基于FPMAX的最大频繁项目集挖掘改进算法

牛新征 佘 堃
(电子科技大学计算机科学与工程学院 成都 611731)

摘 要 挖掘事务数据库中的最大频繁项目集是数据挖掘领域一个重要的研究方向。基于FP-tree的FPMAX算法是目前较为高效与稳定的最大频繁项目集挖掘算法之一。然而对于稠密数据库中的挖掘,FPMAX会产生大量的冗余递归过程,导致额外的条件FP-tree构造开销。而且在支持度较低时,FPMAX则会因用于超集检测的全局MFI-tree较为庞大而导致超集检测的性能下降。为此提出FPMAX的改进算法FPMAX-reduce,其通过采用基于事务共同后缀的前瞻剪枝策略来减少挖掘过程中的冗余递归过程。当递归过程中产生的新条件FP-tree规模较小时,FPMAX-reduce通过构造条件MFI-tree来减小后续超集检测遍历的开销。性能试验表明,FPMAX-reduce算法通过有效的前瞻剪枝,在稠密事务数据库以及低支持度的情况下至多可将递归过程减少至原算法的一半以下,进而有效地提高了FPMAX算法的效率。

关键词 频繁项目集,最大频繁项目集,FP-tree,FPMAX,FP-growth
中图法分类号 TP391 **文献标识码** A

Mining Maximal Frequent Item Sets with Improved Algorithm of FPMAX

NIU Xin-zheng SHE Kun
(School of Computer Science and Engineering, University of Electronic Science and Technology of China, Chengdu 611731, China)

Abstract Finding maximal frequent itemsets is an important issue in data mining research field. The FPMAX algorithm, which is based on the FP-tree structure, has been proved to be one of the high-performance algorithms on maximal frequent itemsets mining. But for data mining task in dense datasets, FPMAX algorithm will construct a large number of redundant conditional FP-tree. What's more, if the quantity of frequent itemsets is large, the MFI-tree structure used for subset testing in FPMAX will become quite big, decreasing the efficiency of subset testing in the algorithm. Therefore, this paper proposed the FPMAX-reduce algorithm to overcome those drawbacks of FPMAX. This novel algorithm uses a pruning technique based on the common suffix of transactions and greatly reduces the construction of redundant conditional FP-tree. Besides, when the scale of the newly constructed conditional FP-tree is small, FPMAX-reduce constructs a corresponding conditional MFI-tree, which deletes the redundant information, to improves the efficiency of subset testing in the following recursive calls. Experimental results show that FPMAX-reduce algorithm effectively improves the efficiency of FPMAX and outperforms many existing available algorithms in dense datasets.

Keywords Frequent itemset, Maximal frequent itemset, FP-tree, FPMAX, FP-growth

1 引言

频繁项目集挖掘是数据挖掘领域中的一个基本问题,并且在关联规则挖掘、查询扩展、归纳数据库等多种应用场合中发挥着重要的作用。然而对一个维数为L的频繁项目集,若其每个子集都是频繁项目集,则一共有2^L-1个子集,当L很大时,将形成一个NP-hard问题。由于最大频繁项目集已经隐含了所有频繁项目集,故可把发现频繁项目集的问题转化为挖掘最大频繁项目集的问题。此外,某些数据挖掘应用仅需挖掘最大频繁项目集,而不必挖掘所有的频繁项目集,因而挖掘最大频繁项目集对数据挖掘具有重要意义。

目前用于挖掘最大频繁项集的算法主要有Max-Miner[1],MAFIA[2],genMax[3],IDMFIA[4],LCMmax[5,6],FPMAX[7]等。其中Max-Miner,MAFIA,genMax,IDMFIA主要通过对集合枚举树进行顺序的调整和前瞻剪枝来缩小搜索空间,并分别用位图、FP-tree等存储结构进行优化。LCMmax对集合枚举树做了精简,只需枚举闭项集,减少了枚举次数,并通过位图、前缀树和带权数列3种存储结构的综合应用,减少了广度优先回溯枚举类算法在支持度计算上的开销。不过由于上述算法多数需要通过枚举候选频繁项集来得到结果,期间产生的许多不必要的候选项制约了算法的性能。FPMAX是FP-tree的深度优先算法,通过递归的构造条件

