一种改进的新 Apriori 算法

李晓虹[1] 尚 晋[2]

(重庆师范大学数学与计算科学学院 重庆 400047)[1] (重庆电子职业技术学院计算机二系 重庆 400021)[2]

摘 要 本文分析了 Apriori 算法的时间复杂性和空间复杂性,利用十字链表来等价代替事务数据库的数组表示,从而使得:一方面,连接操作的次数减少一半,缩短了算法的运行时间;另一方面,挖掘过程中不必保留候选频繁项目集,节省了大量空间开销。实验表明,改进后新算法的性能具有可比性。

关键词 Apriori 算法,十字链表,事务数据库

An Improved New Apriori Algorithm

LI Xiao-Hong[1] SHANG Jin[2]

(College of Mathematics and Computer Science, Chongqing Normal University, Chongqing 400047)[1]

(The Second Department of Computer, Chongqing Electronic Profession College, Chongqing 400021)[2]

Abstract Based on the analysis of Apriori algorithm from the perspectives of time complexity and memory complexity, we use an across linker to substitute the array description of the transactions database. Theroefore, on the one hand, the time cost is reduced by shorten the length of the objects linked. On the other hand, large memory cost is saved due to the un-saving of candidates of frequent itemset. The experiments show that the improved new algorithm's performances are comparable.

Keywords Apriori algorithm, across linker, transaction DB

1 前言

关联规则挖掘是数据挖掘的一个方向,它反映了大量数据中项目集之间的关联或者相关联系。关联规则挖掘过程分为两步:一是采用适当的算法从事务数据库中挖掘出高于初始最小支持数的频繁项目集;二是由频繁项目集生成有价值的规则。其中,频繁项目集的获得是关联规则挖掘中的关键,其计算复杂度决定了整个挖掘算法的时间复杂度。1993 年,Agrawal 等首次提出关联规则挖掘的 Apriori 算法[1],次年又提出 AprioriTid 算法和 AprioriHybrid 算法[2],其后的研究者也提出了很多以 Apriori 算法为核心的改进算法[3,4]。通常这些改进的 Apriori 算法中,或多或少地存在如下两个问题:(1)算法必须花大量的时间处理候选项目集;(2)算法必须多次扫描事务数据库,对候选项目集进行模式匹配。这两个问题是当前关联规则挖掘的热点和难点,也是约束系统性能的瓶颈。

针对上述情况,我们首先介绍了 Apriori 算法,分析了基于树、数组、垂直二进制位和水平二进制位串等的事务数据库表示模式[5],其次用十字链表来表示事务数据库,使得数据挖掘过程中不必保留候选频繁项目集,从而使这种基于十字链表的算法得到性能上的改进。文章最后分析了该算法的时间复杂性和空间复杂性,并与 Apriori 算法和 AprioriTid 算法进行了比较。

2 Apriori 算法

Apriori 算法的核心思想是采用逐层递推的方法,首先扫描数据库,产生频繁 1 项目集 L_1;再由 apriori_gen 函数利用 L_{k-1} 中的成员连接、剪枝后,产生候选频繁项目集 C_k,通过扫描事务数据库计算每个候选频繁项目集的支持数,并与用户指定的最小支持数比较,大于最小支持数的项目集并入频繁 k 项目集 L_k 中;算法直到不再产生候选频繁项目集结束。最后合并全部频繁项目集。其算法描述如下:

```
输入:事务数据库 D,最小支持数 Minsup
输出:D 中所有的频繁项目集
1)  L1={frequent 1-itemset};
2)  for (k=2;Lk-1≠∅;k++) do begin
3)    Ck= apriori-gen(Lk-1);//新的候选集
4)    for all transactions t∈D do begin
5)      Ct=subset(Ck,t);// 事务 t 中包含的候选集
6)      for all candidates c∈Ct do
7)        c. count++;
8)    end
9)    Lk={c∈Ct|c. count>Minsup};
10) end
11) Answer=∪Lk;
```

