关联规则的增量式更新算法*

冯玉才 冯剑琳

(华中理工大学计算机系 武汉 430074)

摘要 关联规则的开采是一个重要的数据开采问题.目前已经提出了许多算法用于高效地发现大规模数据库中的关联规则,而对关联规则维护问题的研究工作却很少.在用户开采关联规则的交互过程中,为了找到真正令其感兴趣的规则,用户将需要不断调整两个描述用户兴趣程度的阈值:最小支持度和最小可信度.本文提出了两种增量式更新算法——IUA(incremental updating algorithm)和PIUA(parallel incremental updating algorithm),用来解决这一关联规则高效维护问题.

关键词 数据开采,知识发现,关联规则,增量式更新,频繁项目集.

中图法分类号 TP311.13

数据开采(Data Mining)又称数据库中的知识发现KDD(knowledge discovery in databases),已经被认为是数据库研究中的一个极富应用前景的新领域.这一领域可以定义为在大规模数据库中高效地发现潜在可用的模式(Patterns)或规则(Rules).推动数据开采迅猛发展的是大型零售组织所面临的决策支持问题.条型码技术的发展已经使得超级市场能够收集和存储数量巨大的销售数据.一条这样的数据记录通常都包括与某个客户相关的交易(Transactions)日期、交易中所购物品项目(Items)等等.通过对以往的大量交易数据进行分析就能够获得有关客户购买模式的有用信息,从而提高商业决策的质量.[1,2]

在交易数据项目之间开采关联规则的问题(Mining Association Rules)是R. Agrawal等人在文献[1]中首先引入的.有一个关联规则的例子就是"90%的客户在购买面包和黄油的同时也会购买牛奶",其直观的意义是,客户在购买某些东西的时候有多大的倾向也会购买另外一些东西.找出所有类似这样的规则,对于确定市场策略是很有价值的.关联规则的其他应用还包括附加邮递、目录设计、追加销售、仓储规划以及基于购买模式对客户进行划分等等.这些应用中的数据库都是极其庞大的,因此,不仅需要设计高效的算法来开采关联规则,而且也迫切需要设计高效的算法来更新、维护和管理已开采出来的关联规则.目前大量的研究工作主要集中在开采算法方面[1~9],而对更新算法方面的研究工作却很少.[10]

本文主要考虑当最小支持度minsup(minimum support)和最小可信度minconf(minimum confidence)发生变化时当前交易数据库中关联规则的更新问题,提出了两种高效的增量式更新算法IUA(incremental updating algorithm)和PIUA(parallel incremental updating algorithm).详细的问题描述在第1节给出.第2节描述IUA算法和PIUA算法.性能分析则在第3节讨论.第4节作出总结.

1 问题描述

1.1 关联规则的开采

关联规则开采问题的形式化描述如下:[1,2]

假设$I=\{i_1,i_2,\ldots,i_m\}$是m个不同项目的一个集合.给定一个交易数据库D,其中每一个交易T是I中一组项目的集合,即$T\subseteq I$.每一个交易都与一个唯一的标识符TID相联.如果对于I中的一个子集X,有$X\subseteq T$,我们就说一个交易T包含X.一条关联规则(Association Rule)就是一个形如$X\Rightarrow Y$的蕴涵式,其中$X\subset I, Y\subset I$,而且$X\cap Y=\varnothing$.如果D中$c\%$的包含X的交易同时也包含Y,则关联规则$X\Rightarrow Y$在D中以可信度(Confidence)成立.如果D中$s\%$的交易包含$X\cup Y$,则关联规则$X\Rightarrow Y$在D中具有支持度(Support).关联规则的开采问题就是生成所有具有用户指定

* 本文研究得到国家863高科技项目基金资助.作者冯玉才,1945年生,教授,博士导师,主要研究领域为数据库,多媒体,GIS.冯剑琳,1970年生,博士生,主要研究领域为数据开采.