到稿日期:2013-03-24 返修日期:2013-06-12 本文受国家自然科学基金(61300192),四川省科技厅科技支撑计划项目(2012GZ0061),中央高校基本科研业务费电子科技大学项目(ZYGX2010J075)资助。
牛新征(1978—),男,博士,副教授,主要研究方向为数据挖掘、网络计算,E-mail:xinzhengniu@uestc.edu.cn;佘 堃(1968—),男,博士,教授,博士生导师,主要研究方向为网络计算、人工智能。

FP-tree 直接在树上得到频繁项，避免了候选项集的产生，并使用 MFI-tree 存储已挖掘到的最大频繁项集以及实现超集检测。实验结果表明，FPMAX 在多数情况下的表现均优于 MAFIA、genMax 等其它算法。后来 GostaGrahne 等人提出的 FPMAX* [8] 改进算法通过引进预计数技术以及构造条件 MFI-tree 策略更进一步地提升了 FPMAX 在稀疏数据库中的效率，在 FIMI'03[9] 的实验中，基于 FPMAX 的 FPMAX* 被证明是较为高效的算法之一。但 FPMAX* 并未针对 FPMAX 算法中存在的包含冗余递归过程的问题进行改进与优化。此外，FPMAX* 中提出的预计数技术存在着在较长事务下性能下降的问题，条件 MFI-tree 则可能在稠密数据库中引入大量不必要的构造条件 MFI-tree 的开销。故在稠密数据库方面 FPMAX* 相对于 FPMAX 的效率提升并不明显。因此本文提出 FPMAX-reduce 算法，其通过对冗余的递归过程采取有效地规避策略以及对条件 MFI-tree 的选择性构造策略，有效地提升了 FPMAX 算法在稠密数据库中的效率。最后，通过对 FPMAX-reduce 与 FPMAX 算法在稠密数据库中的递归生成 FP-tree 子树次数以及算法执行时间验证了 FPMAX-reduce 算法在稠密数据库方面的优越性。

2 相关知识

2.1 频繁项集和最大频繁项集

设 $I=\{i_1,i_2,\cdots,i_n\}$ 是数据库中所有不同项的集合，数据库 D 是事务的集合，其中的每个事务 T 是集合 I 的子集，即有 $T\subseteq I$。对于给定的事务数据库 D，其包含的总事务数为 N，定义项集 X ($X\subseteq I$)的支持度计数(为简单起见，后面统一称为支持数)$count(X)$为 D 中包含事务 T 的个数使得 $X\subseteq T$，集合 X 的支持度 $support(X)$为 $count(X)/N$。

定义 1 对于某个指定的最小支持度 $minSup$，若项集 X 满足 $support(X)\geqslant minSup$，则称项集 X 为频繁项集，反之则称 X 为非频繁项集。最小支持数 $minCount$ 满足 $minCount=minSup\times N$，且 $count(x)\geqslant minCount\equiv support(X)\geqslant minSup$。

定义 2 若对于频繁项集 X，不存在一个频繁项集 Y 使得 $X\subset Y$，则称频繁项集 X 为最大频繁项集。

性质 1 任意最大频繁项集的真子集都不是最大频繁项集。

性质 2 任意频繁项集的子集都是频繁项集。

2.2 FP-tree 与算法 FPMAX，FPMAX*

在 Han 等人提出的 FP-growth[10] 频繁项集挖掘方法中，FP-tree 结构被提出并应用于数据库中频繁项目信息的紧凑保存。

在 FP-tree 结构中，根节点到每一个子孙节点的路径对应数据库中的一个项集。每个节点对应一个数据项，并记录根节点到该节点的路径对应项集的支持度，根节点至该节点的路径上的各节点以节点在对应数据库中的支持度降序排序。FP-tree 的头表中记录了各频繁项的名称和支持数，并存有指向对应项链表的指针。FP-tree 通过合并项集的相同前缀达到数据压缩的目的。

图 1(a)给出了一个小型事务数据库的例子，图 1(b)则给出了该事务数据库在 $minSup=2$ 时对应的 FP-tree。关于 FP-tree 与 FP-growth 的更多细节详见文献[11]。

