维普资讯 http://www.cqvip.com

计算机科学2002Vol. 29№. 12

频繁项集挖掘中的两种哈希树构建方法*)

Two Methods of Building Hash-tree for Mining Frequent Itemsets

杜孝平[1] 罗 宪[2] 唐世渭[1]

(北京大学信息科学中心 视觉与听觉信息处理国家重点实验室 北京100871)[1]

(重庆交通学院 重庆400074)[2]

Abstract Hash-tree is an important data structure used in Apriori-like algorithms for mining frequent itemsets. However, there is no study so far to guarantee the hash-tree could be built successfully every time. In this paper, we propose a static method and a dynamic one to build the hash-tree. In the two methods, it is easy to decide the size of hash-table, hash function and the number of itemsets stored in each leaf-node of hash-tree, and the methods ensure that the hash-tree is built successfully in any cases.

Keywords Hash-tree, Frequent itemset, Data mining, Database, Knowledge discovery

1 引言

从大型数据库中发现频繁项集/模式的研究作为关联规则[1~3]、序贯模式[4]、因果关系[5]、最大模式[6]、多维模式[7]等挖掘问题的核心,已经成为近年数据挖掘领域的研究热点,并有不少有效的挖掘算法[3,4,8]被提出。在这些挖掘算法中,它们大多数都采用了类似于 Apriori 算法[1]的方法进行频繁项集的挖掘与更新[9]。类 Apriori 算法的共同特点是:为了找出库中所有包含 $k(k>1)$个项的频繁 k-项集,首先产生包含频繁 k-项集的候补 k-项集,并将其保存在一个哈希树中,然后通过扫描数据库决定各个候补项集在库中的出现频度,最后根据预先给定的最小频度阈值从候补项集中得到频繁 k-项集。

在挖掘处理进行之前,为哈希树的构建决定合适的哈希函数、哈希表的大小和叶节点所允许存放的项集的数量,以确保哈希树的构建一次成功是保证挖掘处理能够顺利进行的关键因素之一。但是,据笔者所知,迄今尚无研究论及这个重要的问题。本文提出了两个哈希树构建方法——静态建树法和动态建树法来解决这个问题。两个方法在保证建树一次成功的前提下,还注重了决定哈希表的大小、哈希函数及叶节点所允许保存项集数量的简单化。静态建树法简单易用,而动态建树法采用有效的映射方法,大大节省了建树时系统资源的开销。

2 相关研究

2.1 挖掘频繁项集

使 $I=\{i_0,i_1,\cdots,i_{m-1}\}$表示由 m 个不同的项组成的集合,T 为 I 中不同的项所组成的一条事务,D 是由事务所组成的数据库。I 中不同的项所组成的集合称为项集,由 k 个不同的项所组成的项集称为 k-项集。事务 T 包含项集 X,当且仅当 $X\subseteq T$。项集 X 在 D 中的出现频度,δ_x,定义为 X 在 D 中出现的次数,即 δ_x 表示 D 中有多少条事务包含了项集 X。对于一个用户预先给定的一个最小频度阈值 ξ,如果 $\delta_x\geqslant\xi$,则称项集 X 为频繁项集。包含 k 个项的频繁项集称为频繁 k-项集,D 中所有频繁 k-项集的集合记为 F_k。

给定最小频度阈值 ξ 和数据库 D,挖掘频繁项集就是从 D 中找出其出现频度大于或等于 ξ 的所有项集。为了简化讨论,本文在不引起歧义的情况下,将 I 中的项直接用它的下标值表示,例如,项集$\{i_2,i_8\}$表示为$\{2,8\}$。另外,在频繁项集挖掘的研究中,拥有相同的项而排列顺序不同的项集由于它们拥有相同的出现频度,例如 $\delta_{(2,8)}=\delta_{(8,2)}$,本文仅考虑按照升序排列的项集,即$\{2,8\}$。