本文通讯联系人:冯玉才,武汉 430074,华中理工大学计算机系

本文1997-04-22收到原稿,1997-07-18收到修改稿

的最小支持度和最小可信度的关联规则,即这些关联规则的支持度和可信度分别不小于最小支持度和最小可信度.

关联规则的开采问题可以分解成以下两个子问题:

①找出交易数据库 D 中所有具有用户指定最小支持度的项目集(itemset,I 的一个非空子集).具有最小支持度的项目集称为频繁项目集(Frequent Itemsets),反之就称为非频繁项目集.

②利用频繁项目集生成所需要的关联规则.对于每一个频繁项目集 A,找出 A 的所有非空子集 a,如果比率 support(A)/support(a)≥minconf,就生成关联规则 $a\Rightarrow(A-a)$.

由于第 2 个子问题较为容易和直观,目前大量的研究工作主要都集中在第 1 个子问题上.[1~4]

1.2 相关工作

在已经提出的许多算法中,R. Agrawal 等人在文献[2]中提出的 Apriori 算法是最有影响的.[9,10]除了最初提出的性能较 Apriori 差的 AIS 算法及其面向 SQL 的变体 SETM[1,2,8],目前已知的大多数算法都是以 Apriori 为核心,或是其变体,或是其扩展.[2,3,6,9]

Apriori 是一种宽度优先算法,通过对数据库 D 的多趟(Pass)扫描来发现所有的频繁项目集,在每一趟 k 中只考虑具有同一长度 k(即项目集中所含项目的个数)的所有项目集.在第 1 趟扫描中,Apriori 算法计算 I 中所有单个项目的支持度,生成所有长度为 1 的频繁项目集.在后续的每一趟 k 中,首先以前一趟中所发现的所有频繁项目集为基础,生成所有新的候选项目集(Candidate Itemsets),即潜在的频繁项目集,然后扫描数据库 D,计算这些候选项目集的支持度,最后确定候选项目集中哪一些真正成为频繁项目集.重复上述过程直到再也发现不了新的频繁项目集.算法高效的关键在于生成较小的候选项目集,也就是尽可能不生成和计算那些不可能成为频繁项目集的候选项目集.[2,3,8]为了实现这一点,所有已知的算法都利用了这样一个基本性质,即一个频繁项目集的任一子集必定也是频繁项目集.

1.3 关联规则的更新

D. W. Cheung 等人首先考虑了关联规则的高效更新问题.他们考虑的问题是给定交易数据库 DB,假定最小支持度不变,当一个新的交易数据集 db 添加到 DB 中去时,如何生成 $DB\cup db$ 中的关联规则.他们提出了增量式更新算法 FUP,FUP 的基本框架和 Apriori 是一致的.[10]

本文的目的在于,考虑当交易数据库 D 保持不变,而最小支持度和最小可信度发生变化时,关联规则的高效更新问题.事实上,为了发现事先未知的关联规则,用户必然需要通过对最小支持度和最小可信度这两个阈值的不断调整来逐步聚焦到那些真正令其感兴趣的关联规则上去,这将是一个动态的交互过程,因此,迫切需要高效的更新算法来满足用户对较快的响应时间的需求.显然,对于最小可信度发生变化时的关联规则的更新问题就如同 1.1 节的第 2 个子问题一样直观,因此,我们主要考虑最小支持度发生变化时关联规则的高效更新问题,并且这一问题也归结为发现在新的最小支持度下的所有频繁项目集.对于这一更新问题,一种可能的方法就是将关联规则的开采算法如 Apriori 以新的最小支持度重新运行一遍.这种方法虽然简单明了,却有着明显的不足.因为最初用来发现旧的频繁项目集的计算都将被浪费,所有的频繁项目集都必须从头开始计算.本文提出的增量式更新算法 IUA 和 PIUA 将利用从旧的频繁项目集所获得的信息来高效发现所有新的频繁项目集.