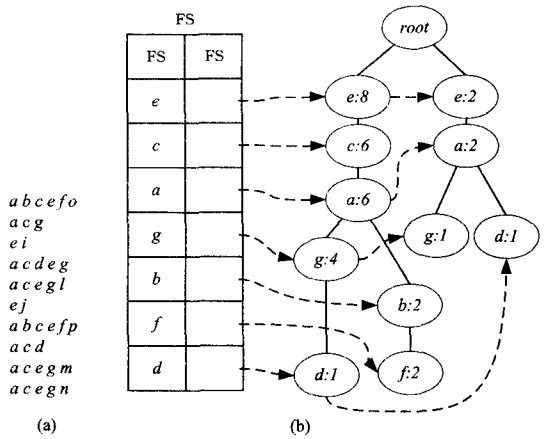

图 1 事务数据库与对应的 FP-tree 示例

定义 3 对于包含在 FP-tree 头表中的一个项 i，对 FP-tree 中根节点到达 i 的路径不包含 i 的部分称为 i 的前缀子路径，i 称为该路径的后缀。FP-tree 中所有 i 的不同前缀子路径组成 i 的条件模式基，由 i 的条件模式基所构建的 FP-tree 称为 i 的条件 FP-tree 或简称为 FP 子树，产生 i 的条件 FP-tree 的 FP-tree 称为父 FP-tree。

定义 4 对于 FP-tree 的头表项 i,j，定义关系 $i<j$ 表示在头表中 i 比 j 靠前，即 $count(i)>count(j)$。

原始 FP-tree 的建立需要两次扫描原始数据库，条件 FP-tree 的建立则需要两次扫描搜索空间中父结点的的条件模式基。

FPMAX 算法是基于 FP-tree 数据结构与 FP-growth 改进而来的针对最大频繁项集挖掘的算法。FPMAX 在搜索过程中将当前 FP-tree 头表中的某一项加入头项集 Head 中，并建立对应的条件 FP-tree 的头表。检验条件 FP-tree 的头表项集与头项集的并集是否为已挖掘出的最大频繁项集的子集，若为子集则进行剪枝，反之则建立对应的条件 FP-tree，并在条件 FP-tree 中继续递归挖掘，直到条件 FP-tree 为单一路径。此时头项集与条件 FP-tree 头表项集的并集为最大频繁项集。FPMAX 使用一个类似 FP-tree 的数据结构 MFI-tree 来存储已挖掘出的最大频繁项集以及进行超集检测。MFI-tree 也由树结构和频繁项头表两部分组成，在超集检测时通过相应头表项中的指针对各可能包含待测项集的路径自底向上进行匹配。

图 2 给出了一个 MFI-tree 的示例，其中已经记录了 $\{c,a,d\}$，$\{e,c,a,b,f\}$，$\{e,c,a,g\}$ 3 个最大频繁项。假设目前需检测的项集按 MFI-tree 头表中的项目顺序排序后为 $\{e,a\}$，则只需经过头表中 a 的指针即可访问到可能包含该项集的路径。关于 FPMAX 的更多细节可参见文献[7]。

文献[8]中所提出的 FPMAX* 算法是基于 FPMAX 算法的一个改进算法，主要提升了 FPMAX 在稀疏数据库中的性能表现。其所做的主要工作有两点：1)引入了基于数组的预计数技术。在构建一棵 FP-tree 插入事务项集 T 时，对 $\{j\mid j$

$\in T\}$，即对$\{i|i<j, i\in T\}$进行计数更新。进而在基于该 FP-tree 生成条件 FP-tree 时可省去一次遍历该 FP-tree 统计频繁项的过程；2）在每一次生成条件 FP-tree 时依据该条件 FP-tree 中的频繁项生成对应的条件 MFI-tree 以减少超集检测的开销。