设 N 表示事务数据库的交易数,T 表示最大交易长度,则第一遍数据库扫描时间为 $O(T\times N)$,在每一步中,连接时间开销是 $O(|L_{k-1}|\times|L_{k-1}|)$,剪枝时间是 $O(|C_k|)$,扫描计数时间是 $O(N\times|C_k|)$。所以 Apriori 算法总的时间开销为 $O(T\times N)+\sum_{k\geq 2}(O(|L_{k-1}|\times|L_{k-1}|)+O(|C_k|)+O(N\times|C_k|))$,由于频繁 2 项目集是在 L_1 上进行连接和剪枝,频繁 3 项目集是在 L_2 上进行连接和剪枝,且 $|L_i|>|L_j|$,$2\leqslant i,j\leqslant k$,$i<j$,因此 Apriori 算法的时间开销主要集中在频繁 2 项目集和频繁 3 项目集的生成以及候选频繁项目集 C_2,C_3 的计数过程中。另外,Apriori 算法的空间复杂性为:$O(|L_1|)+\sum_{k\geq 2}(O(|C_k|)+O(|L_k|))$。因此,如果能减少连接次数进而减少数据库扫描时间,或者对候选频繁项目集不存储,以减少内存空

李晓虹 讲师,硕士研究生,主要研究方向:系统理论,数据挖掘。

间，则算法的性能将有所提高。这是本文提出的事务项目的十字链表方法的工作出发点。

3 Apriori 算法改进

3.1 事务项目的十字链表表示方法

假设：1）项目总集为 $I=\{i_1,i_2,\cdots,i_m\}$，其中元素 $i_l(1\leqslant l\leqslant m)$ 代表具体的某个项目，m 为项目总数；2）事务集合为 $U=\{x_1,x_2,\cdots,x_n\}$，其中元素 $x_l(1\leqslant l\leqslant n)$ 代表某个具体的事务，n 是数据库中事务总量；3）事务标识集合为 $T=\{t_1,t_2,\cdots,t_n\}$，其元素 $t_l(1\leqslant l\leqslant n)$ 是对应某个事务 x_i 的标识，且 t_l 和 x_l 具有一一对应关系；4）事务项目集合 $C=\{C_1,C_2,\cdots,C_n\}$，$C_i\subseteq I(1\leqslant i\leqslant n)$ 表示某个具体事务中由哪些项目组成。

则定义：1）$A=T\cup C$，表示每个事务的组合方式；2）$D=(U,A)$，它代表事务数据库，即事务数据库 D 由具体的事务及其生成关系共同构成；3）$M=(U^T,A,V,f)$，它代表事务项目的十字链表，其中 U^T 表示 U 的转置，$V=\{\langle i_k,t_j\rangle\mid i_k\in I,t_j\in T,k=1,2,\cdots,m,j=1,2,\cdots,n\}$ 表示中间结点集合，其元素 $\langle i_k,t_j\rangle$ 由 C 中每个 C_i 的项目 i_k 与 T 中的每个标识 t_j 共同组成，f 表示一种映射关系，定义为 $f:C\times T\rightarrow V$。

假设某个事务数据库，经过预处理后得到的数据库模式为〈事务标识，项目 1，项目 2，…〉，它按事务标识值由小到大排列，项目名用 I1、I2、I3 等表示，如表 1 所示，其对应的事务项目十字链表如图 1 所示。