2 增量式更新算法 IUA 和 PIUA

给定交易数据库 D,一个项目集的支持度可以就认为是所有包含该项目集的交易的数目.设旧的最小支持度为 s,L_k 为这时所有频繁 k 项目集(frequent k-itemsets,即长度为 k 的频繁项目集)的集合,$k=1,2,\ldots,m_1$,这里 m_1 为所有频繁项目集长度中的最大者.同样地,对于新的最小支持度 s',设 L_k' 为所有频繁 k 项目集的集合,$k=1,2,\ldots,m_2$.对于每一个项目集都有一个域 *count* 用来保存它的支持度计数.

当最小支持度发生改变时,可以分为如下两种情况:

①$s'>s$,原有的一些频繁项目集可能失去最小支持度;

②$s'<s$,原有的一些非频繁项目集可能获得最小支持度.

在第①种情况下,利用频繁项目集的 *count* 域,关联规则的更新是很直观的,算法描述如下:

算法 1. 增量式更新算法($s'>s$)

(1) for ($k=1;k<m_1;k++$) do begin

(2) $L_k'=\{c\in L_k\mid c.count\geq s'\}$;

(3) end

(4) Answer$=\bigcup_k L_k'$

现在主要来考虑第②种情况,我们将提出两种高效的更新算法 IUA 和 PIUA.

2.1 IUA 算法

IUA 算法的基本框架也和 Apriori 算法一致,也需要对交易数据库 D 进行多趟扫描.因为有 $s'>s$,所以原来所有的频繁 k 项目集(L_k)在新的最小支持度下仍然是频繁 k 项目集,因此在每一趟中扫描交易数据库 D 计算候选 k 项目集的支持度计数时,我们就没有必要再考虑一遍 L_k 对应的候选 k 项目集.如果更进一步希望避免又重新生成一遍 L_k 对应的候选 k 项目集,我们可以考虑采取以空间换时间的策略,只要在 Apriori 算法中的每一趟 k,保存相应的(C_k-L_k)即可.

在第 1 趟扫描中,IUA 算法只对原来不在 L_1 中的单个项目进行支持度计算,并确定出所有新的频繁 1 项目集 L_1'',然后通过 $L_1''\cup L_1$ 得到 L_1'.

在后续的每一趟 k 中,包括两个阶段.首先生成计算 L_k' 所需的候选 k 项目集(candidate k-itemsets,即长度为 k 的候选项目集).设 C_k 为所有候选 k 项目集的集合,一种质朴的增量式方法就是我们利用 Apriori 算法中的 apriori_gen 函数[2]按如下方式来生成 C_k:

$$C_k=\text{apriori\_gen}(L_{k-1}')-L_k.$$

为了更好地利用从旧的频繁项目集所获得的信息来高效地发现所有新的频繁项目集,我们提出了一种新的生成所有候选 k 项目集 C_k 的方法,这一新方法的关键就在于我们发掘的如下一个重要事实.

在第 1 趟中,所有的频繁 1 项目集已经被分成两个不相交的集合:L_1 和 L_1''.又因为一个频繁项目集的任一子集必定也是频繁项目集,所以频繁 k 项目集 c 中的每一单个项目 i 所对应的频繁 1 项目集(i)或者从 L_1 中取,或者从 L_1'' 中取.根据这一点,我们就可以将具有新的最小支持度 s' 的所有频繁 k 项目集分成 3 类:

①对于其中的每一个频繁 k 项目集 $c=\{i_1,i_2,\ldots,i_k\}$,$\forall\ j(1\leqslant j\leqslant k)$,必有$\{i_j\}\in L_1$;

②对于其中的每一个频繁 k 项目集 $c=\{i_1,i_2,\ldots,i_k\}$,$\forall\ j(1\leqslant j\leqslant k)$,必有$\{i_j\}\in L_1''$;

③对于其中的每一个频繁 k 项目集 $c=\{i_1,i_2,\ldots,i_k\}$,必有两个非空子集 c_1 和 c_2,使得 $c_1\cup c_2=c$,$c_1\cap c_2=\varnothing$,而且 $c_1\subset L_1$,$c_2\subset L_1''$.

