基于 FP-Tree 有效挖掘最大频繁项集*

颜跃进+, 李舟军, 陈火旺

(国防科学技术大学 计算机学院,湖南 长沙 410073)

Efficiently Mining of Maximal Frequent Item Sets Based on FP-Tree

YAN Yue-Jin+, LI Zhou-Jun, CHEN Huo-Wang

(School of Computer Science, National University of Defense Technology, Changsha 410073, China)

+ Corresponding author: Phn: +86-731-4532956, Fax: +86-731-4573637, E-mail: yyj@nudt.edu.cn, http://www.nudt.edu.cn

Received 2004-01-06; Accepted 2004-06-10

Yan YJ, Li ZJ, Chen HW. Efficiently mining of maximal frequent item sets based on FP-Tree. *Journal of Software*, 2005,16(2):215−222. http://www.jos.org.cn/1000-9825/16/215.htm

Abstract: During the process of mining maximal frequent item sets, when minimum support is little, superset checking is a kind of time-consuming and frequent operation in the mining algorithm. In this paper, a new algorithm FPMFI (frequent pattern tree for maximal frequent item sets) for mining maximal frequent item sets is proposed. It adopts a new superset checking method based on projection of the maximal frequent item sets, which efficiently reduces the cost of superset checking. In addition, FPMFI also compresses the conditional FP-Tree (frequent pattern tree) greatly by deleting the redundant information, which can reduce the cost of accessing the tree. It is proved by theoretical analysis that FPMFI has superiority and it is revealed by experimental comparison that the performance of FPMFI is superior to that of the similar algorithm based on FP-Tree more than one time.

Key words: maximal frequent item set; frequent pattern tree; superset check; maximal frequent item sets projection

摘 要: 最大频繁项集的挖掘过程中,在最小支持度较小的情况下,超集检测是算法的主要耗时操作.提出了最大频繁项集挖掘算法 FPMFI(frequent pattern tree for maximal frequent item set)使用基于投影进行超集检测的机制,有效地缩减了超集检测的时间.另外,算法 FPMFI 通过删除 FP 子树(conditional frequent pattern tree)的冗余信息,有效地压缩了 FP 子树的规模,减少了遍历的开销.分析表明,算法 FPMFI 具有优越性.实验比较说明,在最小支持度较小时,算法 FPMFI 的性能优于同类算法 1 倍以上.

关键词: 最大频繁项集;频繁模式树;超集检测;最大频繁项集投影

中图法分类号: TP311 **文献标识码**: A

在关联规则挖掘、序列模式挖掘、相关性挖掘、多层模式挖掘等数据挖掘问题中,挖掘频繁项集既是基本

* Supported by the National Natural Science Foundation of China under Grant Nos.90104026, 60073001 (国家自然科学基金); the National High-Tech Research and Development Plan of China under Grant No.2002AA144040 (国家高技术研究发展计划(863))

作者简介: 颜跃进(1976−),男,湖南永州人,博士生,主要研究领域为数据仓库,数据挖掘;李舟军(1963−),男,教授,博士生导师,主要研究领域为计算机软件理论,数据仓库,数据挖掘;陈火旺(1936−),男,教授,博士生导师,中国工程院院士,主要研究领域为计算机软件理论,软件工程.

步骤,也是关键步骤[1,2].挖掘频繁项集的缺点是,要考虑太多的候选项集(大小为 l 的频繁项集的 2^l 个候选子集都要考虑).由于最大频繁项集已经隐含了所有频繁项集,而且,许多数据挖掘应用也只需要挖掘最大频繁项集,而不用挖掘所有的频繁项集,因而近些年来,很多人开始了对最大频繁项集挖掘问题的研究[3−12].

