第 35 卷 第 2 期 计 算 机 工 程 2009 年 1 月
Vol.35 No.2 Computer Engineering January 2009
·软件技术与数据库· 文章编号：1000—3428(2009)02—0037—02 文献标识码：A 中图分类号：TP301.6

基于十字链表的 Apriori 改进算法

黄建明，赵文静，王星星
（西安建筑科技大学信息与控制工程学院，西安 710055）

摘 要：针对 Apriori 算法中存在的不足，提出一种把事务数据库映射到十字链表中的改进算法。该算法可以减少连接数据库的次数及事务记录的扫描次数。Apriori 算法与改进算法的性能对比分析表明，改进算法能有效提高执行效率。
关键词：数据挖掘；关联规则；事务数据库；Apriori 算法；十字链表

Improved Apriori Algorithm Based on Across Linker

HUANG Jian-ming, ZHAO Wen-jing, WANG Xing-xing
(School of Information and Control Engineering, Xi'an University of Architecture and Technology, Xi'an 710055)

【Abstract】 By studying Apriori algorithm, this paper points out its disadvantages, and proposes the transaction database mapping for an across linker description, which reduces the time cost of the database linking and the scanning time of the transaction record. It gives an improved algorithm. By analyzing the efficiency of the Apriori algorithm and the improved algorithm, it shows that the improved algorithm is better than Apriori algorithm in execution efficiency.
【Key words】 data mining; association rules; transaction database; Apriori algorithm; across linker

1 概述

数据挖掘可以从数据库中发现有价值的、隐藏的知识，关联规则挖掘[1]是数据挖掘中一个重要的研究方向，其主要目标是发现事务数据库中项集之间有趣的关系，关联规则挖掘过程分为 2 步[2]：(1)根据一种算法找出所有的频繁项集；(2)根据生成的频繁集产生关联规则。其中，第(1)步是核心，决定挖掘关联规则的总体性能，在确定频繁集后，就可以得出相应的关联规则，因此，算法的研究主要针对第(1)步。

关联规则的相关概念[3]与性质如下：

定义 1(关联规则) 设$I=\{I_1,I_2,\cdots,I_m\}$是所有项的集合，其中，$I_k(k=1,2,\cdots,m)$称为项。项的集合称为项集，包含k个项的项集称为k-项集。一个事务T是一个项集，它是I的一个子集，每个事务均与一个唯一标识符Tid相联系。不同的事务一起组成了事务集D，它构成了关联规则发现的事务数据库。关联规则是形如$X \Rightarrow Y$的蕴涵式，其中，$X \subset I, Y \subset I$，且$X \cap Y = \varnothing$。

定义 2(支持度) 关联规则 $X \Rightarrow Y$ 的支持度定义为

$$support(X \Rightarrow Y) = support(X \cup Y) = \frac{|\{T \mid T \in D and (X \cup Y) \subseteq T\}|}{|D|}$$

其中，“| |”表示集合中的元素个数。

定义 3(频繁项集) 如果项集满足最小支持度，即如果项集的出现频率大于或等于min_sup与事务数据库D中事务总数的乘积，则称该项集为频繁项集，简称为频繁集，频繁k-项集的集合记为L_k。

性质 1 频繁项集的所有非空子集也都是频繁的。

2 Apriori 算法

Apriori算法是最经典的关联规则挖掘算法，是频繁项集发现的核心算法，它是一种逐层搜索的迭代方法，利用已知的频繁(k-1)-项集L_{k-1}生成候选k-项集C_k，再扫描一次数据库来判断候选频繁项集是否是频繁项集，直到不再产生候选频繁项集，算法结束。算法描述如下：

输入 事务数据库 D，最小支持度阈值 min_sup
输出 D 中的频繁项集 L

```
(1)L1=find_frequent_1–itemsets(D);
(2)for(k=2;Lk-1≠Ø;k++)  {
(3)Ck=apriori_gen(Lk-1, min_sup);
(4)for each transaction t∈D { //scan D for counts
(5)Ct =subset(Ck,t);   //get the subsets of t that are candidates
(6)for each candidate c∈Ct
(7) c.count++;}
(8) Lk={ c∈Ck| c.count>= min_sup}   }
(9)return   L=∪k Lk;
```