因此我们就得到 L_k' 的一个分划,L_k' 由 3 个互不相交的子集构成.我们将所有第①类频繁 k 项目集构成的集合记为 L_k^1,第②类记为 L_k^2,第③类记为 L_k^3.同时与之相对应的候选 k 项目集构成的集合分别记为 C_k^1,C_k^2,C_k^3.其中 C_k^1 之中可以去掉原有频繁 k 项目集 L_k.这样生成的第①类频繁 k 项目集构成的集合记为 L_k^0,即有 $L_k^1=L_k^0\cup L_k$.于是,我们有 $L_k'=L_k^1\cup L_k^2\cup L_k^3$,而且 $L_k^1\cap L_k^2=\varnothing$,$L_k^2\cap L_k^3=\varnothing$,$L_k^1\cap L_k^3=\varnothing$.

对于 C_k^1 和 C_k^2,我们直接利用 Apriori 算法中的 apriori_gen 函数按如下方式生成:

$$C_k^1=\text{apriori\_gen}(L_{k-1}^1)-L_k;$$

$$C_k^2=\text{apriori\_gen}(L_{k-1}^2).$$

我们提出一个新的候选 k 项目集生成函数 iua_gen(L_j)来生成 C_k^3.事实上,C_k^3 中的每一个候选 k 项目集只需将一个第①类频繁 j 项目集($1\leqslant j\leqslant k-1$)和一个第②类频繁($k-j$)项目集进行简单的拼接即可,这里假定这一对项目集都不为空.iua_gen(L_j)函数分为两步:

(1) 拼接

insert into C_k^3

select $p.item_1,p.item_2,\ldots,p.item_j,q.item_1,q.item_2,\ldots,q.item_{k-j}$

from $L_j p,L_{k-j}^2 q$

(2) 修剪

forall itemsets $c\in C_k^3$ do

  forall ($k-1$)-subsets s of c do

    if ($s\notin L_{k-1}^3$) then

      delete c from C_k^3;

将所有的第①类频繁 j 项目集与相对应的第②类频繁($k-j$)项目集通过 iua_gen(L_j)函数进行拼接,就得到 C_k^3.

现在,我们来说明候选 k 项目集生成函数 iua_gen(L_j)的正确性,也就是 $C_k^3\supseteq L_k^3$ 成立.正如前面所定义的,L_k^3 中的每一个频繁 k 项目集都是由两个不相交的非空子集 c_1 和 c_2 组成.又由于一个频繁项目集的任一子集必定也是频繁项目集,因此 c_1 和 c_2 必然都是频繁项目集.而 iua_gen(L_j)中的拼接步就相当于将每一个第①类频繁 j 项目集分别和每一个第②类频繁($k-j$)项目集进行集合的"并"操作,从而生成所有潜在的频繁 k 项目集.为了避免重复生成相同的候选 k 项目集,我们只需将第①类频繁 j 项目集从 1 到($k-1$)迭代.通过迭代,我们就考虑了将频繁 k 项目集划分为

```
            %CHN ChannelDeclaration
            %LOC [VarDeclaration]
            %RLE[
              ce1|ce2|...|cen
            ]
        ]
    ],
    %PACK[
        ActorName:
            FATHERNAME: parent
            %VAR [AttributeDeclaration]
            %PROC [ActorActionDeclaration]
    ]
  ]
];
```

其中,(1) ce_i 即为条件元(Conditional Element),其形式为:$LB=S_i \wedge P_i \Rightarrow @_i(Q_i \wedge act_i \wedge BG=b_i \wedge TR=t_i \wedge LB=SE_i)$. 其中,$@_i$ 是一时序逻辑算子;P_i,Q_i 都是合式的时序逻辑公式;Q_i 、act, 为当条件 P_i 满足时角色的行为;BG 是场景的背景描述;TR 给出行为的执行时间;LB 是一个指明当前状态值下一状态的控制元.

(2) parent 是子角色的父角色的名字,以此实现父子角色.

(3) AttributeDeclaration 用以定义角色的属性数据.