已有最大频繁项集挖掘算法可以按照文献[3]中提出的搜索空间树的遍历策略分为宽度优先算法和深度优先算法两种.算法 MaxMiner[3],DMFI[4]和 DMFIA[5]属于宽度优先算法.其中,MaxMiner 采用动态排序的方法来保证高效的前瞻剪枝(lookahead pruning),减少了遍历的结点数,降低了搜索时间,其缺陷是没有利用自顶向下的信息和未对 MFCS 进行适当的排序[4,5].算法 DMFI 和 DMFIA 针对 MaxMiner 的缺陷进行了改进,其中算法 DMFIA 通过两次扫描建立 FP-tree 来压缩表示事务数据库中与频繁项集相关的信息,从而避免了算法 DMFI 中对数据库的多遍扫描.算法 Mafia[6],GenMax[7],MinMax[8],SmartMiner[9]和 FpMax[10]是挖掘最大频繁项集的深度优先算法.Burdick 等人在算法 Mafia 中沿用了算法 DepthProject[11]的事务数据库和项集的位图表示方法,除了传统的前瞻剪枝外,Mafia 新采用一种称为父等价(parent equivalence)的方法来做进一步的剪枝;Gouda 和 Zaki 提出的 GenMax 算法使用父结点与子结点的事务标识差集来表示事务数据库,这在稀疏事务数据库情况下减少了事务标识集合的内存开销.另外,Genmax 算法提出使用局部最大频繁项集用于超集检测(superset checking),在一定程度上降低了前瞻剪枝的开销;MinMax 使用垂直 tid-list 数据格式,通过简单的 tid-list(transaction id list)交运算计算支持度,不但使用了传统的几种高效剪枝策略,还提出了一种新颖的多层回溯剪枝策略,进行了更有效的剪枝;SmartMiner 使用一种称为尾信息的数据结构来指导挖掘过程,能充分利用已挖掘出来的最大频繁项集中的相关信息.虽然 SmartMiner 不需要超集检测,但是其尾信息的建立和访问代价不菲.另外,它没有对事务数据库投影进行压缩表示;FpMax 算法是有效的基于 FP-Tree 挖掘最大频繁项集的深度优先算法,它使用 FP-tree 来压缩表示事务数据库中与频繁项集相关的信息,并将最大频繁项集保存在一棵 MFI-tree(maximal frequent item sets tree)中,在一定程度上提高了最大频繁项集的存取速度.另外,算法 Par-MinMax[12]是基于算法 MinMax 并行化挖掘最大频繁项集的算法,它根据等价类划分搜索空间,按某种权值把搜索代价分配到各处理器上,为保证各处理器能够独立工作,它在初始化时还有选择地复制了事务数据库中的部分事务.算法 Par-MinMax 的贡献在于它是并行化挖掘最大频繁项集的首次尝试.

已有算法在挖掘最大频繁项集的过程中,大都需要进行超集检测.例如,DMFIA 在添加新的最大频繁候选项集时,需要判断该候选项集在已有最大频繁项集集合和已有最大频繁项集候选项集集合中是否存在超集[5],这时需要把待测候选项集与这两个项集集合中的每个项集逐一做项匹配,直到找到一个超集为止;而为了进行有效的前瞻剪枝,FpMax 需要在 MFI-tree 中去寻找超集.所有这些超集检测操作都需要将给定项集集合中每一个项集与被检测项集做项匹配,所以在已有最大频繁项集数目较大的时候,超集检测将成为算法的瓶颈.事实上,我们经过实验也发现,在支持度较低、最大频繁项集数目庞大的情况下,已有同类算法的缺陷就是超集检测耗时太大,成为算法中的主要耗时操作.本文提出的 FPMFI(frequent pattern tree for maximal frequent item sets)算法基于 FP-Tree 框架,将已有最大频繁项集集合组织成一棵 MFI-Tree,并采用有效的投影机制,缩减了超集检测的时间开销.另外,我们通过删除 FP 子树中与最大频繁项集无关的冗余信息,压缩了 FP 子树的规模,减少了 FP 子树的数目.最后,本文通过对算法中超集检测的复杂性分析和与同类算法的实验比较验证了 FPMFI 算法的优越性.

1 相关知识

1.1 频繁项集和最大频繁项集

设 $I=\{i_1,i_2,\ldots,i_n\}$ 是 n 个不同项目的集合.如果对一个集合 X,有:$X\subseteq I$ 且 $k=|X|$,则 X 称为 k 项集,或者简单地称为一个项集.记 D 为事务 T 的集合,$T\subseteq I$.对于给定事务数据库 D,定义 X 的支持度为 D 中包含 X 的事务个数,记为 $Sup(X)$.用户可自定义一个小于$|D|$的最小支持度,记为 min_s.

定义 1. 给定事务数据库 D 和支持度 min_s,对于项集 $X\subseteq I$,若 $Sup(X)\geq$min_s,则称 X 为 D 中的频繁项集.

定义 2. 给定事务数据库 D 和支持度 min_s,对于项集 $X\subseteq I$,若 $Sup(X)\geq$min_s,且对$\forall(Y\subseteq I\wedge X\subset Y)$,均有

$Sup(Y)<\min\_s$,则称 X 为 D 中的最大频繁项集.

性质 1. 任何频繁项集的真子集都不是最大频繁项集.

性质 2. 任何频繁项集的子集都是频繁项集.