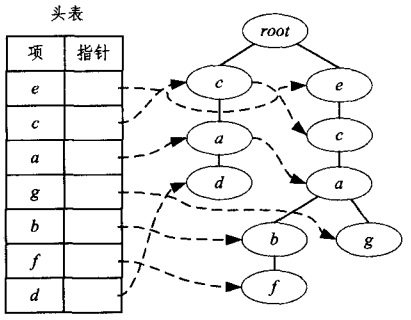

图 2　一个 MFI-tree 示例

3　基于 FPMAX 的改进算法 FPMAX-reduce

虽然目前已有 FPMAX* 对 FPMAX 做出了改进，但 FPMAX* 主要是针对于稀疏数据库环境下的优化，并且其并未减少 FPMAX 中潜在的冗余递归过程的次数。针对 FPMAX 算法在稠密数据库下的性能瓶颈，我们采取了下列优化策略，其中基于事务共同后缀的前瞻剪枝策略也可用于优化 FPMAX* 算法的性能。

3.1　基于事务共同后缀的前瞻剪枝策略

为了解决 FPMAX 算法中存在的产生冗余递归过程问题，我们首先从理论上分析冗余递归的发生情况以及相应的避免策略，于是引入下列定理。

定理 1　若对于插入 FP-tree 中的事务项集，存在项 $i<j$ 使得对于所有事务 T（如 $i\in T$）都有 $j\in T$，则此时将 i,j 加入头项集和只将 i 加入头项集所新建的子树相同。

证明：对于所有含有 i 的事务集合 S 和同时含有 i,j 的事务集合 S' 有 $S=S'$，因此 S 中事务 i 的前缀子路径与 S' 中事务 i 的前缀子路径一一对应，其两个事务集合中由 i 的条件模式基建成的条件 FP-tree 也应完全相同。

根据定理 1，若只将 i 加入头项集得到的频繁项集的集合为 S，则将 i,j 同时加入头项集得到频繁项集的集合为 $S'=\{T\cup j|T\in S\}$，其中 S 中的任意一个项集在 S' 中存在对应的超集，所以 S 中不存在最大频繁项集，无需单独将 i 加入头项集。因此我们在递归过程中可不把满足这种条件的 i 加入头项集，因为在将 j 加入头项集的递归过程中已搜索过将 i,j 同时加入头项集的情况。

如事务为$\{a\},\{a\},\{ace\},\{ace\},\{b\},\{b\},\{bcde\},\{bce\},\{d\},\{d\},\{d\}$，在 $minCount=2$ 时，将 c,e 同时加入头项集得到的频繁项集有$\{ace\},\{bce\}$，而只将 c 加入头项集得到的频繁项集$\{ac\},\{bc\}$都不会是最大频繁项集。

定理 2　设 i,j 是当前 FP-tree 头表中相邻的两个项，且有 $i<j$。若对于树中每个表示项 i 的节点，其有一孩子节点表示的项为 j 且两个节点支持数相同，则此时将 i,j 加入头项集和只将 i 加入头项集所新建的条件 FP-tree 相同。如图 3 中的项 c 和 d。

证明：要建立的条件 FP-tree 满足以上条件时，其插入的事务中若含有项 i 则必然含有项 j，根据性质 1 可知结论成立。

根据定理 2，可以在将头表中的项加入头项集中前先检测是否满足性质 2，如果满足则跳过将此项加入头项集的过程。

例如，在图 3(a)中 FP-tree 的挖掘过程中可跳过单独将 c 加入头项集的过程，因为同时将 c,d 加入头项集中与单独将 c 加入头项集中得到的子树相同，如图 3(b)所示。

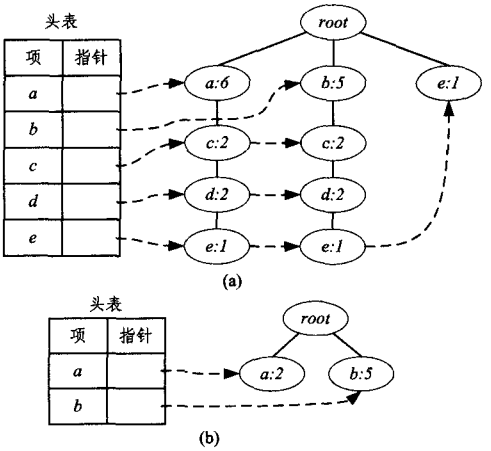

图 3　一种可能导致建立冗余条件 FP-tree 的情况

性质 2 是性质 1 的特殊情况，但是更容易检测，因此在 FPMAX-reduce 算法中当头表中项较多时使用策略 2 的部分剪枝，而在项少于经验参数 MN 时使用策略 1 的完全剪枝，并使用位图结构快速检测是否满足性质 1。