(4) ActorActionDeclaration 用以定义角色行为.它的描述形式为:

```
%PROC [
  actoraction1;
  ......
  actoractionn
]
```

它给出动画中角色动作的详细描述,由若干动作函数(Actoraction)构成.一个动作函数具体刻画角色的一个运动特征,它的描述项有:平移参数、旋转参数、变比参数、参考坐标系等.每个动作函数具有若干虚参变元,由变换来刻画.变换包括平移变换、旋转变换、伸缩变换等.变换定义了4种运动类型:F,L,S,D,分别表示不变、线性变化、样条曲线变化和以数据文件给定变化过程.一个动作函数描述实例如下:

```
walk(%INP x1,y1,z1:FLOAT; %INP x2,y2,z2:FLOAT)==[
%ALG [LB=START⇒ $OT (x1,y1,z1→x2,y2,z2) ∧ $OL ∧ $OLB=L1;
      LB=L1⇒ $OReference (refman) ∧ $OF ∧ $OLB=STOP
     ]]
```

(5) 这里 ACT,BG,TR 为我们引入的动作原语的描述,它们合在一起构成了行为客体的动作原语描述.除了行为客体的第1次动作原语描述和背景发生变化外,背景描述(BG)部分可以省略,省略的缺省值为前一时间段的背景;执行时间描述(TR)也可省略,缺省值为1.一般来说,TR 给出行为动作的执行时间,但有例外.譬如一人在等待一辆车的到来,这人"等待"这一动作的执行时间省略或者给出了一执行时间.但实际上,"等待"这一动作的执行时间取决于车到来所用的时间,这就涉及到两个行为客体之间的动作同步协调问题.因此"等待"这一动作的执行时间,将持续到车到来为止.其中的协调关系,通过消息传递来实现.何时结束,由车向人发出信号而定.

2.2 同步协调

ADL 中,对每一个角色(包含光源和摄影机)的运动行为的描述为一个代理机构,角色之间行为的同步协调关系是通过进程之间通道的消息传递来实现的(上面已提到).通道是可以动态决定的,XYZ/E 中,有两条与通道操作相关的通信命令,即输出命令(写通道)ch!y 和输入命令(读通道)ch?x,这里 ch 是从输出进程到输入进程之间的通道的名字,y 是输出进程的输出表达式,x 是输入进程的输入信号变量.传递的信号既可以是不同角色之间的同步协调信号,也可以是角色之间、父子角色之间的命令应答信号.

譬如,猫追老鼠,老鼠逃进洞或者逃跑过程中被猫逮住吃掉的动画过程:猫出现,老鼠开始逃跑;老鼠逃跑,猫开始追赶.老鼠何时开始逃,取决于猫的出现.猫一出现,老鼠警觉,于是开始逃.其间的同步,由猫发出信号,老鼠接收;而猫何时开始追赶老鼠,取决于它何时发现老鼠.老鼠开始逃跑,发出在逃信号,猫发觉(即接收到这一信号),于是开始追赶.而老鼠逃得及时,逃进洞,猫停止追赶,或者不幸被猫逮着,猫也因而停止追赶.这一过程由老鼠发信号,猫接收.至于是老鼠逃进洞信号,还是被逮住信号,可视场面情况动态决定.此外,猫追的动作和老鼠逃的动作,其本身也需要同步协调.比如猫追的动作,怎样跨前腿,动后腿,迈左腿,抬右腿,这需要协调一致,有板有眼.实现它,是通过猫向腿

两个不相交非空子集的所有组合情况.因此,C_k^3必然是L_k^3的一个超集,亦即$C_k^3 \supseteq L_k^3$.其次,在修剪步,iua_gen(L_j^i)删除的只是那些根本不可能出现在L_k^3中的项目集.在这里,我们也是利用了"一个频繁项目集的任一子集必定也是频繁项目集"这一基本性质.综上,我们就有$C_k^3 \supseteq L_k^3$成立.同理可知,$C_k \cup L_k \supseteq L_k'$也是成立的.

C_k生成之后,紧接着就扫描数据库D,最终生成L_k'.