1.2 FP-Tree和算法DMFIA,FpMax

Han,Pei 等人首先使用一种称为 FP-tree(frequent pattern tree)的结构来保存原始数据库中与当前挖掘过程相关的频繁信息.FP-tree 中每个结点对应一个项.树中每一条从根结点到某个子孙结点的路径表示一个项集,每个结点记录了根结点到该结点的路径对应项集的支持度.由于某些项集前部项的重叠,FP-tree 可以通过共享这些重叠的项来达到压缩保存的目的.FP-tree 中还有头表,它记录了相应频繁项的名称和支持度.FP-tree 树体中相同项的指针链接在一条链上,链首指针也保存在头表中.FP-tree 的建立需要两次扫描原始数据库,FP 子树的建立需要两次扫描搜索空间中父结点的 FP 子树(conditional frequent pattern tree)中相应的条件模式基.FP-tree 结构细节及相关概念详见文献[13].

DMFIA 算法采用自顶向下和自底向上的双向搜索策略的宽度优先算法,采用最大频繁候选项集集合 $MFCS_D$ 来保存自顶向下的信息.分批计算 $MFCS_D$ 中项集的支持度,对于里面的频繁项集,将其加入已有最大频繁项集集合 MFS_D 中;而对于不频繁的项集,通过每次删除该项集中的一个项来产生新的候选.对于每删除一个项得到的项集,在确定其为新候选之前,需要在 MFS_D 和 $MFCS_D$ 中为它寻找超集,若超集存在,它就不是 $MFCS_D$ 的一员,进行剪枝;否则,将其加入 $MFCS_D$ 中.DMFIA 算法与 DMFI 算法的显著不同在于,它不是基于原始数据库,而是基于 FP-tree 树来进行支持度计算的.整个算法初始化时将原始数据库压缩到一棵 FP-tree 中,计算支持度通过访问该树中相应的条件模式基来进行.基于该 FP-tree 树,DMFIA 把 DMFI 算法中的多遍对原始数据库的访问转化为对内存中 FP-tree 的访问,避免了对硬盘上原始数据库的多遍扫描.它的缺点与所有候选产生测试(candidate generation-and-test)方法一样需要维护大量的候选项集.算法 DMFIA 的细节见文献[5]中算法 1.

FpMax 算法也是基于 FP-tree 的算法,在深度优先遍历搜索空间树到达某个结点时,它不仅保存了基于原始数据库建立的 FP-tree,也保存了该结点到根结点搜索路径上的每一个结点对应的 FP 子树.这些子树压缩表示了与相关结点挖掘有关的频繁信息.在当前结点上,通过在相应项集里添加对应 FP 子树头表里某个项来生成搜索空间里的子结点.在构建子结点的 FP 子树之前做超集检测,如果在已有最大频繁项集集合里存在首尾项集并集的超集,则进行前瞻剪枝;否则,创建子结点 FP 子树,递归调用算法在该子结点上进行挖掘,直至某个子孙结点的 FP 子树是单路径树.某个结点的 FP 子树为单路径树表明,该结点对应项集与 FP 子树头表项集的并集为最大频繁项集,将其加入 MFI-tree.

最大频繁项集树 MFI-tree 是算法 FpMax 中用来压缩保存已有最大频繁项集的数据结构.它的结构与 FP-Tree 一样,包括头表和树体,每条从根结点到某个叶结点的路径表示一个最大频繁项集,中间结点只记录该结点到根结点的路径长度.在超集检测时,可以通过头表方便地访问到可能包含待测项集的最大频繁项集对应路径,然后自底向上与待测项集进行项匹配.例如,对于图 1(a)中的 3 个最大频繁项集$\{a,b,c,e,f,g\}$,$\{a,b,d,e,f,g\}$和$\{a,c,d,e,g\}$,可以组织成如图 1(b)所示的 MFI-Tree.另外,对于项集 $h=\{g,d\}$,h 的 FP 子树头表项集 $t=\{a,b,c\}$时的超集检测需要遍历该 MFI-tree 中 3 条含项 g 的叶结点到根结点的路径来寻找项集 $h\cup t$ 的超集.FpMax 算法具体流程及 MFI-tree 结构详见文献[10].

2 基于 FP-Tree 挖掘最大频繁项集的算法 FPMFI

给定 D 和最小支持度 $\min\_s$,对于 D 中项集 h,记 MFI_h 为构建 h 的 FP 子树时已有的最大频繁项集集合,t 为 h 的 FP 子树头表中所有项的集合,$|h|$为项集 h 的大小.另外,假设项集中的元素都按 FP-Tree 中项的顺序排序.