2.2 候补项集的产生方法

所有的类 Apriori 算法采用的都是一种通过反复"产生-测试候补项集"的方法来找出数据库 D 中所有的频繁项集。下面以 Apriori 算法为例进行说明。它在第一次扫描 D(称为循环1)时通过统计库中各个项的出现频度,得到所有的频繁1-项集,F_1。然后将 F_1 作为种子集合,在循环2生成一定包含全部频繁2-项集的候补集合(记为 C_2),再扫描 D 决定 C_2 中候补集合的出现频度,最后获得 F_2。如此循环,直至没有候补项集被产生或者频繁项集被得到为止。$C_k(k>1)$的产生是用已经得到的 F_{k-1} 与 F_{k-1} 通过连接与剪除处理来完成的,其生成方法如下[1],先由连接处理产生 C_k。然后,从 C_k 中删除所有包含有非频繁$(k-1)$-项集的候补 k-项集。

```
insert into C_k
select p.item_1, p.item_2, ..., p.item_{k-1}, q.item_{k-1}
from F_{k-1} p, F_{k-1} q
where p.item_1 = q.item_1, ..., p.item_{k-2} = q.item_{k-2}, p.item_{k-1} <
      q.item_{k-1}
```

2.3 哈希树的结构及构建

除第一个循环以外,Apriori 算法在每个循环都把候补项集存放在一棵哈希树中,以便能对其进行快速查找和出现频度计算。一棵哈希树由根节点、内部节点和叶节点所组成,根节点的深度定义为1;所有的候补项集都保存在叶节点上,每个叶节点可保存几个候补项集;树的每个内部节点都包含一个哈希表,深度为 d 的内部节点所属哈希表的每一个桶指向深度为 $d+1$的另外一个内部节点或叶节点;在初始构建哈希树时,所有的节点都视为叶节点,当向树中插入一个候补 k-项集 $X=\{x_1,x_2,\cdots,x_k\}$时,从根节点向下直到找到一个叶节点。在深度为 d 的内部节点,通过对 X 中的第 d 个项,$x_d(1\leqslant d\leqslant k)$,用哈希函数进行哈希处理来决定指向深度为 $d+1$

*)本文的研究得到国家重点基础研究发展规划(973)项目(No. G1999032705)和留学回国人员科研启动基金资助。

维普资讯 http://www.cqvip.com

的哪一个内部节点或叶节点。当叶节点中被插入的候补项集的数量超过某个阈值时，该叶节点被转换为一个内部节点。图1给出了6个候补3-项集被插入哈希树后的结果示意图。图中$\mathcal{N}(x_d)$为项x_d在集合I中的下标值。例如，当向哈希树中插入候补项集$\{i_1,i_3,i_4\}$(即集合$\{1,3,4\}$)时，由于$f(i_1)=1$，根节点的桶1所指向的是一个内部节点，再哈希第二个项得$f(i_3)=3$，深度为2的内部节点的桶3所指为空，可以存放候补项集，所以$\{1,3,4\}$被插入到桶3所指叶节点里而不用哈希i_4。

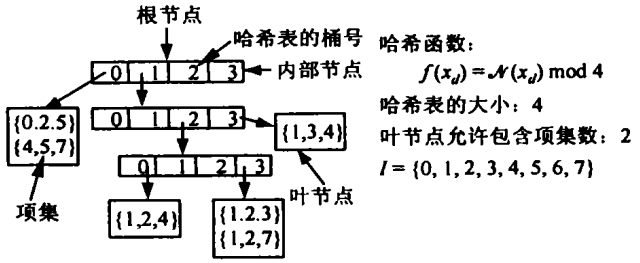

图1　样例哈希树

2.4　存在的问题

一般地，对于存储k-项集的哈希树，其最大树深不能超过k。否则项集的插入将失败，因为k-项集中最多只有k个项用于决定不同深度的哈希表的分枝。如果在图1所示例子中再插入$\{1,6,7\}$，那么哈希树的构建将会失败：由于最终定位的叶节点已经存放着$\{1,2,3\}$和$\{1,2,7\}$，其项集数量已经达到规定的阈值2且树深也已达到极限值，不能再转换为内部节点。这样的情形称为候补集合的插入溢出。不避免建树时插入溢出的产生，就无法保证频繁项集挖掘处理的成功。本文在下一节给出两个解决方案。

3　哈希树构建法

3.1　静态建树法

在频繁项集挖掘处理中，不同的最小频度阈值将会导致不同候补项集的产生。为了保证挖掘处理能够顺利进行，哈希树的构建必须适应于最坏的情形：保证所有可能出现的项集都能被插入到哈希树中。为此，一个最直观而简单的方法是，考虑为I中所包含的每个项建立一个特定不重复的哈希值，那么，由I中的项所组成的任意两个不同的k-项集，由于不可能在哈希树的每个相同深度的内部节点都具有相同的哈希值，它们会存放在不同的叶节点上。如定理1所述：