表 1

Tid	Items
1	I1,I2,I5
2	I2,I4
3	I1,I3
4	I2,I5
5	I1,I4
6	I2,I3
7	I2,I3,I5

在图 1 中，Tid 指向的表称为事务标识头表，它的每个结点包含事务标识和一个 down 指针。Item 指向的表称为项目头表，每个结点包含项目名和一个 next 指针。链表的中间结点如图 2 所示，它由项目名 Item，事务标识 Tid，down 指针和 next 指针 4 个部分组成。在十字链表中，每一列 down 指针所指的结点数对应事务数据库一个事务项目集，每一行 next 指针所指的结点数对应事务数据库中该项目的支持数。比如：Tid 值为 1 的事务〈1，I1，I2，I5〉，通过 Tid 头表中第一个结点的 down 指针，即图 1 中向下的虚线箭头，连接与其相关的具体项目 I1、I2 和 I5；Item 值为 I2 的项目，通过 Item 头表的第二个结点的 next 指针，即图 1 中向右的粗线箭头，指向包含 I2 的具体事务。因此，所有数据库中关于事务与项目的有关信息在十字链表中通过结点方式表达出来，事务数据库和十字链表的表示方法等价。

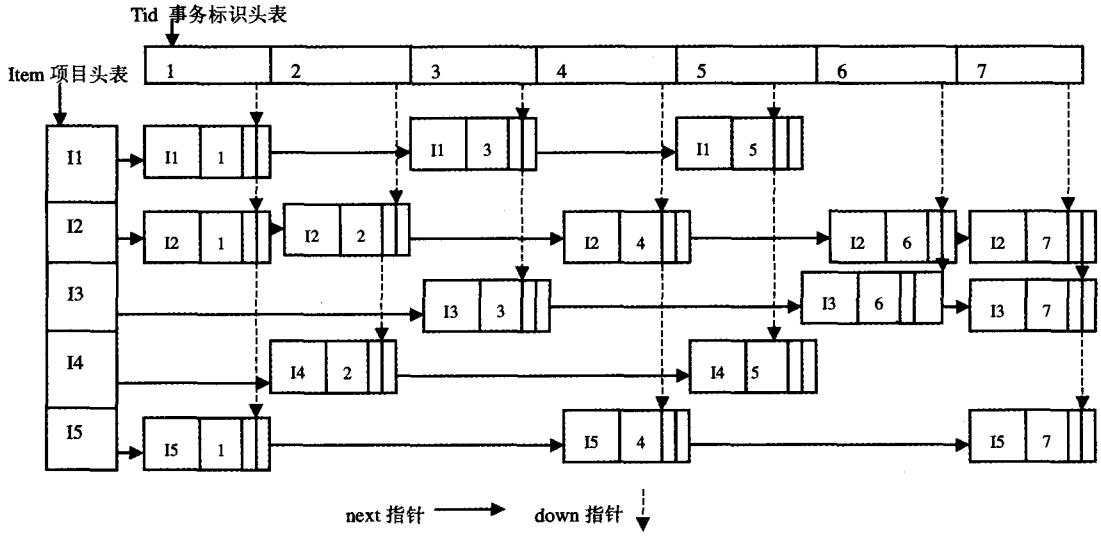

图 1

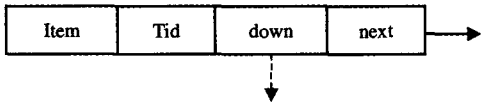

图 2

3.2 改进的 Apriori 算法

根据上述事务项目的十字链表表示法，将事务数据库的扫描转化为对十字链表的操作，并对十字链表进行频繁项目集的挖掘。首先通过项目头表中每个项目结点的 next 指针，求得每个项目的支持数，与给定 Minsup 比较即可生成频繁 1 项目集 L_1。而频繁 $k(k\geqslant 2)$ 项目集的生成，由 L_{k-1} 中的结点两两连接形成组合 c，并对其进行剪枝判断后，立即进行支持数计算。支持数的计算方法是：在十字链表中，同时扫描组合 c 中包含的项目所在的行，并根据各项目结点的 next 指针，统计出结点中 Tid 值同时相同的结点个数，即组合的支持数，如果支持数大于给定的 Minsup，则将组合并入频繁 k 项目集 L_k 中。比如组合〈I2，I5〉，在十字链表中同时扫描项目名为 I2 和 I5 所在行，根据 next 指针统计出结点标识 Tid 值同时相同的结点数为 3，即支持数为 3；类似地〈I2，I3〉的支持数为 2。相比 Apriori 算法的计数方法而言，本文算法只扫描了组合 c 中所包含的项目所在行，没有对整个 DB 进行扫描，因此减少了扫描 DB 的时间。另外，算法以连接→剪枝→计数方式产生频繁项目集，避免了 Apriori 算法中对候选频繁项目集 C_k 的临时存储，从而节省了内存开销。

