一种快速挖掘频繁项目集算法*

白石磊 毛雪岷
(中国科学技术大学 自动化系 合肥 230027)
(中国科学院合肥智能机械研究所 合肥 230031)
王儒敬 熊范纶
(中国科学院合肥智能机械研究所 合肥 230031)

摘 要 发现频繁项目集是多种数据挖掘应用中的关键问题,传统的算法是通过对事务数据库的多次扫描实现的,最新的研究主要围绕减少事务数据库的扫描次数进而减少挖掘过程的 I/O 代价来提高效率.本文提出一种快速挖掘频繁项目集的算法 FDFI(fast discovery frequent itemsets).该算法利用深度优先搜索的特点,结合频繁项目集的性质,有效地缩小了搜索空间,并采用独特的支持度计数策略,只需一次数据库扫描,就可计算所有项目集的支持度,大大减少了数据扫描量.最后作者对这一算法的性能进行了理论分析和实验验证.

关键词 数据挖掘,频繁项目集,深度优先搜索,项目标识映射
中图法分类号 TP311

1 引 言

随着数据库技术的发展和信息时代的来临,各行各业都积累了大量的数据,数据库中存储的数据量急剧增大.在这些海量数据中隐藏着许多重要的信息和知识,可以很好地支持人们进行预、决策.可数据库管理系统所能做到的只是对数据库中的数据进行存取、查询和简单的统计等操作,而不能对这些数据进行有效地分析处理,从而造成"数据丰富、知识贫乏"的状况.与此同时,人工智能领域的一个重要分枝——专家系统的研究和应用近年来也取得了很大的发展,然而专家系统中也同样面临着知识获取的瓶颈.因此,为了满足人们实际应用的需要,也由于数据库技术和机器学习技术的发展,数据库中的知识发现(Knowledge Discovery in Database, KDD),或称为数据挖掘(Data Mining, DM),已成为当前数据库与人工智能领域的一个热点研究课题.

数据库中的知识发现就是在数据库中提取隐含的、新颖的、潜在有用的知识[1].关联规则和序贯模式是 KDD 研究的重要方面.而发现频繁项目集则是挖掘关联规则[2]和序贯模式[3]的关键技术和步骤.发现频繁项目集是多种数据挖掘应用中的关键问题,传统的算法是通过对事务数据库的多次扫描实现的.目前挖掘频繁项目集的典型算法是 Apriori 算法或其变种[2,6,7].Apriori 算法采用自底向上宽度优先搜索,搜索空间巨大,而且须多次遍历数据库,当数据量巨大和频繁集维数较高时,时空开销将变得难以忍受.随着计算机性能的提高,探索合适的数据结构来支持基于一次数据库扫描的高效算法成为可能,最新的研究主要围绕减少事务数据库的扫描次数进而减少挖掘过程的 I/O 代价来提高效率[5].计算项目集的支持度是发现频繁项目集中最耗时的工作,缩小搜索空间和减少数据库扫描是降低开销的有效手段.因此本文提出了一种深度优先搜索快速挖掘频繁项目集的算法 FDFI(fast discovery frequent itemsets).

2 相关概念

2.1 关联规则和频繁项目集

关联规则[2]是由 R. Agrawal 于 1993 年提出的.

* 国家自然科学基金重点资助(No. 69835001)、国家 863 计划重点课题(2001AA115170)资助项目
收稿日期:2002-07-30;修回日期:2002-12-15

设 $I=\{i_1,i_2,\cdots,i_n\}$ 是一组物品(项目)集,D 是一组事务集(称之为事务数据库),D 中的每个事务 T 是一组物品(项目),显然满足 $T\subseteq I$.

定义1 称事务 T 支持物品(项目)集 X,如果 $X\subseteq T$.

定义2 称项目集 X 具有大小为 $s\%$ 的支持度,如果 D 中有 $s\%$ 的事务支持项目集 X. 记为 $sup(X)$.

定义3 支持度不小于用户定义的最小支持度 min_sup 的项目集称为频繁项目集(frequent itemsets),反之,则称为非频繁项目集(infrequent itemsets). 项目集中项目的数量叫做项目集的维数或长度.

定义4 关联规则是如下形式的一种蕴含:$X\rightarrow Y$,其中,$X\subset I$,$Y\subset I$,且 $X\cap Y=\varnothing$.

定义5 称关联规则 $X\rightarrow Y$ 在事务数据库 D 中有大小为 $s\%$ 的支持度,如果项目集 $(X\cup Y)$ 的支持度为 $s\%$,即 $support(X\rightarrow Y)=P(X\cup Y)$.