定理1　给定由m个不同的项组成的集合$I=\{i_0,i_1,\cdots,i_{m-1}\}$，对于由$I$中$k$个项所组成的不同$k$-项集$X=\{x_1,x_2,\cdots,x_k\}$，采用Apriori算法所提供的哈希树建树原则来建立保存这些k-项集的哈希树时，如果哈希表大小为m，哈希函数为$f(x_d)=\mathcal{N}(x_d)$，x_d为k-项集中的第$d(1\leqslant d\leqslant k)$个项，$\mathcal{N}(x_d)$为项$x_d$在$I$中的下标值，那么哈希树的每个叶节点仅需保存一个$k$-项集，就可保证哈希树构建不产生插入溢出。

证明：假设在建树过程中产生了k-项集的插入溢出，必定存在两个不同的k-项集，$X=\{x_1,x_2,\cdots,x_k\}$和$Y=\{y_1,y_2,\cdots,y_k\}$，都插入到了树中的同一个叶节点。根据哈希树的构建方法，在该叶节点到树根之间路径上的任何一个深度为$d(1\leqslant d\leqslant k)$的内部节点上，$X$和$Y$在该内部节点上的哈希值$f(x_d)$和$f(y_d)$必定相等：$\mathcal{N}(x_d)=\mathcal{N}(y_d)$。从而知道$x_d$和$y_d$在$I$中是同一个项。据此，$X$和$Y$是同一个$k$-项集。这与$X$与$Y$是两个不同的$k$-项集的假设相矛盾，命题得证。

图2给出了一个由I=$\{0,1,2,3\}$所形成的全部2-项集插入到由定理1所述方法建立的哈希树中的例子以印证定理1的正确性。

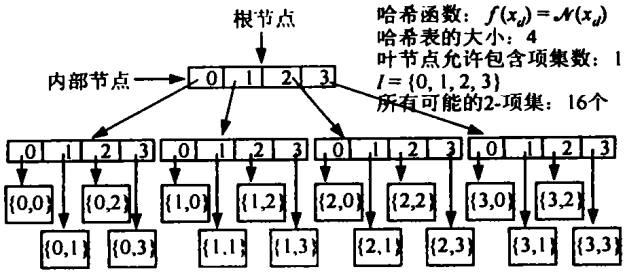

图2　全部2-项集插入哈希树中的例子

在定理1中，对于项集X中的项的顺序没有作任何限制，属于最一般化的情形，根据定理1可以得到如下推理：

推理1　在定理1的基础上，如果将保存到哈希树中的k-项集限定为由I中不同的k个项所组成的升序k-项集，用同样的哈希树的构建法也不会产生插入溢出现象。

由于这一建树方法的哈希表的大小是由数据库中所包含的不同项的个数决定的，因此称它为静态建树法。这一方法的优点是保证哈希树的构建能够一次成功，不会产生溢出现象；叶节点所存放的项集数量少，查找效率高；哈希表的大小及哈希函数的决定简单。其不足是建立哈希树所用的系统资源开销浪费巨大。因为候补项集中所包含的项仅仅占I中的少部分，并且随着所查找的频繁项集的长度的增加，其数量会急剧减少。如对拥有1000个不同项的数据库，如果在某个循环仅有10个项包含在候补项集中，则所建树的每个哈希表至少有99%的桶被闲置。接下来，我们提出一种动态建树方法来减少静态建树法所造成的资源浪费问题。

3.2　动态建树法

从对静态建树法的讨论中注意到，只有用那些出现在候补项集C_k中的项哈希到的哈希表中的桶才被使用。为此，在每次构建哈希树时，只要哈希表的大小不小于所建立的C_k中所包含的不同项的数量，并采用适当的方法保证将这些项哈希到哈希表的不同桶中，那么根据定理1和推理1可以判断哈希树的构建能够保证一次成功。

