图 2 所示。

1	0	1	1	1	0	…	…	0	0	0
第 0 位	第 1 位	第 2 位	第 3 位	第 4 位						第 |T| 位

图 2　$\text{Tidset}(I_2)$

如果将图 2 内存中的二进制数转换为十进制数字即为 184,则可以设长整型数组 $a[1][0]=184$。

本文按上述方法,将每个项目的索引项集转换为十进制整数存储到长整型数组中。接下来就可以通过位运算来求支持度。例如,I_1 和 I_2 的支持度,可以通过 $a[0][0]\&a[1][0]$ 求得。具体过程如图 3 所示。

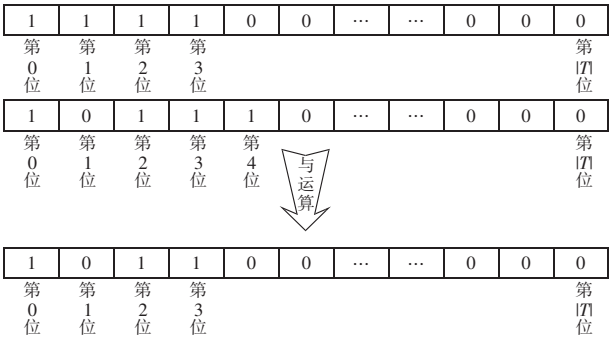

图 3　$\text{Tidset}(I_1)\cap\text{Tidset}(I_2)$

只要求得图 3 中 $\text{Tidset}(I_1)$ 和 $\text{Tidset}(I_2)$ 经过位运算之后二进制数中 1 的个数即可求得 $I_1\rightarrow I_2$ 的支持度。

2.2　支持度计算

为了高效统计整数二进制中 1 的个数,本文采用分治的思想,先计算每对相邻的 2 位中 1 的数目,然后分别计算相邻的 4 位、8 位、16 位中 1 的个数。例如,对于无符号长整数 m,可以利用如下运算语句高效统计 m 中 1 的个数:

```
m = (m & 0x55555555) + ((m >>1) & 0x55555555);
m = (m& 0x33333333) + ((m >>2) & 0x33333333);
m = (m& 0x0f0f0f0f) + ((m >>4) & 0x0f0f0f0f);
m = (m& 0x00ff00ff) + ((m >>8) & 0x00ff00ff);
m = (m& 0x0000ffff) + ((m >>16) & 0x0000ffff);
```

2.3　算法描述

Eclat 算法没有充分利用 Apriori 先验性质,它会产生一些额外的候选集并耗费大量的时间计算其支持度,在最坏的情况下,这种额外极其可观。因此本文对 Eclat 算法进行了改进,从左到右挖掘频繁项集,挖掘 $k-1$ 阶项集时把 $k-1$ 阶频繁项集的信息记录下来,在挖掘 k 阶项集的时候利用频繁 $k-1$ 阶项集对候选集进行剪枝。

首先扫描一遍事务数据库,建立项目的索引表,根据支持度计数方法找出一阶频繁项集,然后利用 1 阶频繁项集对 2 阶候选集进行剪枝,产生 2 阶频繁候选集;根据位运算求得支持度,若支持度大于给定的最小支持度阈值则将其存储,产生 2 阶频繁项集。对于 K 阶频繁项集,若 K 阶候选集的 $k-1$ 子集不全在 $k-1$ 频繁项集中,则将其删除,否则产生 K 阶频繁候选集,然后根据位运算求得支持度。若支持度大于给定最小支持度则存储,产生 K 阶频繁项集。算法的伪代码如下:

```
输入:事务集 T,支持度 min_sup,置信度 min_cof。
输出:关联规则 x→y(support(x,y)≥min_sup,x⊂T,y⊂T)。
A-FIMBII(T)
{
  L = L1 = {Ii,sup(Ii) > min_sup}     //L 初始值为 1 阶频繁项集
  k = 2
  for(int i =0;i < Lk-1. length -1;(i ++,k ++))
```

```
for(int j = i + 1;j < L_{k-1}. length,j ++ )
{
  C_k = I_i ∞ I_j
  foreach itemsset L_i ∈ L_{k-1}//扫描 L_{k-1}中的元素
  foreach candidate c ∈ C_k//扫描 C_k 的元素
  if I_l is the subset of C_k//扫描 L_i 是否为 C_k 的子集
    then      //如果是计数加 1
  c. count ++ ;
  C_k = {c ∈ C_k | c. count = k} ;
  count = bitwise_and_operation(I_i ,I_j) ;//通过位运算求得支持度
f(count >= minsup * |T|)
  {
    L = L_{k-1} ∪ L
}}}
```

3 实验及分析

3.1 实验环境

为验证本文所提 A-FIMBII 算法的性能，将其与经典的 Apriori、Eclat 算法进行比较。实验环境为 Intel PIV2 .8 GHz 双核，内存 2 GB，软件环境为 Microsoft Visual Studio 2010 中开发。操作系统为 Windows XP Profession SP3。

3.2 实验数据集

实验数据集来自关联规则挖掘研究平台——频繁项集实验库 http://fimi. ua. ac. be/，选择事务数据集大小不同的四个数据集，分别为 chess、mushroom、pumsb_star、T40I10D100K，如表 1 所示。

表 1 数据集的稠密程度

数据集	事务数	项目数	稠密程度
chess	3196	75	42.6
mushroom	8124	119	68.3
pumsb_star	49046	7116	6.9
T40I10D100K	100000	1000	10

