快速关联规则挖掘算法研究*)

高 俊[1] 施伯乐[2]

(上海应用技术学院计算机系 上海200233)[1] (复旦大学计算机与信息技术系 上海200433)[2]

摘 要 本文在分析FP-growth关联规则挖掘算法的基础上,提出了一种称之为MFP的算法,给出了算法的工作原理。MFP算法能在一次扫描事务数据库过程中,把该数据库转换成MFP树,然后对MFP树进行关联规则挖掘。

关键词 关联规则挖掘,FP-growth算法,MFP算法

Research on Fast Association Rule Mining Algorithm

GAO Jun[1] SHI Bai-Le[2]

(Dept. of Computer Science, Shanghai Institute of Technology, Shanghai 200233)[1]

(Dept. of Computing and Information Technology, Fudan Univ., Shanghai 200433)[2]

Abstract Based on fully analyzing the FP_growth, an association rule mining algorithm, this paper present a new association rule mining algorithm called MFP. The MFP algorithm can convert a transaction database into a MFP tree through scanning the database only once, and then do the mining of the tree.

Keywords Association rule mining, FP_growth algorithm, MFP algorithm

1 引言

关联规则挖掘是数据挖掘领域的一个重要组成部分,用于从给出的大量实例信息中发现蕴含其中的关联规则[1]。关联规则挖掘应用最多的是挖掘布尔型关联规则,其挖掘过包括三个主要阶段。首先是将现有的数据库转换成事务数据库的形式(transaction database),然后采用适当的算法从事务数据库中挖掘出所有的频繁模式,最后由频繁模式生成有价值的关联规则[2]。其中的第二阶段是关键,它将决定关联规则挖掘的正确性和挖掘的效率。目前,挖掘频繁模式的经典算法是Apriori算法和FP_growth算法[3]。与Apriori算法相比,FP_growth算法的关联规则挖掘的时间效率有较大提高。有关研究表明:在相同的软、硬件环境下,FP_growth算法的关联规则挖掘速度比Apriori算法约高一个数量级;被挖掘的事务数据库越大,FP_growth算法的挖掘效率越高[4]。其主要原因是:通常事务数据库的数据量很大,因而扫描事务数据库的开销成为整个关联规则挖掘的主要部分;Apriori算法需要多次扫描事务数据,而FP_growth算法只需两次扫描事务数据库。

FP_growth算法用于从事务数据库中挖掘布尔型关联规则的频繁模式[5]。它的整个挖掘过程比较复杂,但是可以简单地划分成三个基本步骤[6]:首先是扫描事务数据库,根据给出的*min_sup*(最小支持度阈值)建立L表;然后第二次扫描事务数据库,依据L表,构建FP-tree;最后对构建的FP-tree进行挖掘,找出所有的频繁模式。有了频繁模式就可以根据行业背景方便地建立所需的关联规则[7]。

尽管FP_growth算法的关联规则挖掘效率比Apriori算法高,但是它仍然需要扫描两次事务数据库。第一次扫描事务数据库,得到L表;第二次扫描事务数据库,构造出FP-tree。由于扫描实际的事务数据库的开销很大,若能在此基础上再减少挖掘算法对事务数据库的扫描次数,则能进一步有效地提高关联规则挖掘效率。为此,我们设计了一种称之为MFP的快速关联规则挖掘算法。MFP算法有两个基本步骤:一是扫描事务数据库,在扫描过程中就把事务数据库转换成类似于FP-tree的树(在下面称为MFP树),并且保留了所有事务数据库中item间的关联信息;二是挖掘MFP树,从中找出所有可能的关联规则。与FP_growth算法相比,MFP算法只需对事务数据库扫描一次,因而可提高关联规则挖掘的时间效率。我们查阅了近五年来的VLDB年会论文集、SIGMOD年会论文集和其他有关文献资料,尚未发现有与我们的工作类似的研究报道。

2 快速关联规则挖掘算法MFP的工作原理

2.1 相关定义

为了便于讨论,先给出MFP算法涉及到的一些主要术语的定义。

定义1(事务数据库D) 用于存储交易记录的数据库,数据库中的每一个交易记录都有唯一的标识。一个交易记录包括了一笔交易涉及到的所有*item*的有序排列。图1为一简化的事务数据库D[8]。

TID	List of item_ID's
T001	I1,I2,I5
T002	I2,I4
T003	I2,I3
T004	I1,I2,I4
T005	I1,I3
T006	I2,I3
T007	I1,I2
T008	I1,I2,I3,I5
T009	I1,I2,I3