3.2　基于条件 MFI-tree 的超集检测优化策略

在 FPMAX 算法中，一个全局的 MFI-tree 被用来存储递归过程中发现的最大频繁项集以及对当前发现的最大频繁项集候选项集的超集进行检测，该策略是基于如下性质与定理。

性质 3　对于自底向上深度优先遍历 FP-tree 树空间过程中发现的单一路径 FP-tree 头表项集与头项集中的项目构成的项集不可能为一个在其后以相同方式产生的项集的子集。

定理 3　对于自底向上深度优先遍历 FP-tree 树空间过程中发现的单一路径 FP-tree 头表项集与头项集中的项目构成的项目集，若不为已发现的任一最大频繁项集的子集，则其为最大频繁项集。

证明：根据性质 1，若一个 FP-tree 为单一路径树，则该 FP-tree 的头表项集与头项集的项目构成的项目集可作为最大频繁项集的一个候选集。又根据性质 3，该项集不会为后续产生的最大频繁项集候选项集的子集，故定理可证。

当输入数据非常庞大时，挖掘过程中全局 MFI-tree 可能成长得十分庞大，一次超集检测可能要耗费数千次的比较。针对这个问题，在文献[7]中 GostaGrahne 等人提出在每次构造条件 FP-tree 时构造对应的条件 MFI-tree。条件 MFI-tree 中删去了不存在当前条件模式基中非频繁的项目。当递归至某一条件 FP-tree 时需进行的超集检测在其对应的条件 MFI-

tree 上进行，从而减少了比较的开销。然而该算法需要在每次产生条件 FP-tree 时都构造条件 MFI-tree，稠密数据中会由于大量构造条件 MFI-tree 产生较大开销。

针对上述不足，本文提出的基于条件 MFI-tree 的超集检测优化策略思路为：

(1)将每个 FP-tree 都与一个 MFI-tree 相关联。与条件 FP-tree 相关联的 MFI-tree 或为已存在的 MFI-tree，或为一个新生成的 MFI-tree。一个全局 MFItreeStack 栈保存各 MFI-tree，FP-tree 中通过一个 *mfitreeIndex* 记录与其关联的 MFI-tree 在栈中的位置。当在一个条件 FP-tree 上的递归结束时，若其关联的 MFI-tree 为该条件 FP-tree 所生成，则将该 MFI-tree 从栈中移除并销毁。由于 FPMAX 的深度优先遍历顺序，可以保证栈顶的 MFI-tree 即为当前递归所关联使用的 MFI-tree。

(2)设项目 α 为当前 FP-tree 的头表中的一项，对于 α 的条件模式基构建的条件 FP-tree，比较 α 的条件 FP-tree 的头表大小 *headSize* 与其父 FP-tree 使用的 MFI-tree 中 α 的最长前缀路径长度 *maxLength*。若 $headSize < maxLength \times THRESHOLD$ ($0 \leqslant THRESHOLD \leqslant 1$)，则从该 MFI-tree 构造新的条件 MFI-tree 与 α 的条件 FP-tree 相关联，否则将该 MFI-tree 与 α 的条件 MFI-tree 相关联。其中 *THRESHOLD* 为控制阈值，可依据具体数据库特征进行调整。

(3)构造 α 的条件 FP-tree 对应的条件 MFI-tree 时，需从当前 MFI-tree 中取出满足当前头项集模式的最大频繁项，并删去不在 α 的条件 FP-tree 的头表中的项。

(4)在递归至某一 FP-tree 对一个项集进行超集检测时，仅需提取存在于当前 MFI-tree 的头表中的项目在当前 MFI-tree 进行检测，但是若证明不为已有最大频繁项集的子集时，需要用该项集更新存在于 MFItreeStack 栈中的所有 MFI-tree。