在最大频繁项集的挖掘过程中,如果存在 $m\in MFI_h$,使得 $h\cup t\subset m$,根据性质 1 可知,对 $\forall\ x\subset h\cup t,x$ 都不是频繁的.这时,我们可以将 h 从搜索空间删除出去,进行前瞻剪枝.寻找同时满足条件 $m\in MFI_h$ 和 $h\cup t\subset m$ 的项集 m 的过程,我们称之为超集检测.

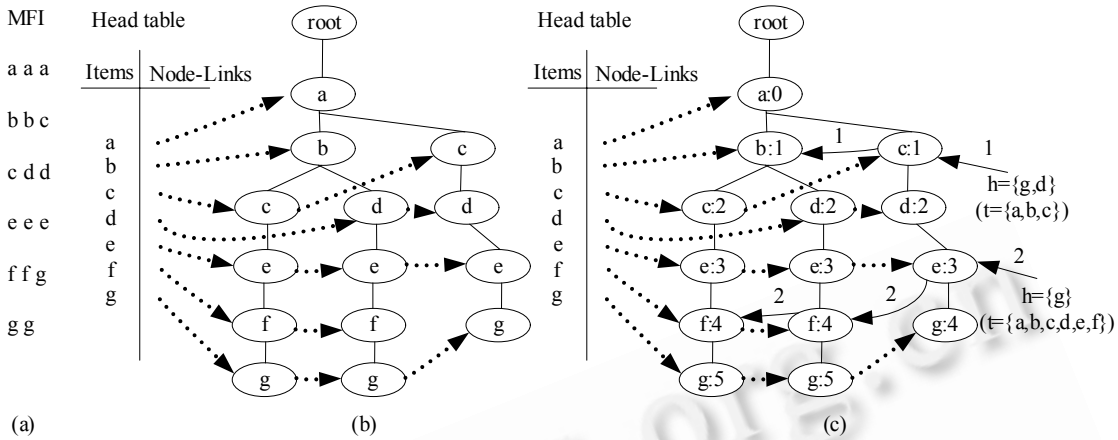

Fig.1 Examples of MFI-tree

图 1 MFI-Tree 的例子

2.1 基于最大频繁项集投影的超集检测

定义 3. $MFIP_h=\{p|p=h'\cap t\wedge h'\cup h\in MFI_h\}$,称 $MFIP_h$ 为 D 中项集 h 的最大频繁项集投影,简称投影.$MFIP_h$ 中最大的元素简称为 D 中项集 h 的最大投影.

定理 1. 记 D 中项集 h 的最大投影为 Max$p$,若$|t|=|$Max$p|$,则存在 $m\in MFI_h$,使得 $h\cup t\subseteq m$.

证明:存在 h'使得 Max$p=h'\cap t$ 且 $h'\cap h\in MFI_h$,则 Max$p\subseteq t$ 且 Max$p\subseteq h'$,又$|t|=|$Max$p|$,故有 $t=$Max$p$,$t\subseteq h'$.取 $m=h\cup h'$,即知定理 1 成立. □

定理 2. 对 $\forall x\in t$ 和 $\forall p\in MFIP_h$,设 p 排序后为 $\{p_1,p_2,\ldots,p_n\}$,如果存在 $1\le k\le n$ 使得 $p_k=x$,则 $\{p_1,p_2,\ldots,p_{k-1}\}\in MFIP_{h\cup\{x\}}$($k=1$ 时$\varnothing\in MFIP_{h\cup\{x\}}$).

证明:存在 h'使 $p=h'\cap t\wedge h'\cup h\in MFI_h$,令 $h''=h'-\{p_k\}$,t'为 $h\cup\{x\}$的 FP 子树头表中所有项的集合,显然,$h\cup\{x\}\cup h''\in MFI_h$,往证 $\{p_1,p_2,\ldots,p_{k-1}\}=h''\cap t'$. 对 $\forall p_m(1\le m\le k-1)$,有 $h\cup\{x,p_m\}\subseteq h\cup\{x\}\cup h''\in MFIP_h$,由性质 2,$Sup(h\cup\{x,p_m\})\ge$min_$s$,由 FP 子树的构造过程可知,$p_m\in t'$,又由 $p=h'\cap t$ 有 $p_m\in h'$,$p_m\in h''$,$p_m\in h''\cap t'$.又对 $\forall(y\in h''\wedge y\notin\{p_1,p_2,\ldots,p_{k-1}\})$,如果 $y\in\{p_{k+1},\ldots,p_n\}$,由 FP 子树的头表生成过程可知,$y\notin t'$;如果 $y\notin\{p_{k+1},\ldots,p_n\}$,则由 $y\notin t$ 和 $t'\subset t$ 得 $y\notin t'$.故有$\{p_1,p_2,\ldots,p_{k-1}\}=h''\cap t'$.定理 2 成立. □