图1 事务数据库

*)基金项目:上海市高等学校科技发展基金项目(03HK08)。高 俊 副教授,理学硕士,主要研究领域为数据挖掘、数据库。施伯乐 首席教授,博士生导师,主要研究领域为数据库、知识库。

其中 $T001$ 为事务数据库的首记录标号，$T001$ 相对应的交易包括了 $I1$，$I2$，$I5$，记录中的 *item* 按标号的顺序排列。

定义2 MFP 树由一个标号为 *null* 的根结点和数个树结点构成，每一个结点可带有 n 个树结点（$n=0,1,2,\cdots$）。当 $n=0$ 时，该结点称为叶结点。由于 MFP 树用于表示一事务数据库，树的结点用数据库中的 *item* 的标号表示。除根结点外，每一个结点由 $Ii.count$ 和 $Ii.pointer$ 二个域构成。其中 $Ii.count$ 为该结点上出现的相同 *item* 个数，标示在结点标号 Ii 右侧一括号中；$Ii.pointer$ 为指向其父结点的指针。

定义3（路径结点组合 C） 它是 MFP 树的一叶结点到根 *null* 的路径上所有结点（不包括根结点）的一个任意组合，由构成 C 的结点标识，在标识符右侧用括号中的一数值表示该组合的记数值，该记数值等于叶结点的 $Ii.count$ 值。设一 MFP 树如图2所示。

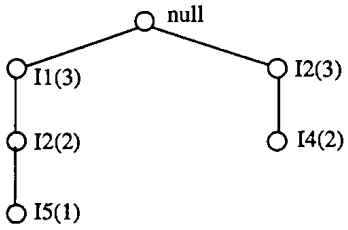

图2 MFP 树及路经组合

$I4$，$I5$ 是叶结点，分别形成二条到根结点的路径。由于 $I5.count=1$，$I5$ 路径上的结点为 $I1$、$I2$、$I5$，所以三个结点形成四个组合，四个组合分别是：$\{I5,I2\}_{(1)}$，$\{I5,I1\}_{(1)}$，$\{I5,I2,I1\}_{(1)}$，$\{I2,I1\}_{(1)}$；$I4$ 路径上的结点为 $I2$、$I4$，$I4.count=2$，则二个结点形成的组合是：$\{I4,I2\}_{(2)}$。

定义4 表 TL 是一指针队列，表中每一个元素均指向一个 MFP 树的叶结点，指针与叶结点一一对应。

定义5 候选频繁模式集 CF，集合元素为 C。

2.2 算法的工作原理

MFP 算法分为构造 MFP 树和挖掘 MFP 树二部分。

（1）构造 MFP 树 首先创建 MFP 树的根结点 *null*，然后取事务数据库 D 的第一条记录，并将其插入到 MFP 树中。插入过程为：取出记录中的第一个 *item*，若 MFP 树的根结点存在与该 *item* 标号相同的子结点，则在该子结点的记数值（count 域）上加1，否则在根结点下创建一新的子结点，以该 *item* 的标号作为新创建子结点的标识；然后以被插入的结点为子树的根，将记录中的下一个 *item* 按上述步骤插入到子树中；如此重复，使得记录中的所有 *item* 都被插入到 MFP 树中，完成一条记录插入过程。接着，再从事务数据库 D 中取下一条记录，重复上述记录插入过程，直到数据库中的记录全部被插入到 MFP 树中。由此，经过一次扫描把事务数据库 D 转换成 MFP 树，从树叶结点到根结点的一条路径对应了数据库的一条或多条记录，并且保留了原来记录中 *item* 之间的关联信息。