3.3 最大频繁项集挖掘算法 FPMAX-reduce

通过前面的介绍，我们详细地分析了 FPMAX-reduce 算法对 FPMAX 算法所做的改进，本节给出 FPMAX_reduce 算法的伪代码。

FPMAX_reduce 是一个递归调用的算法，在算法开始前需要通过两次扫描原始事务数据库建立初始 FP-tree，并使用该 FP-tree 的头表项初始化一个 MFI-tree 放入 MFItreeStack 底部。

在 FPMAX_reduce 执行过程中，若发现本过程中的 FP-tree 不为单路径树，则依据当前 FP-tree 的头表大小以及在生成该 FP-tree 时设置的 isNeedTree 中的标志位来判断是否符合前述基于事务共同后缀的前缀剪枝的情况，若满足条件则可略去该递归过程。

若不满足条件，则在产生对应的条件模式基后在与其父 FP-tree 相关联的 MFI-tree 中进行超集检测过程(subsetChecking)。进行该过程时，只需取出待测项集中包含于该 MFI-tree 的头表中的项进行检测即可。若判断为非已挖掘最大频繁项集子集，则首先根据该条件模式基中的频繁项目数判断是否在构建条件 FP-tree 时依据前述剪枝策略计算 isNeedTree 标志位。然后依据频繁项目个数与当前使用的 MFI-tree 的头表项目个数关系决定其继续使用当前 MFI-tree 或产生新的条件 MFI-tree。

若需建立新的条件 MFI-tree，则从当前 MFI-tree 中取出包含产生当前条件模式基的后缀项的项集，仅保留其中包含于新构建的 FP-tree 的头表的项，将其插入至新的条件 MFI-tree 中。完成构建后将新的条件 MFI-tree 加入 MFItreeStack 的栈顶。

若一次 FPMAX_reduce 过程中的 FP-tree 为单路径树，则需使用该 FP-tree 的头表与全局头项集中的节点构成的新最大频繁项集更新(update)MFItreeStack 栈中当前保留的所有 MFI-tree。对每一 MFI-tree，选取新最大频繁项集中包含于其头表中的项目组成的项集进行插入。

在一棵 FP-tree 上的递归完成后，若当前使用的 MFI-tree 是由其所生成，则将该 MFI-tree 从 MFItreeStack 中移除，从而节省空间并可保证 MFItreeStack 顶端即为后续要使用的 MFI-tree。

在 FPMAX_reduce 算法结束时，MFItreeStack 底部的 MFI-tree 即记录了原始事务集中挖掘出的所有最大频繁项信息。

```
方法：FPMAX_reduce(T)
输入：T：FP-tree
全局变量：Head：记录当前 FP-tree 所基于的后缀项目
          MN，THRESHOLD：阈值变量
          MFItreeStack：记录生成的 MFI-tree 的栈
输出：更新后的 MFItreeStack
方法：
1. if T only contains a single path P
2.   for each MFItree in MFItreeStack
3.     update(MFItree，P∪Head)；
4.     return；
5. else for each item i inT. headerTable
6.   if T. headerTable. size<=MN and T. isNeedTree of i is false
7.     continue；
8.   else if T. headerTable. size> MN and each node of i has a child j
9.   continue；
10. Insert i to Head；
11. Construct the conditional pattern base of i；
12. Tail={frequent items in base}；
13. if subsetChecking(Head∪Tail，T. mfitreeIndex) is false
14.   if Tail. size<=MN
15.     Construct i's conditional FP-tree Ti and initialize Ti. isNeedTree；
16.   else
17.     Construct i's conditional FP-tree Ti without Ti. isNeedTree；
18.   if Tail. size<=THRESHOLD * MFItreeStack[T. mfitreeIndex]. headerTable. size
19.     construct conditional MFI-tree Mi from MFItreeStack[T. mfitreeIndex]；
20.     Insert Mi to MFItreeStack；
21.     Ti. mfitreeIndex=T. mfitreeIndex+1；
```

```
22.   else
23.     T_i. mfitreeIndex=T. mfitreeIndex;
24.   call FPMAX_reduce(T);
25. remove i from Head;
26. if MFItreeStack. TopItemIndex equals T. mfitreeIndex
27.   popMFItreeStack;
```