图 3 和图 4 所示列出了频繁 1 项目集和频繁 2 项目集的链表存储结构。链表中每个结点由两个成员构成，一个是指

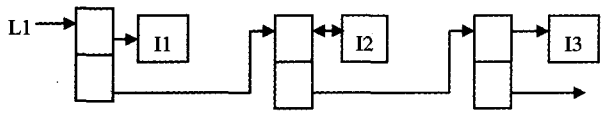

图 3

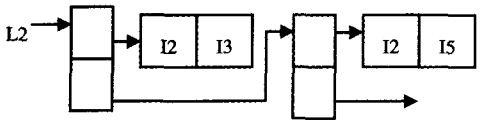

图 4

向项目的 Item 指针，一个是指向下一个结点的 next 指针。在进行 L_1 连接时，其节点中所有单个项目两两连接，形成长度为 2 的项目组合，如图 3 中的〈I1〉和〈I2〉，连接后为〈I1，I2〉；在进行 L_2 连接时，再将长度为 2 的项目组合连接成长度为 3 的项目组合，如图 4 中的〈I2，I3〉和〈I2，I5〉，连接后为〈I2，I3，I5〉。

基于上述思想，将改进后的 Apriori 算法称为 New_Apriori，用伪代码描述如下：

```
输入：事务数据库的十字链表 M，和最小支持数 Minsup；
输出：频繁项目集 L；
1)  L1=find_1(M,Minsup);
2)  for(k=2;Lk-1≠Ø;k++) do begin
3)   for(p=Lk-1;p! =NULL;p=p→next) do begin
4)     for(q=p→next;q! =NULL;q=q→next) do begin
5)       c=my_gen(p,q); //p,q 结点的项目连接
6)       if(prunc(Lk-1,c)&&check_c(Lk,c))
7)         continue; //回到 q 循环处
8)       count=count_c(M,c);
9)       if(count≥Minsup)
10)        insert(Lk,c); //将组合 c 插入 Lk 链表中
11)    end
12)   end
13)  end
14) Answer=⋃k Lk;
```

在伪代码中，my_gen 函数和 prunc 函数功能是连接、剪枝判断，与原 Apriori 算法一致，check_c 函数是判断 c 是否已被插入到了 L_k 中，count_c 函数用于计算组合 c 的支持数。第 6 行和第 7 行的功能是：如果组合 c 被剪枝或已经在 L_k 中时，则跳过以后的计数扫描，回到 q 循环处进行下一次连接。当统计出组合 c 的支持数大于最小支持数时，则将组合 c 插入 L_k 中。

4 算法性能与实验

设 N 表示事务数据库的交易数，T 表示最大交易长度。在 New_Apriori 算法中，find_1 函数的时间复杂度为 $O(T\times N)$。在其后的循环步骤中，连接时间复杂度为 $O(|L_{k-1}|\times L_{k-1}-1/2)$，剪枝判断时间复杂度为 $O(|C_k|)$，扫描计数时间复杂度为 $O(N\times|C_k|)$，而基本的 Apriori 算法的连接时间复杂度为：$O(|L_{k-1}|\times|L_{k-1}|)$。因此尽管改进后的算法和 Apriori 算法具有相同的时间复杂性，但由于连接次数减少了，提高了算法收敛的速度，使得改进后的算法在时间开销上有所节约。