分析2.2节候补项集的产生方法可知，C_k是F_{k-1}通过连接与修剪处理产生的，C_k中所包含的不同项的总数一定不大于F_{k-1}中所包含的不同项的总数。为此，可以用F_{k-1}中所包含的不同项的总数作为构建保存C_k的哈希树中哈希表的大小，并在获得F_{k-1}的同时得到该值。为保证将F_{k-1}中的不同项哈希到哈希表的不同桶中，首先采用下述映射方法为建树做准备：

1. 将F_{k-1}中所包含的不同的项按I中坐标值的升序排列，并将其按升序从0开始用整数依次重新编号。例如，$F_2=\{\{i_3,i_5\},\{i_3,i_7\},\{i_3,i_9\},\{i_5,i_7\},\{i_7,i_9\}\}$将被排列为：$i_3,i_4,i_5,i_7,i_9$并依次重新编号为0,1,2,3,4。

2. 用一个映射表来实现它们之间的相互映射关系。上例的映射表为：

F_{k-1}中的项	i_3	i_4	i_5	i_7	i_9
新编号	0	1	2	3	4

完成上述映射处理后，在构建哈希树时，首先将候补项集中的项替换为新编号后再插入哈希树中去。在用数据库中的

维普资讯 http://www.cqvip.com

事务进行哈希树中候补项集的出现频度计算时，亦先将事务中的相关项替换为新编号后再在哈希树中进行出现频度计算处理。在进行替换编号操作时，事务中没有出现在上述映射表中的项被忽略，因为它们对候补项集的出现频度计算是无用的。最后，在出现频度计算完成后，由哈希树输出 F_k 时，再用映射表将编号映射回原来的项。

由推理1可知，用上述动态建树方法构建哈希树时，可以保证不会产生溢出现象。这一方法大大节约了系统资源的使用，因为哈希表几乎没有被闲置的桶。例如，在上面所举例子中，如果 I 中所包含的项也为1000的话，那么同静态建树法相比，在每形成一个内部接点时，其哈希表所使用的空间将由静态法的1000个单元空间变为动态法的5个单元空间，仅为5‰。

3.3 两种建树法的比较

静态建树法与动态建树法各有其优势与不足。它们的共同特征是：

1. 哈希表大小和哈希函数的决定方法简单且实用。不需要针对不同的挖掘处理进行测试与调整。

2. 叶节点所允许保存的项集数量的最小值都可设定为一个，使得对哈希树的查找处理快速而有效。

3. 针对任何情形下的频繁项集查找处理，都能保证哈希树的构建一次成功，避免了挖掘处理因哈希树的构建失败而失败的出现可能。

除此之外，它们分别还拥有其各自的不同特征，从而导致它们适用于不同的情形的挖掘处理。

对于静态建树法，其哈希表的大小由数据库中所包含的不同项的总数决定。建树及对树处理时可以直接利用各个项在 I 中的下标值，处理方便迅速。其缺点是树中各个内部节点所用哈希表中太多的桶被闲置，造成大量系统资源的浪费。因此，静态建树法适用于数据库较小、库中所含总的不同项的数量较少且系统资源充足，保证有足够的内存供挖掘处理所使用的频繁项集挖掘情形。这是一个以空间换时间的建树方法。

而与静态建树法相对应，动态建树法是根据挖掘处理过程中实际需要处理的项动态地调整其哈希表的大小。这使得哈希树构建所占用的系统资源大大减小，各个内部节点所用哈希表中桶的利用率大大增加。但是它为此必须另外建立一个项与哈希表中所用代码之间的映射表，从而较静态建树法增加了额外的映射处理开销。这是一个以时间换空间的建树方法。它适用于大数据库、库中不同项多且系统资源及可用内存空间有限的频繁项集挖掘情形。

结束语 针对目前广泛使用的类 Apriori 频繁项集挖掘算法中所用哈希树构建上存在的问题，本文提出了两个建树方法--静态建树法和动态建树法，来确保挖掘过程中哈希树的构建简洁、有效并且一次成功。本文详细叙述了静态和动态建树方法，并证明了所提方法的正确性。在分析其各自的特征、优势与不足的基础上，给出了它们各自适用的挖掘处理情形。

参 考 文 献

1 Agrawal R, Srikant R. Fast Algorithms for Mining Association Rules. In VLDB'94. 487～499

2 Du X P, Kaneko K, Makinouchi A. Fast Algorithm to Find Frequent Itemsets for Mining of Association Rules. In IS'00. 408～414