上述过程一直重复到不再有新的频繁项目集生成为止.我们可以根据$L_k^i(i=1,2,3)$是否为空独立地决定是否还需进入$(k+1)$趟计算L_{k+1}'.显然,对L_k^3的计算总是最后一个结束的.在下面的算法描述中,我们只简单地以判断L_k'是否为空来决定IUA算法是否还需进入$(k+1)$趟.

IUA算法的基本框架描述如下所示.

算法 2. IUA算法($s'<s$)

(1) L_1''={new frequent 1-itemsets};$L_1'=L_1'' \cup L_1$;
(2) for ($k=2;L_{k-1}' \neq \varnothing;k++$) do begin
(3) C_k^1=apriori_gen(L_{k-1})$-L_k$;
(4) C_k^2=apriori_gen(L_{k-1}^2);
(5) $C_k^3=\varnothing$;
(6) for ($j=1;j \leqslant k-1;j--$)do begin
(7) $C_k^3=C_k^3 \cup$iua_gen(L_j^1); /* 生成所有第③类候选k项目集 */
(8) end
(9) for all transactions $t \in D$ do begin
(10) C_{t1}=subset(C_k^1,t); /* C_k^1中包含于交易t的候选k项目集构成C_{t1} */
(11) for all candidates $c \in C_{t1}$ do
(12) $c.count++$; /* C_{t1}中候选k项目集的支持度计数加1 */
(13) C_{t2}=$subset(C_k^2,t)$;
(14) for all candidates $c \in C_{t2}$ do
(15) $c.count++$;
(16) C_{t3}=$subset(C_k^3,t)$;
(17) forall candidates $c \in C_{t3}$ do
(18) $c.count++$;
(19) end
(20) $L_k^1=\{c \in C_k^1 | c.count \geqslant s'\}$;$L_k^1=L_k^1 \cup L_k$;
(21) $L_k^2=\{c \in C_k^2 | c.count \geqslant s'\}$;
(22) $L_k^3=\{c \in C_k^3 | c.count \geqslant s'\}$;
(23) $L_k'=L_k^1 \cup L_k^2 \cup L_k^3$;
(24) end
(25) Answer=$\cup_k L_k'$.

2.2 PIUA算法

在IUA算法中,我们已经将所有的频繁k项目集分成了互不相交的3类,这使得IUA算法能够很容易实现基于共享内存(Shared-memory)多处理机结构的并行化,即PIUA算法.事实上,象PIUA这样的基于共享内存多处理机结构的并行算法特别有利于在限时应用中用来加快单个大顺序算法的计算.

设有处理机P_1,P_2,P_3,它们有一个共享内存,都能同样地运行IUA算法,但只有处理机P_3能够访问外存中的交易数据库D.让处理机$P_i(i=1,2,3)$负责计算对应的C_k^i和L_k^i对,处理机P_3同时还负责扫描交易数据库D.PIUA算法($s'<s$)的基本框架可以简单地描述如下:

算法 3. PIUA算法($s'<s$)

(1) 在第1趟中由处理机P_3生成所有的频繁1项目集L_1';
(2) 在后续的每一趟k中,包括下面的(3)-(6)步,由处理机P_3负责P_1,P_2,P_3的同步;
(3) 处理机P_i并行地计算对应的候选k项目集集合C_k^i;
(4) P_1和P_2等待P_3扫描交易数据库D到共享内存,然后处理机P_1,P_2,P_3并行访问共享内存中的交易数据、并行计算对应候选k项目集的支持度计数;
(5) 处理机P_i并行地计算对应的频繁k项目集集合L_k^i,最后由处理机P_3生成L_k';
(6) 处理机P_i根据对应的频繁k项目集集合L_k^i是否为空独立地决定是否还进入$(k+1)$趟;
(7) 处理机P_3最终生成Answer;

在上述算法中我们也可以有其他选择.例如,当处理机P_3扫描外存中的交易数据库D时,P_1,P_2可以不必等待,而是先行计算多趟的候选项目集,然后在一趟中计算已生成的多趟候选项目集的支持度计数或者就让P_3计算.显然,

在既共享内存又共享外存(Shared-everything)的多处理机结构中,很容易扩展图3中的PIUA算法来得到更高的并发度.