从空间复杂度方面来看，原 Apriori 算法的空间复杂度为 $O(|L_1|)+\sum_{k\geq 2}(O(|C_k|)+O(|L_k|))$，改进后的算法只需保留频繁项目集 L_k 的空间，没有保留候选频繁项目集 C_k，空间开销为 $O(|L_1|)+\sum_{k\geq 2}O(|L_k|)$。因此改进后的算法节省的存储空间为：$\sum_{k\geq 2}O(|C_k|)$。

用 VC6.0 语言分别实现了 Apriori 算法、AprioriTid 算法和本文新算法 New_Apriori，并在相同环境下进行比较测试。测试环境为：PIII CPU、256MB 内存、80G 硬盘，Win2000 操作系统，数据来源于 2001 年 1～6 月上半年的沪市股票。抽取其中的 200 支股票，进行单层单维布尔关联规则的挖掘分析，实验结果如图 5 所示。从图中可以看出：1)无论支持数的值是大还是小，New_Apriori 算法时间曲线总在 Apriori 算法的下面，即 New_Apriori 算法时间开销小于 Apriori 算法，进一步计算表明，New_Apriori 算法比 Apriori 算法节省时间约 25%左右。2)当最小支持数较小时，New_Apriori 算法具有最小的耗时，当最小支持数增大时，New_Apriori 算法与 AprioriTid 算法时间开销相当。这是由于 AprioriTid 算法含有 $\overline{C}_k$ 中连接、剪枝的时间开销，而 New_Apriori 算法则没有这项时间开销，当最小支持数较小时，$\overline{C}_k$ 中的成员较多，AprioriTid 算法所需的连接、剪枝以及计数的时间较大，随着最小支持数的增加，$\overline{C}_k$ 中的成员逐渐减少，所需的连接、剪枝以及计数的时间逐渐减小，从而导致 AprioriTid 算法的时耗降低。

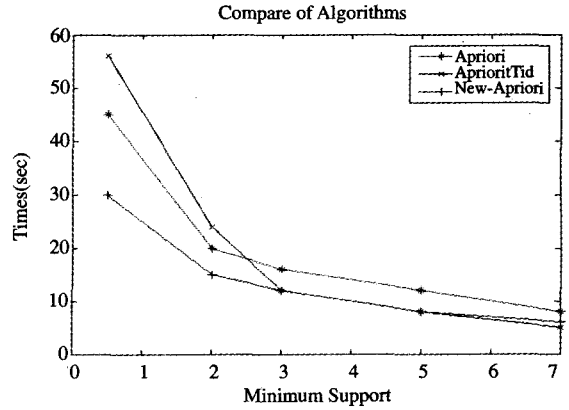

图 5

结束语 关联规则的挖掘是当前数据挖掘领域的主要研究课题。本文对关联规则挖掘中 Apriori 算法进行了改进，提出了一种基于事务项目十字链表表示的 New_Apriori 算法，实验结果表明，它的性能较原始的 Apriori 算法有了明显的提高，较 AprioriTid 算法也可一比高低。由于数据库与十字链表结合起来，提高了插入和删除结点的操作方便性，因此该方法更适用于事务数据的增量挖掘。

参 考 文 献

1 Agrawal R, Imielinski T, Swami A. Mining association rules between sets of items in large databases [C]. In: Proc. of the 1993 ACM on Management of Data, Washington, D. C, May 1993. 207～216

2 Agrawal R, Srikant R. Fast algorithms for mining association rules [C]. In: Proc. of the 20th Int'l Conf. VLDB's94, Santiago, Chile, Sept. 1994. 487～499

3 Srikkant R, Agrawal R. Mining quantitative association rules in large relational tables [C]. In: Proc. of the 1996 ACM SIGMOD Conf. on Management of Data, Montreal, Canada, June 1996. 1～12

4 王晓峰，王天然，赵越. 一种自顶向下挖掘长频繁项的有效方法[J]. 计算机研究与发展，2004,41(1):148～155

5 刘君强，孙晓莹，潘云鹤. 关联规则挖掘技术研究的新进展[J]. 计算机科学，2004,31(1):110～113