定义6 称规则 $X\rightarrow Y$ 在事务数据库 D 中有大小为 $c\%$ 的可信度,如果 D 中支持项目集 X 的事务中有 $c\%$ 的事务同时也支持项目集 Y,即 $confidence(X\rightarrow Y)=P(X\cup Y)/P(X)$.

有关项目集具有如下性质:

性质1 如果 X 是频繁项目集,则 X 的任何子集都是频繁项目集.

性质2 如果 X 是非频繁项目集,则 X 的任何超集都是非频繁项目集.

挖掘关联规则问题就是在给定的事务数据库 D 中产生所有满足用户给定的最小支持度(min_sup)和最小可信度(min_conf)的关联规则的过程. 关联规则的发现过程可分为两个步骤[7]:第一步,发现支持度大于给定的最小支持度的频繁项目集;第二步,从频繁项目集中产生关联规则,即当可信度 $conf\geqslant min\_conf$ 时,规则成立. 挖掘的性能主要由第一步决定. 发现频繁项目集的过程占了挖掘算法的主要开销,也就使得该过程成了算法研究的焦点.

2.2 深度优先搜索

知识表示和搜索是人工智能所面临的两个基本问题. 搜索作为通用问题求解策略,渗透于人工智能的各个方面. 深度优先搜索(depth-first search)也是一种盲目的搜索策略. 在基本的深度优先搜索中,首先扩展最新产生的(即最深的)节点,如图1所示. 深度相等的节点可以任意排列.

挖掘频繁项目集问题有它自己的特点,即在搜索过程及挖掘中间结果中,项集之间可能存在着相互包含关系,利用这个性质,可以剔除一些不必要的重复计算,缩小搜索空间. 目前挖掘频繁项目集的算法多采用宽度优先,而宽度优先搜索却无法利用这个特点,深度优先搜索则可以利用深度信息达到此目的. 目前已有工作者进行这方面的研究[8]. 针对于此,本文提出了基于深度优先搜索的频繁集挖掘算法.

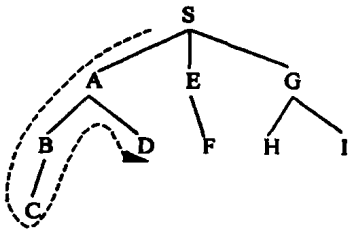

图1 深度优先搜索图

3 FDFI算法

3.1 集合枚举树

采用如图2所示的集合枚举树[4],来说明深度优先搜索的扩展策略与剪枝策略. 该枚举树为有序完全枚举树,可以方便的枚举所有可能的项目集. 频繁项目集的发现过程可视为在该集合枚举树上的搜索过程.

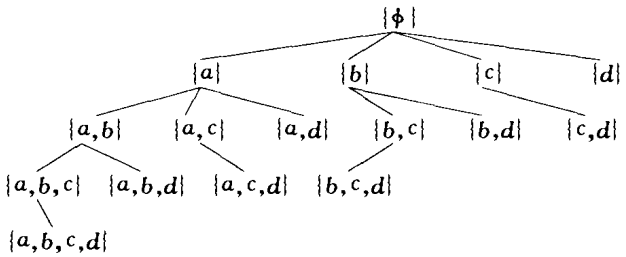

图2 4维项目集合 $D=\{a,b,c,d\}$ 的完全枚举树

树的生成方法如下:根节点为 $\{\varnothing\}$,一维项目 $L1:i_1,i_2,\cdots,i_m$ 排序构成深度为1的节点. 随着深度增加,节点 n 的子节点 n_c 生成方法是,$n_c=n\cup\{i_j\}$,$i_j\in L1$,且 j 大于 n 中各项目的序号.

3.2 扩展与剪枝策略

为了提高算法效率,以Apriori为代表的发现频繁项目集的算法都采用性质2来对候选项进行剪枝. 而通过深度优先搜索,则可以充分利用性质1和性质2协作进行剪枝. 对某节点 n:若 $sup(n)\geqslant min\_sup$,则 n 加入频繁项目集,扩展 n 的所有子节点,进一步搜索;反之,若 $sup(n)\leqslant min\_sup$,则 n 为非频繁项目集,由性质2可知,n 的超集(即 n 的所有子节点)也均为非频繁项目集,可剪掉. 然后回溯,

搜索 n 的兄弟节点或叔父节点 k,此时,应先考察 k 是否包含于已经发现的频繁项目集中,若包含于已发现的频繁项目集中,则根据性质 1,k 必为频繁项目集,不必再对 k 进行考察,直接扩展 k 的子节点即可.如图2,若$\{a,b,c\}$为频繁项目集,则接下来的搜索过程中,遇到节点$\{a,c\}$、$\{b,c\}$时,因其包含于已发现的频繁项目集$\{a,b,c\}$中,故不必再对其进行扫描计数,直接将其置入频繁项目集,进一步扩展考察其子节点即可.