3 性能分析

对于本文所考虑的关联规则更新问题,一种直接的办法就是重新运行一遍Apriori算法.现在我们就将增量式更新算法IUA与重新运行一遍Apriori算法进行性能比较.

在$s'>s$的情况下,IUA算法显然将大大优于Apriori算法.现在来考虑$s'<s$的情况.模拟实验是在奔腾120上Windows NT 4.0下进行的,使用了与文献[2]同样的生成程序来合成所需的测试数据.[11]图1显示了测试数据库包含100 000个数据记录时的实验结果.

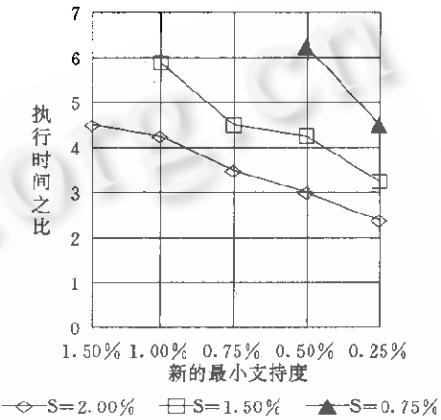

图1　性能加速比(T10.I4.D100K)

图1中3条折线分别对应旧的最小支持度s取值2.00%,1.50%和0.75%.测试数据库的有关参数使用文献[2]中同样的记号来表示:$|D|$表示交易数据记录的数目;$|T|$表示交易数据记录的平均长度;$|I|$表示最大的潜在频繁项目集的平均长度;$|L|$表示最大的潜在频繁项目集的数目;N表示交易项目的个数,而一个测试数据库实例则记为$Tx.Iy.DmK$,也就是$|D|=mK$,$|T|=x$以及$|I|=y$.在我们的实验中,$N=1\ 000$,$|L|=2\ 000$.实验结果表明在100 000个交易数据记录时,IUA算法比Apriori算法快2~6倍,而当交易数据记录增加到1 000 000个时,性能加速比为2~14倍.因此,IUA算法在增量式更新关联规则的意义上优于Apriori算法.

现在,我们来对实验结果作出分析说明.事实上,算法高效的关键在于如何高效地生成较小的候选项目集.在第1趟中Apriori算法需要对全集I中的所有单个项目进行支持度计数以生成频繁1项目集,而IUA算法只需要考虑$(I-L_1)$中的单个项目.在后续的第k趟中,在Apriori算法中,所有的候选k项目集为$C_k^a=\text{apriori_gen}(L_{k-1}')$;而在IUA算法中,$L_k$将从$C_k^1$中去掉,$C_k$肯定小于$C_k^a$.显然$L_1$愈大,则IUA算法的意义愈大.实验结果表明,当频繁项目集的分布在新旧最小支持度两种情况下相差不大时可以获得较高的性能加速比;其次,对于生成L_1^1对应的那部分候选k项目集,IUA算法也优于Apriori算法.在Apriori算法中是使用apriori_gen函数将两个频繁$(k-1)$项目集扩展成为候选k项目集,它需要$(k-1)$个连接(Join)条件来实现复杂的连接[2],而在IUA算法中只需使用iua_gen函数对两个子项目集进行简单的拼接即可;第3,在修剪步,IUA算法在生成$C_k^i(i=1,2,3)$时,只需对L_{k-1}^i单独进行检查就能实现有效的修剪,而对应地在Apriori算法中必须检查整个L_{k-1}'才能确定有效的修剪,因此,Apriori算法中修剪步的检查范围要比IUA算法多两倍的$|L_{k-1}'|$,即$3*|L_{k-1}'|-(|L_{k-1}^1|+|L_{k-1}^2|+|L_{k-1}^3|)$.当$L_{k-1}^1$,$L_{k-1}^2$和$L_{k-1}^3$在单独的情况下分别都可以装进内存,而$L_{k-1}'$却不能装进内存时,这一点尤其具有明显的意义.事实上,这时在Apriori算法中已经不能再做有效的修剪,因为不能在内存中得到整个L_{k-1}'.[3]在模拟实验中,当交易数据记录增加到1 000 000个,并且出现以上情况时,就得到了较高的性能加速比.