由定义 1 我们知道,投影是所有最大频繁项集去掉 h 中的项后在 t 上的投影.它仅仅保存了与超集检测相关的信息.定理 1 说明,对项集 $h\cup t$ 的超集检测仅需判断项集 t 和 h 的最大投影大小是否相等即可.而定理 2 提供给我们一个由上一层最大频繁项集投影生成子层最大频繁项集投影的逐层投影方法.在 MFI-tree 中,我们很容易表示项集的最大频繁项集投影,并实现逐层投影.如图 1(c)所示,我们把每个结点到根结点的路径长度记录下来,并使用一个与 h 相关的投影结点链来将 h 的所有最大频繁项集投影保存下来.如图 1(c)所示,项集 $h=\{g,d\}$,$t=\{a,b,c\}$时和项集 $h=\{g\}$,$t=\{a,b,c,d,e,f\}$的投影结点链分别是图中实线箭头链 1 和链 2,投影就是结点链结点到根结点对应的项集,链上结点的计数就是投影大小.计算投影由过程 1 可实现.

过程 1. MfiProjection(x,$MFIP_h$).

输入:项 x, h 的投影 $MFIP_h$

输出:项集 $h\cup\{x\}$的投影 $MFIP_{h\cup\{x\}}$

(1) $MFIP_{h\cup\{x\}}=\varnothing$

(2) 对 $MFIP_h$中每个结点 n

(3) if ($n.parent.item<x$)

(4) $n=n.parent$

(5) else if (n 不在 $MFIP_{h\cup\{x\}}$中)

(6) 将 n 加入 $MFIPs[i]$中;

(7) return $MFIP_{h\cup\{x\}}$

此外,在一个新的最大频繁项集插入 MFI-tree 之后,相应的各层投影需要更新,对于给定 MFI-tree 中叶结点和投影集合,更新投影由过程 2 实现.

过程 2. Update(*leaf*,*MFIPs*).

输入:叶结点 *leaf*,投影集合 *MFIPs* //投影集合中投影按其所处层数排序

输出:更新后的投影集合 *MFIPs*

(1) n=*leaf*; i=0

(2) while (n 不是根结点 AND i<|*MFIPs*|)

(3) if (n.item<x_i) //x_i 与 $MFIPs[i]$对应的 t 项集中的最大项

(4) 将 n 加入 $MFIPs[i]$中; i++;

(5) else n=n.parent

(6) return *MFIPs*

2.2 非冗余FP子树

定义 3. 在项集 h 的 FP 子树中,称一条从根结点到某个叶结点的路径上所有项的集合为 h 的一条全路径.项集 h 的全路径 p_h 中叶结点项在 FP 子树中的计数称为全路径 p_h 的支持度,记为 $Sup(p_h)$.

定理 3. 对于项集 h 任意的全路径 p_h,如果以下两者之一成立:

(1) 存在 h 的一条频繁的全路径 p_h',使得 $p_h\subset p_h'$,

(2) 存在 $q\in MFIP_h$ 使得 $ph\subset q$,

则对$\forall(s\subset p_h)$,均有 $h\cup s$ 不是最大频繁项集.

证明:根据 FP 子树构造过程 $Sup(h\cup p_h')\geq$min_s,而根据性质 2,$Sup(h\cup q)\geq$min_s,又由(1)或(2)有($h\cup p_h$)$\subset$($h\cup p_h'$)或($h\cup p_h$)$\subset$($h\cup q$).根据性质 1 可知定理 3 成立. □

根据定理 3,我们在建立项集 h 的 FP 子树时,仅仅保存不满足定理 3 两个条件的全路径集即可.这样,可以根据频繁全路径集和最大频繁项集投影进一步压缩 FP 子树的规模,从而减少了以后各层 FP 子树的遍历开销.

定义 4. h 的满足定理 3 两个条件的全路径集称为 h 的冗余全路径集,反之称为 h 的非冗余全路径集.由 h 的所有非冗余全路径集构建的 FP 子树称为 h 的非冗余 FP 子树.