3 Han J, Pei J, Yin Y. Mining Frequent Patterns without Candidate Generation. In: Proc. 2000 ACM-SIGMOD Int. Conf. on Management of Data, Dallas, TX, May 2000. 1～12

4 Agrawal R, Srikant R. Mining Sequential Patterns. In: ICDE'95. 3～14

5 Silverstein C, Brin S, Motwani R, Ullman J. Scalable Techniques for Mining Causal Structures. In: VLDB'98. 594～605

6 Bayardo R J. Efficiently Mining Long Patterns from Databases. In: SIGMOD'98. 85～93

7 Kamber M, Han J, Chiang J Y. Metarule-guided Mining of Multi-dimensional Association Rules Using Data Cubes. In: KDD'97. 207～210

8 Park J S, Chen M S, Yu P S. An Effective Hash-based Algorithm for Mining Association Rules. In: Proc. ACM SIGMOD Int. Conf. Management of Data, San Jose, CA, May 1995. 175～186

9 Du X P, Makinouchi A. Ameliorated Algorithm to Maintain Discovered Frequent Itemsets. Res. Rep. ISEE Kyushu University, 2001, 6(1): 19～24

（上接第128页）

X. H. Hu 的约简算法得到的结果同样是{a,b,c,d}；而基于信息熵的算法的约简结果为{a,b,c,g}，可以验证是独立约简，但显然不是最小约简（{b,c,d}才是最小约简）。

接下来利用本文给出的新算法计算约简。

l 和 r 的选取是本算法的关键。l 与 r 差距太大则体现不出该方法的特点，太小了则必须以巨大的计算量和存储量为代价；属性的个数以及核属性的个数都影响着 l 和 r 的取值。本例中我们取 $l=3, r=1$。仍考虑 $M(IS)'=\{ab,ac,ad,be,cf,dg,abd,bf,cg\}$ 显然核为空。

先用 SFS 法添加3个属性，则

$B=\{a,b,c\}$。

计算 a,b,c 的重要性。

$Insig(a,B)=card(\{ab,ac,be,cf,abd,bf,cg\})=7$

$Insig(b,B)=card(\{ab,ac,ad,cf,abd,cg\})=6$

$Insig(c,B)=card(\{ab,ac,ad,be,abd,bf\})=6$

相对来说，a 是最不重要的一个，所以 $B=\{b,c\}$，$M(IS)'=\{ad,dg\}=TM(IS)$。

$M(IS)'$ 中还剩3个属性，用 SFS 法计算它们出现的频数，$P(a)=1, P(d)=2, P(g)=2$。所以选 d，从而 $B\{b,c,d\}$，此时 $TM(IS)$ 为空。显然 B 就是约简，并且是最小约简。

在表1中，单个属性来看，a 是相当重要的，所有的算法都首先选择。随着属性的增加，后入选的属性的组合对于分类必不可少，同时可以取代 a 的作用，使其由最重要变为冗余。传统的算法对此无能为力，而本文提出的带局部回溯的约简策略则为解决这个问题提供了一种新思路。

结束语 本文提出了一种带局部回溯策略的属性约简新算法，克服了约简过程中属性一旦选入，即使变得冗余，也无法剔除出去的弊端，从而在更多的情况下可以找到最小约简。

回溯过程中 l 和 r 的选择还有待进一步研究，既要考虑未入选属性间的相关性，又不至于引起计算量过大，需要对二者合理兼顾；如果能在约简过程中动态地选择、调整，结果应该更为理想。这些都有待进一步的研究。

参 考 文 献

1 边肇祺，张学工，等. 模式识别. 清华大学出版社，2000. 202～204

2 于洪，杨大春，王国胤，等. 基于 Rough Set 理论的知识约简算法. 计算机科学，2001，28(5. 专刊)：31～34

3 苗夺谦，王珏. 粗糙集理论中知识粗糙性与信息熵关系的讨论. 模式识别与人工智能，1998，11(1)：34～40

4 Pawlak Z. Rough Set-Theoretical Aspects of Reasoning about Data. Kluwer Academic Publishers, Dorderecht, Boston, London, 1991

5 Polkowski L, Skowron A. Rough Sets: Perspectives, Rough Sets in Knowledge Discovery. In: Polkowski L, Skowron A, eds. Physica-Verlag, 1998. 1～27