为了便于后续的关联规则的挖掘，在构建树的过程中，使 TL 表的指针指向 MFP 树的叶结点，指针与叶结点一一对应。

按上述过程，可把图1所示的事务数据库 D 转换成图3所示的 MFP 树，结点标号 Ii 右侧括号中的数值为该结点的 $Ii.count$ 值。

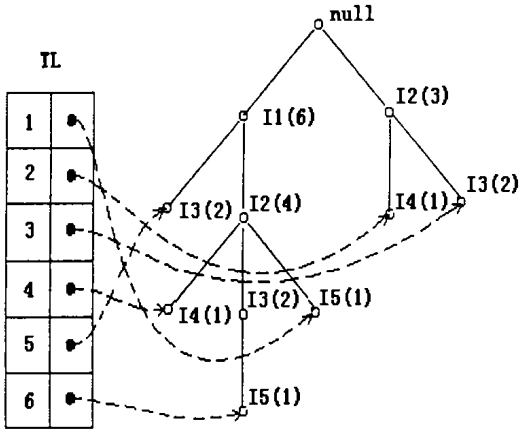

图3 事务数据库 D 转换成的 MFP 树

（2）挖掘 MFP 树 取 TL 表中的首元素，得到该元素指向的叶结点和从叶结点到根 *null* 的路径。对该路径上所有结点进行组合（根结点 *null* 除外），按定义3设置各组合的标识和记数值。然后将所有的组合送入候选频繁模式集 CF，若 CF 中已经存在相同的组合（即组合的标识相同），则进行合并。合并操作为：组合标识保持不变，记数值为二者之和。该路径上所有的组合进入候选频繁模式集 CF 后，对该路径上的结点进行修正。修正过程为：使该路径上的所有结点的 *count* 值减去叶结点的 *count* 值；减去后，若该路径上的叶结点存在不带其他子结点的父结点且父结点不是根 *null*，则将指向该叶结点的指针转为指向其父结点，以它为叶结点，对新的路径进行同样的操作。否则，删除 TL 表中的这一指针。

此后，再取表 TL 的下一元素，重复上述过程直到表中的元素都被取出，完成所有 MFP 树叶结点的处理。

最后，用给出的 *min_sup* 值（最小支持度）剔除 CF 中记数值小于 *min_sup* 的组合。这样，留在 CF 中的就是所要找的频繁模式，依此可构造出所有的候选关联规则，并可用给出的 *min_conf* 值（最小置信度）筛选出所需要的关联规则。下面是对已构造的 MFP 树（图3）进行挖掘的过程。

由 TL 表中的首元素指针找到叶结点 $I5$，从而得到路径结点 $I5$、$I2$、$I1$。对该三结点进行组合得 $\{I5,I2\}_{(1)}$，$\{I5,I1\}_{(1)}$，$\{I5,I2,I1\}_{(1)}$，$\{I2,I1\}_{(1)}$。此时 CF 为空，因而四组合直接进入 CF。然后对该路径上的结点进行修正，将 $I5$、$I2$、$I1$ 的 *count* 值减1。由于 $I5$ 的父结点 $I2$ 带有其他子结点，所以删除 TL 表中的这一指针元素。此后，再取表 TL 的首元素，重复上述过程直到 TL 表中剩下最后一元素。TL 表中最后的指针指向叶结点 $I5$，从而得到路径结点 $I5$、$I3$、$I2$、$I1$ 和该四结点的各个组合，把组合送入 CF，并修正该路径上的结点。此时由于 $I5$ 的父结点 $I3$ 无其他的子结点，所以不删除 TL 表中该指针，只是将指向 $I5$ 的指针改为指向其父结点 $I3$。然后再对以 $I3$ 为叶结点的路径（由结点 $I3$、$I2$、$I1$ 构成）进行同样的操作。最后，从 CF 中剔除记数值小于 *min_sup*（本例中 *min_sup* $=2$）的组合。对图3所示的 MFP 树挖掘后得到的 CF 如下所示：

$CF=\{\{I3,I1\}_{(4)},\{I2,I1\}_{(4)},\{I3,I2\}_{(4)},\{I5,I2\}_{(2)},\{I5,I1\}_{(2)},\{I4,I2\}_{(2)},\{I5,I2,I1\}_{(2)},\{I3,I2,I1\}_{(2)}\}$

3 MFP 算法

3.1 算法

MFP 算法的伪代码如下所示。

（下转封四）

(上接第201页)

```
输入:事务数据库 D,min_sup
输出:频繁模式集 CF
方法:(1) 构建 MFP 树
        创建根结点 null;
        打开 D;
        while(not EOF)
        {
         R=当前记录;
         call insert_tree(R,null);
         记录指针下移一位;
        }
procedure insert_tree (record,root_nod)
  {while(record()ϕ)
     {first=record 中首 item;
     从 record 中删除该 item;
     if (root_nod 中存在标号为 first 的子结点 a)
       a.count=a.count+1;
     else
       {创建标号为 first 的结点 a;
        a.count=1;
        a.pointer 指向 root_nod;}
     if(record=ϕ)
       {删除 TL 表中指向该路径上其他结点的指针;
        在 TL 表中创建指向结点 a 的指针;}
```