2.3 基于FP-Tree挖掘最大频繁项集的算法FPMFI

过程 3 描述了算法 FPMFI 的流程,其中的全局变量投影集合 *MFIPs* 是搜索空间中根结点到 h 对应结点路径上各结点投影的集合,h 和 *MFIPs* 初始化时都为空,MFI-tree 初始化时是只有头表的空树.与算法 FPMax 一样,算法 FPMFI 也是一个递归调用算法.调用初始条件为:T_h 为扫描原始数据库建立的非冗余 FP-tree;$MFIP_h$ 为空.调用本过程的过程在第 7 行做的超集检测说明,$h\cup t$ 在已有最大频繁项集中不存在超集,所以如果发现本过程中非冗余 FP 子树是单路径树,则算法第 2 行将 $h\cup t$ 插入 MFI-tree 并用新产生的路径来更新投影.算法第 3,4,11 行通过一个循环从下往上将 T_h 头表中的项添加到项集 h 中来生成新的考察项集.第 5 行通过扫描 T_h 中项集 h'的条件模式基寻找与 h'在一起仍构成频繁项集的项,进而建立 h'的非冗余 FP 子树头表 t'.下一层投影的生成由过程 1 在第 6 行完成.第 7 行是基于最大频繁项集投影的超集检测,如果检测成功,算法跳至第 11 行进入循环的下一步;否则,扩大 *MFIPs* 并构建 h'的非冗余 FP 子树 T_h'后递归调用 FPMFI.

过程 3. FPMFI(T_h,$MFIP_h$).

全局变量:项集 h,最大频繁项集树 MFI-tree,投影集合 *MFIPs*.

输入:h 的非冗余 FP 子树 T_h,h 的投影 $MFIP_h$

输出:最大频繁项集树 MFI-tree

(1) if T_h 是单路径树

(2) 在 MFI-tree 中插入项集 $h\cup t$ 得到新的叶结点 *leaf*
(3) update (*leaf*,*MFIPs*) //调用过程 2 更新投影集合
(3) else 对 T_h 头表中的每个项 x
(4) $h'=h\cup\{x\}$
(5) 为 h'的非冗余 FP 子树创建头表 t'
(6) $MFIP_h'$=MfiProjection(x,$MFIP_h$) //调用过程 1 计算下层投影
(7) if $|t'|=|\max p|$ //max$p$ 为 $MFIP_h$'中的最大元素
(8) $MFIPs=MFIPs\cup MFIP_h$
(9) 创建非冗余 FP 子树 T_h'
(10) FPMFI(T_h',$MFIP_h'$)
(11) $h=h-\{x\}$

3 算法分析与比较

假设项集 $h\cup t$ 排序后为 $\{i_1,i_2,\ldots,i_j,\ldots,i_k\}$(其中 $j=|t|$,$1\le j\le k$),MFI-tree 中 i_k 的结点链个数为 n,在 FpMax 算法中,超集检测的代价约为 $\sum_{i=1}^{m} l_i$ (l_i 为 MFI-tree 中 i_k 的结点链中第 i 个结点到根结点的路径长度).对 h 中元素 i_{j+1},令 $m=|MFIP_{h-\{i_{j+1}\}}|$,则 $m<n=|MFIP_{i_k}|$.不妨假设 MFI-tree 中 i_k 的 n 个结点链中前 m 个结点的路径集合包含 $MFIP_{h-\{i_{j+1}\}}$ 中每个项集,计算 h 投影的代价最大为 $\sum_{i=1}^{m}(l_i-k+j)$.

经过分析我们知道,基于投影的每次超集检测访问 MFI-Tree 的代价都小于算法 FpMax 中的相应超集检测的代价,在最大频繁项集数量庞大的情况下,超集检测是算法中相当频繁的一步操作,对它的优化能大大提高算法的效率.另外,由第 2.2 节知,使用项集 h 非冗余 FP 子树可以压缩 FP 子树的规模,对于项集 h 的各个超集 ($h\cup x$,$x\in t$)的 FP 子树构造来说,h 的非冗余 FP 子树也能产生更少的条件模式基,从而进一步提高了算法的性能.分析表明,FPMFI 算法优于 FpMax 算法.

为进一步验证算法 FPMFI 的优越性,我们用 VC++6.0 在内存为 DDR 1G、CPU 为 Pentium IV 2.4G 的装有 Windows 2000 Professional 操作系统的机器上实现了 3 个基于 FP-Tree 框架的算法 FpMax,FPMFI 和 DMFIA.实验采用的数据是文献[3]中的 Mushroom 数据库,该数据库共有 8 124 个事务,事务平均长度为 23,包含了蘑菇的 115 种不同属性(项).在实验结果中,算法总运行时间包括读取事务数据库的 IO 时间,不包括输出最大频繁项集集合的 IO 时间;算法 DMFIA 的超集检测时间是指文献[5]中算法 1 第 13 行执行时间的累计;算法 FpMax 的超集检测时间是文献[10]中第 4 页的过程 FpMAX 第 7 行执行时间的累计;算法 FPMFI 的超集检测时间是本文中过程 3 中第 2,6,7 行执行时间的累计.