3.3 实验结果

由于各数据集的稠密程度不同(所有事务与项目之比)，因此不同的数据集给定不同的支持度，然后分别测试 Apriori、Eclat、A-FIMBII 挖掘频繁项集所用的时间，在四种数据集的执行时间结果分别如图 4 ~7 所示。

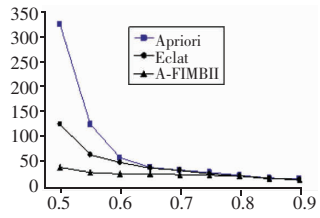

图 4 三种算法在 chess 数据集上的实验结果

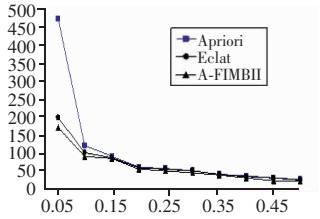

图 5 三种算法在 mushroom 数据集上的实验结果

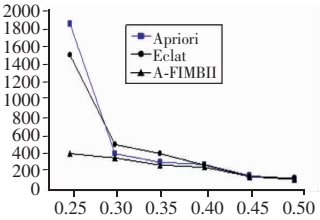

图 6 三种算法在 pumsb_star 数据集上的实验结果

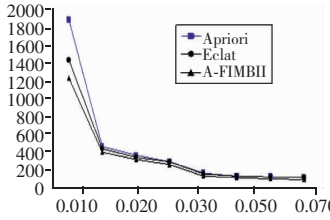

图 7 三种算法在 T40I10D100K 数据集上的实验结果

3.4 实验结果分析

从实验结果可以看出，Eclat 算法的执行效率在支持度不是很小的情况下，执行效率要优于 Apriori 算法，基于先验位运算的 A-FIMBII 算法在不同稠密程度的数据集中生成频繁项集的时间开销明显优于 Apriori、Eclat 算法。但是，当数据集特别大且支持度很小时，A-FIMBII 算法的性能有所下降。分析其主要原因有以下几个方面：

a)基于先验位运算的 A-FIMBII 只需要扫描一次数据库，且利用位运算求解支持度，执行速度较快，相对于 Apriori 重复扫描数据库，Eclat 求解支持度慢的缺点，A-FIMBII 总体执行效率比较优越。

b)A-FIMBII 利用 $K-1$ 阶频繁项集对 K 候选集进行剪枝，很大程度上避免了产生过多的候选项集，并且保存了 K 阶频繁项集中每个频繁项集的二进制数组，可直接对两个 K 阶频繁项集的二进制数组求“与”运算。

c)在数据集规模非常大时，为每个项目分配的空间也相应变大，将消耗大量的内存空间。

从图 4 ~7 中可以看出，当支持度较大时，Apriori 算法的执行时间比 Eclat 算法稍快。这是因为当支持度较大时，频繁项集数据会减小，并且每个频繁项集共同含有的项目数较大，使得 Eclat 计算两个长候选集的交集需要大量的时间开销。

4 结束语

本文提出一种在垂直数据结构下，基于先验位运算的频繁项集挖掘算法。该算法利用二进制数据存储事务到项目的索引。通过位运算计算两个项目的支持度计数，并利用先验性质减少候选项集的产生。在不同规模的 chess、mushroom、pumsb_star、T40I10D100K 数据集上，将 Apriori、Eclat、A-FIMBII 算法进行了比较。实验结果表明，A-FIMBII 具有较好的执行效率。

参考文献：

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

[2] HAN Jia-wei, PEI Jian, YIN Yi-wen. Mining frequent patterns without candidate generation[C]//Proc of ACM SIGMOD International Conference on Management of Data. New York: ACM Press, 2000: 1-12.

[3] PARK J S, CHEN M S, YU P S. An effective hash based algorithm for mining association rules[C]//Proc of ACM SIGMOD International Conference on Management of Data. New York: ACM Press, 1995: 175-186.

[4] BRIN S, MOTWAI R, ULLMAN J D, *et al*. Dynamic itemset counting and implication rules for market basket data[C]//Proc of ACM SIGMOD Conference on Management of Data. 1997: 265-276.

[5] ZAKI M J. Scalable algorithms for association mining[J]. IEEE Trans on Knowledge and Data Engineering, 2000, 12(3): 372-390.

[6] SHENOY P, HARITSA J R, SUDARSHAN S, *et al*. Turbo-charging vertical mining of large databases[C]//Proc of International Conference on Management of Data. 2000: 22-23.

[7] BURDICK D, CALIMLIM M, GEHRKE J M. AFIA: a maximal frequent itemset algorithm for transactional databases[C]//Proc of International Conference on Data Engineering. 2001: 443-452.

[8] 熊忠阳，陈培恩，张玉芳. 基于散列布尔矩阵的关联规则 ECLAT 改进算法[J]. 计算机应用研究，2010，27(4)：1323-1325.

[9] 宋长新，马克. 改进的 Eclat 数据挖掘算法的研究[J]. 微计算机信息，2008，24(24)：92-94.

[10] ZHANG Yu-fang, XIONG Zhong-yang, CHEN Yan, *et al*. Immune algorithms based on data processing in intrusion detection[J]. Journal of Computational Information Systems, 2008, 4(1): 293-300.

[11] 傅向华，陈冬剑，王志强，等. 基于倒排索引位运算的深度优先频繁项集挖掘[J]. 小型微型计算机系统，2012，33(8)：1747-1751.