(2)挖掘 MFP 树(tree,TL)

```
while (TL()ϕ)
{得到 TL 首元素指向的叶结点 a;
 组合 a 到 null 路径上的结点形成 C1,C2,…,Cm;
 for (i=1; I<= m; i++)
  { Ci.count=a.count;
    Ci 标号设为构成该组合的结点的有序排列;
  if (Ci∈CF)
    CF.Ci.count= CF.Ci.count+Ci.count;
  else
    Ci 送入 CF;}
    该路径上其他结点的 count 值减去叶结点的 count 值;
  if (a 的父结点不存在其他子结点且不是根结点)
    TL 中指向 a 结点的指针指向 a 的父结点;
  else
    删除 TL 中指向 a 结点的指针;
  }
  删除 CF 中 count 值(min_sup)的组合;
```

3.2 算法的实验运行结果

为了测试MFP算法的实际运行效果,在同样的软、硬件环境下(P4-2G CPU、256MB 内存、40G 硬盘,Win2000 Server 操作系统),我们用 MFP 算法和 FP_growth 算法分别对一实验商业事务数据库进行布尔关联规则挖掘测试。该数据库约有20000条记录,涉及的 *item* 数为200左右。反复测试表明:二算法挖掘出来的候选频繁集相同,而 MFP 算法的挖掘速度约提高20%。由 MFP 算法的工作原理可知:事务数据库增大时,MFP 算法的时间效率将进一步增加。

当事务数据库很大,以至于难以在给定的计算机内存中构建一个与整个数据库对应的 MFP 树时,可以将事务数据库按一定条件横向分割成几个子数据库,然后逐一在内存中构造它们对应的 MFP 树。将各自 MFP 树的路径结点组合送入一个总的 FC。最后再对 FC 进行处理,得到所有的频繁模式。因此,MFP 算法特别适用于对大型事务数据库的关联规则挖掘。

结束语 关联规则的挖掘具有重要的实用价值,在数据挖掘领域应用广泛。由于被挖掘的事务数据库规模较大,关联规则挖掘算法的运行效率就显得特别重要。我们对 FP_growth 算法进行了深入的研究,设计了一种称之为 MFP 的关联规则挖掘算法。该算法只需对事务数据库扫描一次,就可把事务数据库转换成 MFP 树,然后对 MFP 树进行挖掘。实验表明:MFP 算法的关联规则挖掘时间效率较高。由于 MFP 算法的输入为事务数据库 D,其形式与 FP_growth 算法或 Apriori 算法的输入相同,因此,MFP 算法可应用于任何 FP_growth 算法或 Apriori 算法适用的场合。

参考文献

1 Imielinski T, Virmani A. MSQL: Aquery languang for database mining. Data Mining and Knowledge Discovery, 1999, 3: 373～408

2 Groth R. Data Mining: Building Competitive Advantage. Prentice Hall, 1999

3 Goebel M, Gruenwald L. A survey of data mining and knowledge discovery software tools. SIGKDD Explorations, 1999, 1: 20～33

4 Grahne G. Efficient mining of constrained correlated sets. In: Proc. 2000 Intl. Conf. Data Engineering (ICDE'00), San Diego: 2000. 512～521

5 Han J. Mining frequent patterns without candidate generation. In: Proc. ACM-SIGMOD Int. Conf. Dallas. 2000

6 蒋良孝. 一种基于 FP-增长的决策规则挖掘算法. 计算机科学, 2003(8): 23～25

7 Han J, Pei J. Freespan: Frequent pattern-projected sequential pattern mining: [Technical Report CMPT2000-06]. Simon Fraser University, 2000. 6～12

8 Han J. Data Mining: Concepts and Techniques. Burnaby: Simon Fraser University, 2000. 155～163

计算机科学

(1974年1月创刊)

第32卷第3期 (月刊)

2005年3月25日出版

ISSN 1002—137X

CN50—1075/TP

定价: 25.00元 国外定价: 5美元

邮发代号: 78—68

发行范围: 国内外公开

主管单位: 国家科学技术部

主办单位: 国家科技部西南信息中心

编辑出版: 《计算机科学》杂志社

重庆市渝中区胜利路132号 邮政编码: 400013

电话: (023) 63500828 E-mail: jsjkx@swic.ac.cn

网址: www.jsjkx.com

社 长: 牟炳林

主 编: 彭 丹

副主编: 朱宗元

主编助理: 徐书令

印刷者: 重庆科情印务有限公司

总发行处: 重庆市邮政局

订购处: 全国各地邮政局

国外总发行: 中国国际图书贸易总公司 (北京399信箱)

国外代号: 6210—MO