图 2 显示了算法 FpMax 和 DMFIA 在不同的最小支持度情况(分 5 档: 5%,1%,0.5%,0.2%,0.1%)下超集检测时间与总运行时间的百分比情况.图 2 说明,在最小支持度较小、挖掘出来的最大频繁项集数目庞大的时候,超集检测成为算法中最为耗时的一步操作,说明对超集检测的优化十分重要.

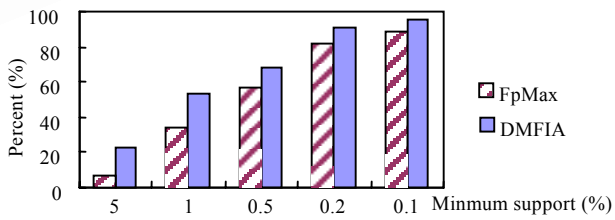

Fig.2 The ratios of superset checking time to execution time in algorithm FpMax and DMFIA
图 2 算法 FpMax 和 DMFIA 中超集检测时间与总运行时间的百分比

图 3 显示了算法 FpMax,FPMFI 和 DMFIA 在不同最小支持度情况(分 5 档: 5%,1%,0.5%,0.2%,0.1%)下的超集检测时间.由于最小支持度越小,超集检测操作越频繁,所以随着最小支持度的减少,算法 FPMFI 中的超集检测时间相对于算法 FpMax 和 DMFIA 中的超集检测时间也减少得更多.如图 3 所示,在最小支持度为 0.01 和 0.001 时,算法 FPMFI,FpMax 和 DMFIA 的超集检测时间分别为 1.3s,1.7s,4.4s 和 21.7s,68.6s,124.6s,算法 FPMFI 相对于算法 FpMax 时间减少量由 0.4s 变化到 46.9s,而相对于算法 DMFIA 时间减少量由 3.1s 变化到 102.9s.图 3 进一步说明了基于最大频繁项集投影的超集检测机制的高效性.

图 4 显示了算法 FpMax,FPMFI 和 DMFIA 在不同最小支持度情况(分 5 档:5%,1%,0.5%,0.2%,0.1%)下的总运行时间.由于算法 FPMFI 采用了基于最大频繁项集投影的超集检测机制,并将 FP 子树压缩为非冗余 FP 子树,其性能优于算法 FpMax 和 DMFIA.如图 3 中显示,在最小支持度为 0.001 时,算法 FPMFI,FpMax 和 DMFIA 的执行时间分别为 29.7s,77.0s 和 130.3s,算法 FPMFI 比同类算法性能提高了 1 倍以上.

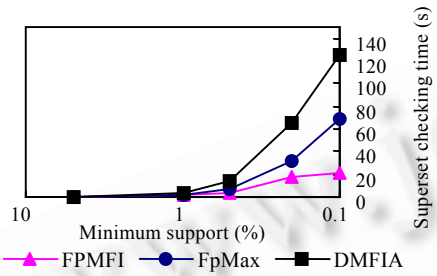

Fig.3 Superset checking time of algorithm FPMFI, FpMax and DMFIA

图 3 算法 FPMFI,FpMax 和 DMFIA 的超集检测时间

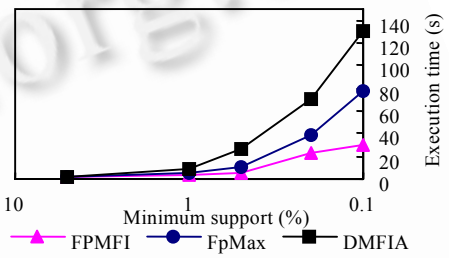

Fig.4 Execution time of algorithm FPMFI, FpMax and DMFIA

图 4 算法 FPMFI,FpMax 和 DMFIA 的执行时间

4 结束语

本文提出了基于最大频繁项集投影来做超集检测的思想,并删除了 FP 子树中相对于最大频繁项集冗余的信息,从而达到了提高基于 FP-Tree 挖掘最大频繁项集算法效率的目的.算法分析和实验比较表明,在最小支持度较小、最大频繁项集数目庞大的情况下,本文的 FPMFI 算法具有十分明显的优越性.