IUA算法的另外一大优点就是很容易实现基于共享内存多处理机结构的并行化.由于目前主要的并行关联规则开采算法都是基于无共享(Shared-nothing)多处理机结构的Apriori算法的并行化[4],因此,对于并行增量式更新算法PIUA的性能评价,我们就留待以后进一步的工作.

4 结　语

当用户交互式开采令其真正感兴趣的关联规则时,需要频繁调整最小支持度.针对这一问题,本文提出了两种高效的关联规则更新算法——IUA和PIUA.IUA算法的基本思想也可用于D. W. Cheung等人所考虑的关联规则更新问题,而且对于增量式开采一般化的关联规则[4](Generalized Association Rules)或者其他类型的规则如序列模式[12](Sequential Patterns)也是可以提供借鉴的.目前,我们正在将Apriori算法、IUA算法以及FUP算法集成为一个完整的关联规则开采系统,并将在自行研制的数据库管理系统DM2上实施.

参考文献

1 Agrawal R *et al*. Mining association rules between sets of items in large databases. In: Proceedings of ACM SIGMOD Conference on Management of Data, Washington, DC, May 1993. 207～216

2 Agrawal R, Srikant R. Fast algorithms for mining association rules. In: Proceedings of the 20th International Conference on Very Large Databases, Santiago, Chile, September 1994. 487～499

3 Agrawal R, Srikant R. Fast algorithms for mining associatien rules. IBM Research Report RJ9839, 1994

4 Agrawal R, Shafer J C. Parallel mining of association rules: design, implementation, and experience. IBM Research Report RJ 10004, 1996

5 Srikant R, Agrawal R. Mining generalized association rules. In: Proceedings of the 21st International Conference on Very Large Databases, Zürich, Switzerland, September 1994. 407～419

6 Park J S *et al*. An effective hash based algorithm for mining of association rules. In: Proceedings of ACM SIGMOD Conference on Management of Data, San Jose, California, May 1995. 175～186

7 Savasere A. An efficient algorithms for mining association rules in large databases. In: Proceedings of the 21st International Conference on Very Large Databases, Zürich, Switzerland, September 1995. 432～444

8 Houtsma M, Swami A. Set oriented mining association rules. In: Proceedings of the 11st International Conference on Data Engineering, Taipei, March 1995. 25～33

9 Toivonen H. Sampling large databases for association rules. In: Proceedings of the 22th International Conference on Very Large Databases, Bombay, India, 1996. 1～12

10 Cheung D W *et al*. Maintenance of discovered association rules in large databases: an incremental updating technique. In: Proceedings of the 12th International Conference on Data Engineering, New Orleans, Louisana, 1996. 106～114

11 http://www.almaden.ibm.com/cs/quest/data/assoc.gen.tar.Z

12 Agrawal R, Srikant R. Mining sequential patterns. In: Proceedings of the 11st International Conference on Data Engineering, Taipei, March 1995. 3～14

Incremental Updating Algorithms for Mining Association Rules

FENG Yu-cai FENG Jian-lin

(*Department of Computer Science Huazhong University of Science and Technology Wuhan* 430074)

Abstract Mining association rules is an important data mining problem. There have been many algorithms proposed for efficient discovery of association rules in large databases. However, very little work has been done on maintenance of discovered association rules. When users interactively mine association rules, they may have to continually tune two thresholds, minimum support and minimum confidence, which describe the users' special interestingness. In this paper, two incremental updating algorithms——IUA and PIUA are presented for such efficient maintenance problem of association rules.

Key words Data mining, knowledge discovery, association rules, incremental updating, frequent itemsets.