表1　事务数据库

TID	Itemsets
1	*acf*
2	*dg*
3	*abe*
4	*bdg*
5	*adg*
6	*acf*
7	*acef*
8	*bd*
9	*def*
10	*acdf*

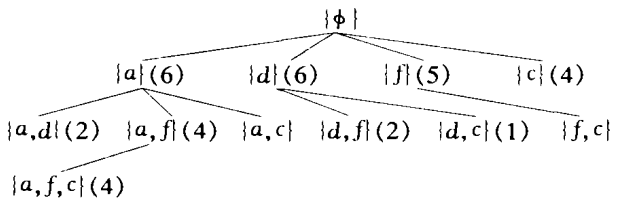

图3　FDFI算法的搜索路径和搜索空间

FDFI算法的搜索过程在频繁1-项目集的基础上进行.算法中设置了两个队列 $L1$ 和 OPEN(算法中以堆栈来实现),分别保存一维频繁项目集和待扩展节点,CLOSED表用来保存已发现的频繁项目集.将频繁 -1 项目集按支持度降序排列,置入 $L1$ 队列.首先从 $L1$ 队列取出首元素 n,(1)扩展当前节点的所有子节点,插入 OPEN 队列头部.(2)从 OPEN 队列取出首元素 k,如果 ①$k \subset I(I \in$ CLOSED),将 k 加入 CLOSED 表,返回(1).②$k \not\subset I(I \in$ CLOSED),对 k 进行考察,若 $sup(k) \geqslant min\_sup$,将 k 加入CLOSED表,返回(1);反之,删除 k,返回(2).直到 OPEN 表为空,说明以 n 为根的子树搜索完毕.再返回 $L1$ 队列取首元素,重复执行(1)、(2),直到 $L1$ 队列为空,全局搜索结束.CLOSED表中即为所有频繁项目集.

例如对表1所示的数据库,采用FDFI算法,其搜索空间和搜索路径如图3所示(假设预先设定 $min\_sup = 4$).

3.3　排序

某项集的支持度越高,就越有可能是最大频繁项目集的一部分.所以在一次扫描发现频繁 -1 项目集 $L1$ 之后,对 $L1$ 中的元素按支持度大小降序排列,使支持度高的项目优先得到扩展,这对尽早发现频繁大项集有重要意义.而频繁大项集对控制搜索路径,减少搜索空间起着很重要的作用.

3.4　支持度计数策略

计算项目集的支持度是发现频繁项目集中最耗时的工作.众所周知,对数据库的扫描伴随着繁重的磁盘 I/O 任务.当数据库规模巨大时,每遍历一次都需耗费大量的时间.Apriori 系列算法中,扫描次数较多,限制了发现算法的速度.本文提出了一种 TID 映射方法,只需一次数据库扫描,就可完成所有项目集的支持度计数工作,从而大大减少了数据读写任务,降低了算法的时间复杂度.

对数据库进行扫描,为每一个项目建立一个线性存储结构,如下所示:

Item-n	*tids*(n)	*sup*(n)

ltem-n 域为项目标识,*tids*(n)域为事务数据库中包含项目 n 的事务标识 TID 的集合,*support*(n)域为项目 n 的支持度计数,也就是 *tids* 域元素的数目.如对表1所示的事务数据库,通过一次数据库扫描,可得到如下结果:

a	1,3,5,6,7,10	6
b	3,4,8	3
c	1,6,7,10	4
d	2,4,5,8,9,10	6
e	3,7,9	3
f	1,6,7,9,10	5
g	2,4,5	3

取 $min\_sup = 4$,得到一维频繁项目集为$\{a\}$,$\{c\}$,$\{d\}$,$\{f\}$,删除支持度小于最小支持度的元素,得到如下结果:

a	1,3,5,6,7,10	6
c	1,6,7,10	4
d	2,4,5,8,9,10	6
f	1,6,7,9,10	5

按照上文所述深度优先搜索策略,当考察 n 维($n \geqslant 2$)项目集$\{i_1, i_2, \cdots, i_n\}$的支持度时,无需再

对原始数据库进行操作，只需简单地求项目集中各项目的 $tids$ 域的交集 $i_1.tids \cap i_2.tids \cap \cdots \cap i_n.tids$ 即可，通过集合交运算就可得到 $sup(n)$. 如对项目集 $\{a,d\}$，

$$sup(ad) = sizeof(a.tids \cap d.tids)$$
$$= sizeof((1,3,5,6,7,10) \cap (2,4,5,8,9,10)$$
$$= sizeof(5,10) = 2.$$