基于投影的最大频繁项集挖掘并不仅仅适合于基于模式增长的最大频繁项集挖掘算法,如何将它使用在其他类型的最大频繁项集挖掘中,是我们正在考虑的一个问题.另外,FP-Tree 结构较适合于频繁项集的挖掘,而对于最大频繁项集的挖掘而言,是否存在更合理的数据结构用于保存事务数据库,以便更有效地挖掘最大频繁项集,也是一个值得研究的课题.

References:

[1] Agrawal R, Srikant R. Fast algorithms for mining association rules. In: Proc. of the 20th Int'l Conf. on VLDB. 1994. 487−499. http://www.almaden.ibm.com/cs/people/srikant/papers/vldb94.pdf

[2] Yan YJ, Li ZJ, Chen HW. Frequent item sets mining algorithms. Computer Science, 2004,31(3):112−114 (in Chinese with English Abstract).

[3] Bayardo R. Efficiently mining long patterns from databases. In: Haas LM, ed. Proc. of the ACM SIGMOD Int'l Conf. on Management of Data. New York: ACM Press, 1998. 85−93.

[4] Lu SF, Lu ZD. Fast mining maximum frequent itemsets. Journal of Software, 2001,12(2):293−297 (in Chinese with English abstract).

[5] Song YQ, Zhu YQ, Sun ZH, Chen G. An algorithm and its updating algorithm based on FP-Tree for mining maximum frequent itemsets. Journal of Software, 2003,14(9):1586−1592 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/14/1586.htm

[6] Burdick D, Calimlim M, Gehrke J. Mafia: A maximal frequent itemset algorithm for transactional databases. In: Proc. of the 17th Int'l Conf. on Data Engineering. 2001. 443−452. http://www.cs.cornell.edu/boom/2001sp/yiu/mafia-camera.pdf

[7] Gouda K, Zaki MJ. Efficiently mining maximal frequent itemsets. In: Proc. of the 1st IEEE Int'l Conf. on Data Mining. 2001. 163−170. http://www.cs.tau.ac.il/~fiat/dmsem03/Efficient%20Mining%20Maxmal%20Frequent%20Itemsets%20-%202001.pdf

[8] Wang H, Li QH. An improved maximal frequent itemset algorithm. In: Wang GY, eds. Proc. of the Rough Sets, Fuzzy Sets, Data Mining, and Granular Computing, the 9th Int'l Conf. (RSFDGrC 2003). LNCS 2639, Heidelberg: Springer-Verlag, 2003. 484−490.

[9] Zhou QH, Wesley C, Lu BJ. SmartMiner: A depth 1st algorithm guided by tail information for mining maximal frequent itemsets. In: Proc. of the IEEE Int'l Conf. on Data Mining (ICDM2002). 2002. 570−577. http://www.serviceware.com/pdffiles/datasheets/ServiceWare-Smartminer-Datasheet.pdf

[10] Grahne G, Zhu JF. High performance mining of maximal frequent itemsets. In: Proc. of the 6th SIAM Int'l Workshop on High Performance Data Mining (HPDM 2003). 2003. 135−143. http://www.cs.concordia.ca/db/dbdm/hpdm03.pdf

[11] Agarwal RC, Aggarwal CC, Prasad VVV. Depth 1st generation of long patterns. In: Proc. of the 6th ACM SIGKDD Int'l Conf. on Knowledge Discovery and Data Mining. 2000. 108−118. http://www.cs.tau.ac.il/~fiat/dmsem03/Depth%20First%20Generation%20of%20Long%20Patterns%20-%202000.pdf

[12] Wang H, Xiao ZJ, Zhang HJ, Jiang SY. Parallel algorithm for mining maximal frequent patterns. In: Zhou XM, ed. Advanced Parallel Processing Technologies (APPT 2003). LNCS 2834, Heidelberg: Springer-Verlag, 2003. 241−248.

[13] Han J, Pei J, Yin Y. Mining frequent patterns without candidate generation. In: Proc. of the Special Interest Group on Management of Data. 2000. 1−12. http://www.utsc.utoronto.ca/~mendel/d43/04f/slides/datamining.pdf

附中文参考文献:

[2] 颜跃进,李舟军,陈火旺.频繁项集挖掘算法.计算机科学,2004,31(3):112−114.

[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] 宋余庆,朱玉全,孙志辉,陈耿.基于 FP-Tree 的最大频繁项集挖掘及其更新算法.软件学报,2003,14(9):1586−1592.http://www.jos.org.cn/1000-9825/14/1586.htm