Apriori算法的第(1)步是找出频繁 1-项集的集合L_1，在第(2)步~第(8)步，L_{k-1}用于产生候选C_k，以找出L_k。其中，第(3)步的apriori_gen函数用于产生候选项集，然后使用Apriori性质删除那些具有非频繁子集的候选项集；产生了所有的候选项集后，第(4)步扫描数据库；第(5)步使用subset函数找出每个事务中所有的k-项子集，并判断这些子集是否与候选项集匹配；第(6)步、第(7)步则对所有模式匹配成功的候选项集累加计数。

通过对 Apriori算法的分析可知，算法存在 2 点不足：(1)每生成L_k时，需要扫描数据库一次，即算法须多次扫描数据库；(2)在每次扫描数据库的过程中，候选项集中有些项目或事务不需要扫描，但算法仍扫描了这些项目和事务。针对上述情况，本文提出了一种基于十字链表的改进算法。

3 Apriori 算法的改进

为了使算法只扫描一次数据库，节约存储设备 I/O 时间，本文将事务数据库中的信息用十字链表表示，把对数据库的

作者简介：黄建明(1983－)，男，硕士研究生，主研方向：数据挖掘，数据库；赵文静，教授；王星星，硕士研究生
收稿日期：2008-07-10 **E-mail：**riluo410@163.com

扫描转变为对内存中十字链表的扫描。

3.1 事务数据库的十字链表表示方法

设事务$T=\{t_i|1\le i\le m\}$，其中，m为事务T的事务长度；$t_i\in I$。十字链表的描述可以形式化地表示为：$List=(U,A,V,f)$，其中，$List$为十字链表名；U为事务的集合形式，$U=\{T_i|1\le i\le n\}$，T_i代表某个具体的事务，n为事务的总数；A表示每个事务的组合方式，它由事务T与一个唯一标识符Tid组成；$V=\{<t_i, number>|t_i\in T, i=1,2,\cdots,m, number=m-i+1\}$，表示中间节点集合，其中，元素$<t_i, number>$由事务$T$的项目$t_i$与事务$T$的长度$number$共同组成；$f$表示一种映射关系，定义为 $f: T\to V$。

对于一个经过预处理的数据库模式$<Tid, Item>$，规定事务中的项按字典序存放，把此事务数据库映射为十字链表，则十字链表由事务标识头表、项目头表与链表中间节点组成，其中，每个事务标识节点包含 2 个域：事务名称和链域 next 指针(指向该事务的第 1 个项)；每个项目节点包含 2 个域：项目名称和链域 down 指针；链表中间节点由 4 个域组成，如图 1 所示，包括项目名 *Item*、该项目到事务末尾项目的长度 *number*、指向下一个事务中此项的 down 指针及指向同一事务中下一项的 right 指针。

Item	*number*
down	right

图 1　十字链表中间节点表示

例如：有一事务数据库 D，其中有 6 条事务记录，项目名用 a,b,c 等表示，如表 1 所示。把此事务数据库中的信息用十字链表表示，如图 2 所示。

表 1　样例数据表

Tid	*Items*
1	a, b, e
2	b, d
3	a, c
4	b, e
5	a, d
6	b, c, d

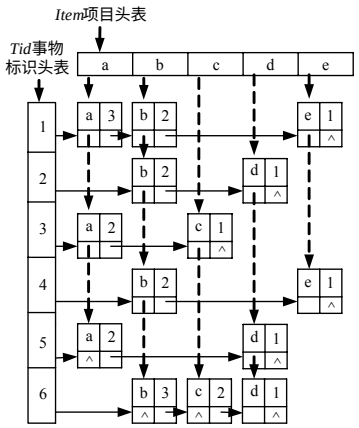

图 2　事务数据库的十字链表表示

3.2 改进的 Apriori 算法

本文基于事务数据库D的十字链表的表示，把在事务数据库中挖掘频繁项集转化为对十字链表的操作，首先假设：c_i是候选k-项集C_k中的项集，$c_i[j](j=0,1,\cdots,k-1)$表示c_i的 第j项(例如，$c_2[3]$表示c_2的第 3 项)。