若要考察 $\{a,d\}$ 的子节点 $\{a,d,f\}$ 的支持度，则有

$$sup(adf) = sizeof(ad,tids \cap f.tids)$$
$$= sizeof((5,10) \cap (1,6,7,9,10)) = 1.$$

通过该方法只需一次数据库扫描，就可以快速计算出所要考察的项目集的支持度.

3.5 FDFI 算法描述

输入：事务数据库 DB，最小支持度 min_sup

输出：所有频繁项目集 FI

初始化：OPEN = $\emptyset$, CLOSED = $\emptyset$, $L1 = \emptyset$

(1) for each transaction(t) in DB //扫描数据库

(2) for each item(i), if $i \subseteq t$ do{

(3) TID(t) 加入 $i.tids$ 域;

(4) $i.sup$ ++;}

(5) *if* $i.sup \geq min\_sup$, do{

(6) $L1 \leftarrow i$; //用一维频繁项目集对 $L1$ 进行初始化

(7) CLOSED $\leftarrow i$;}

(8) 对 $L1$ 中的元素 i 按 $sup(i)$ 降序排列;

(9) if $L1 \neq \emptyset$, do{

(10) 从 $L1$ 中取出首元素 n, delete n from $L1$;

(11) *Ck_general*(n), 插入 OPEN 表头部; //扩展 n, 压入堆栈

(12) if OPEN $\neq \emptyset$, do{

(13) 从 OPEN 表中取首元素 k,

(14) if $k \subset I(I \in$ CLOSED)do

(15) {CLOSED $\leftarrow k$, delete k from OPEN, 转(11);}

(16) else do{

(17) $k.sup = Count(k)$; //用上文所述支持度计数策略考察 $k.sup$

(18) *if* $k.sup \geq min\_sup$, CLOSED $\leftarrow k$, delete k from OPEN, 转(11);

(19) else delete k from OPEN, 转(12)};

(20) else 转(9);}

(21) return CLOSED.

4 算法分析与实验结果

从搜索空间上分析，FDFI 算法采用深度优先搜索，综合运用频繁项目集的性质1和性质2来控制搜索过程. 若在某个深度上项目集 I_n 的支持度小于最小支持度，则其所有后继节点均可剪除(性质2). 若某项目集包含于已发现的高维频繁项目集中，则可省略对该项目集的支持度计算而直接置入频繁项目集中(性质1). 从而通过深度优先搜索策略有效地缩小了搜索空间. 特别是对频繁集维数较高的挖掘任务，算法的优势则越明显. 对一个 n 维频繁集的发现过程，传统的挖掘算法需要考察 2^n 个项集，而 FDFI 算法则只需考察 n 个项集. 例如对 (A,B,C,D)，FDFI 算法只需 (A)、(A,B)、(A,B,C)、(A,B,C,D) 四次搜索即可发现，而宽度优先搜索算法则对除此之外的 (A,C)、$(A,D)\cdots(B,C,D)$ 等候选项也要进行考察. 从数据扫描量上分析，FDFI 算法通过构造 Item-TID 映射，仅需一次数据库扫描，就可计算所有候选项目集的支持度，从而大大减少了数据扫描量，节省了磁盘 I/O 时间. 虽然构造 Item-TID 映射需要耗费一定的内存空间，但随着计算机硬件的发展，这种“以空间换时间”的方法是现实可行的.

为了验证 FDFI 算法的有效性、可扩展性与先进性，我们在 Pentium Ⅲ 800MHz 处理器，128M RAM, Windows XP 操作系统的计算机上用 Visual C++6.0 实现了 FDFI 算法，利用 http://www.ics.uci.edu/~mlearn/MLSummary.html 上的蘑菇数据库(mushroom database)进行了实验. 该数据库有 8 124 条记录，记录了蘑菇的帽子形状、帽子的颜色、颈的形状、颈的颜色、气味、生存环境、是否有毒等 23 种属性，每种属性有 2 到 12 个枚举值. 而其中的属性 stalk-root 有 2 480 个样本缺省，另外 veil-type 属性有两个枚举值 partial 和 universal，而所有样本值均为 partial，故对这两个属性不予考虑. 实验结果如下图所示，显示了在不同数据量、不同的最小支持度(分别为 10%、5%、2%)下算法的性能变化. 实验结果表明，相同的数据量下，最小支持度越低，候选项目集将越多，耗费时间越多，同一支持度下，算法的执行时间随数据量近似于线性增长，显示了 FDFI 算法具有较好的扩展性. 试验过程中还发现，算法的复杂性一般不在于数据集(数据库)中的元组(记录)的数量，而在于属性的个数. 频繁集的数量关于属性个数呈指数式增长，这种增长可能才是算法复杂性的真正源泉.

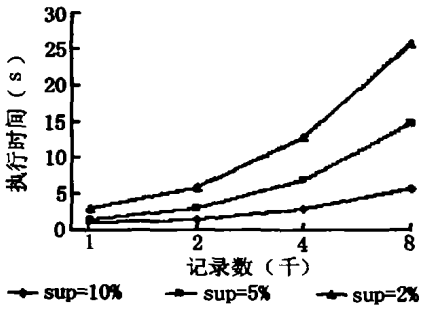

图 4　FDFI 算法性能分析

5　小　结

本文提出了一种有效的快速发现频繁项目集的算法 FDFI,利用深度优先搜索的特点,结合频繁项目集的性质,有效地缩小了搜索空间,并采用独特的支持度计数策略,只需一次数据库扫描,就可计算所有项目集的支持度,大大减少了数据扫描量.理论分析和实验结果表明,相对于现有常用的发现频繁项目集算法,FDFI 具有更优越的性能.从而为海量数据库中挖掘关联规则和序贯模式提供了有力的支持.

参　考　文　献

[1] Chen M S, Han Jiawei, Yu P S. Data Mining: An Overview from a Database Perspective. IEEE Trans on Knowledge and Data Engineering, 1996, 8(6): 866－883

[2] Agrawal R, Srikant R. Fast Algorithms for Mining Association Rules. In: Bocca J B, Jarke M, Zaniolo C, eds. Proc of the 20th International Conference on Very Large Databases. San Francisco: Morgan Kaufmann Publishers, 1994, 487－499

[3] Agrawal R, Srikant R. Mining Sequential Patterns. In: Yu P S, Chen A L P, eds. Proc of the 11th International Conference on Data Engineering. Los Alamitos, CA, IEEE Computer Society, 1995, 3－14

[4] 路松峰,卢正鼎.快速开采最大频繁项目集.软件学报, 2001, 12(2): 293－297

[5] 毛国君,刘椿年.基于项目序列集操作的关联规则挖掘算法.计算机学报, 2002, 25(4): 417－422

[6] Han Jiawei, Kamber M. Data Mining: Concepts and Techniques. San Francisco California: Morgan Kaufmann Publishers, 2001, 225－268

[7] Agrawal R, Lmielinski T, Swami A. Mining Association Rules between Sets of Items in Large Databases. In: Proc of the ACM SIGMOD Conference on Management of Data. Washington D C, 1993, 207－216

[8] Agarwal R C, Agarwal C C, Prasad V V V. Depth First Generation of Long Patterns. In: Proc of the 6th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining(KDD 2000). Boston, MA, USA, 2000, 108－118

AN ALGORITHM OF FAST DISCOVERY FREQUENT ITEMSETS

Bai Shilei, Mao Xuemin

(*Department of Automation, University of Science and Technology of China, Hefei* 230027)

(*Institute of Intelligent Machines, Chinese Academy of Sciences, Hefei* 230031)

Wang Rujing, Xiong Fanlun

(*Institute of Intelligent Machines, Chinese Academy of Sciences, Hefei* 230031)

ABSTRACT

To discovery frequent itemsets is the pivotal question of many kinds of data mining. Many algorithms have been proposed based on Apriori method, which need iterations to the database. Recent algorithms attempted to improve the mining efficiency reducing the number of database passes to control the I/O cost. An algorithm FDFI (fast discovery frequent itemsets) is presented in this paper. Using depth-first searching and cooperated with the properties of frequent itemsets, FDFI can reduce the searching space more efficiently. At the same time, particular support-count strategy based TID-mapping structure is introduced. Via TID-mapping the support of all itemsets can be computed with only one time database scanning. Thus FDFI can decrease data scanning and disk I/O time sharply. Finally the effectiveness of this algorithm is analyzed and some experimental results are given. The experiments show that FDFI is efficient, especially when the length of frequent itemsets is higher.

Key Words　Data Mining, Frequent Itemsets, Depth-First Search, Item Identification Mapping