性质 2 若事务T中的项与$c_i[1]$匹配成功，且此项的$number$值小于k，则该事务T对产生频繁项集是无用的。

很明显，在生成频繁k-项集时，需要扫描十字链表计算各个候选k-项集的支持数，若事务T中与$c_i[1]$进行匹配的项的$number$值小于k，则事务T不可能包含以$c_i[1]$为首的k-项频繁集的所有项，因此，不需要扫描此事务。

性质 3 候选k-项集中的任一项集c_i与十字链表中的事务匹配时，假设$c_i[j]$与十字链表中某事务T中的项t已匹配成功，$c_i[j+1]$与t的下一项匹配时，若此项按字典序在$c_i[j+1]$之后，停止匹配，此事务对c_i的支持数没有影响。

很明显，如果十字链表中事务T中的项已经匹配到字典序在$c_i[j+1]$之后的项，说明在十字链表的事务T中不包含$c_i[j+1]$，不能匹配成功，即对c_i的支持数没有影响。

改进算法的步骤如下：

(1)寻找频繁 1-项集。扫描十字链表，计算候选 1-项集中各项的支持数；每项的支持数与给定的min_sup进行比较，如果大于min_sup，此项并入频繁 1-项集L_1中。

(2)寻找频繁k-项集L_k：

1)连接L_{k-1}产生候选k-项集，根据Apriori性质对产生的候选k-项集剪枝后生成候选项集C_k。

2)在候选项集C_k中，沿着十字链表中$Item$项目头表搜索，将$c_1[1]$与$Item$项目头表中节点的项目名称匹配，直到匹配成功为止。

3)沿十字链表中 *Item* 项目头表中匹配成功节点的 down 指针向下搜索，判断 down 指针指向的此中间节点的 *number* 域是否小于 k，如果 *number* 值小于 k，此节点继续沿 down 指针向下搜索，直到它的 down 指针指向空，否则，此节点沿 right 域开始向右匹配 k-项集。

4)在匹配过程中，如果项集c_i与事务T的前j项($j=0,1,\cdots,k-1$)已经匹配成功，在进行$j+1$ 项匹配时，如果按字典序$c_i[j+1]$位于和事务T进行匹配的项之后，根据性质 3，停止匹配，否则，k-项集c_i在事务T中匹配成功，说明该k-项集c_i在事务T中存在，则支持数加 1。

5)判断k-项集的支持数是否小于min_sup，如果不小于，就将该k-项集添加到L_k中。

6)取下一个$c_i[1]$，进行步骤 2)~步骤 5)，直到C_k中的所有k-项集全部扫描完毕为止。

(3)输出频繁k-项集L_k。

算法描述如下：

输入 事务数据库 D 映射后的十字链表 $List$，最小支持度阈值 min_sup

输出 D 中的频繁项集 L

```
(1)L1=find_frequent_1–itemsets(List)；
(2)for(k=2;Lk-1≠Ø;k++)  {
(3)Ck=apriori_gen(Lk-1，min_sup);
(4)for each candidate c∈Ck
(5)for each transaction t &&firstitem k1∈t
(6)if(match(c,t)) //候选k-项集ci在事务T中进行匹配，匹配成  //功返回True
(7)c.count++; }
(8)Lk={ c∈Ck| c.count>= min_sup} }
(9)return  L= ∪k Lk;
```

4 算法性能分析

从开始到剪枝，改进算法与原算法的过程一样。在统计支持数时，Apriori算法检查每一个事务T中是否包含C_k中的每一个k-项集，需要将C_k中的每个k-项集与事务项t匹配，

(下转第 41 页)