4 算法分析与比较

为了验证算法 FPMAX-reduce 相对原 FPMAX 算法的优越性。我们使用 C++语言在 Visual Studio 2010 开发环境，Windows7 64 位操作系统，配置 4GB DDR3 1333 内存与 Intel Core i5 M480 的计算机上分别依据本文前述的伪代码与文献[6]的相关介绍实现了 FPMAX-reduce 与 FPMAX 算法。实验采用的数据来自 MUSHROOM、PUMSB、PUMSB-star 等 3 个稠密公共事务数据库，数据库中的事务数目分别为 8124、49046、49046。这些公共事务数据库的测试支持数选择见表 1。其中 MUSHROOM 由 UC Irvine Machine Learning Repository 提供，PUMSB 包含来自 Public User Microdata Sample 得到的人口普查数据，PUMSB-star 则是由 PUMSB 删去所有支持度大于 80%的项后产生的相对低密度数据库。

表 1 事务数据库的测试样本选取

测试样本编号	MUSHROOM 支持数选取	PUMSB 支持数选取	PUMSB-star 支持数选取
sample1	406 (5%)	44141 (90%)	24523 (50%)
sample2	162 (2%)	41689 (85%)	19618 (40%)
sample3	81 (1%)	39236 (80%)	14714 (30%)
sample4	41 (0.5%)	36784 (75%)	9809 (20%)
sample5	16 (0.2%)	34332 (70%)	7357 (15%)

我们的测试中，保证在一次比较中两种算法接受相同的原始数据输入与支持数参数。考虑到如果生成包含大量最大频繁项集的结果文件会影响算法本身执行时间的精确性，并且 FPMAX-reduce 与 FPMAX 在原始事务数据库的读取与结果生成方面并无区别，因此在计算算法运行时间时不包括读取原始事务数据库与输出最大频繁项集的 I/O 时间。此外，在一次算法执行后，我们还记录其递归建立条件 FP-tree 的次数。为保证数据准确，算法在每个比较中运行 10 次，计算均值作为结果。

表 2 递归建立条件 FP-tree 的次数

数据库	支持数	FPMAX (次)	FPMAX-reduce (次)
MUSHROOM	406 (5%)	12616	8075
MUSHROOM	162 (2%)	42343	23005
MUSHROOM	81 (1%)	85428	39665
MUSHROOM	41 (0.5%)	164813	62864
MUSHROOM	16 (0.2%)	358880	111429
PUMSB	44141 (90%)	1136	936
PUMSB	41689 (85%)	6624	4697
PUMSB	39236 (80%)	27156	15857
PUMSB	36784 (75%)	77482	39341
PUMSB	34332 (70%)	151924	72194
PUMSB-star	24523 (50%)	142	125
PUMSB-star	19618 (40%)	1580	1344
PUMSB-star	14714 (30%)	6455	5775
PUMSB-star	9809 (20%)	34235	29790
PUMSB-star	7357 (15%)	82055	67896

表 2 记录了 FPMAX 和 FPMAX-reduce 在不同的最小支持度下在各事务数据库中进行挖掘时递归建立 FP-tree 的次数情况。该数值统计结果说明，FPMAX-reduce 中采取的基于事务共同后缀的前瞻剪枝策略可以有效地减少原 FPMAX 中的冗余条件 FP-tree 的构建，并且随着支持度的下降，FPMAX-reduce 对冗余递归过程的避免效果更为明显。实验数据表明，在某些数据库(如 MUSHROOM，PUMSB)中，当支持度较低时，FPMAX-reduce 仅需原 FPMAX 一半或更少的递归，即可完成最大频繁项集的挖掘。

图 4—图 6 分别给出了算法 FPMAX-reduce 与 FPMAX 在不同数据库的不同最小支持度下的总运行时间，由运行结果来看，较高支持度下，由于冗余递归数目以及最大频繁项集均较少，FPMAX-reduce 优化效果并不明显而与 FPMAX 算法效率相当。但在较低支持度下 FPMAX-reduce 算法较 FPMAX 算法效率有了较大提升，尤其是在大型稠密数据库 PUMSB 及 MUSHROOM 数据库中，改进后的算法执行时间可减少至原算法一半以下。虽然 FPMAX-reduce 算法增加了冗余递归情况的判断以及构造条件 MFI-tree 的开销，但从测试结果中 FPMAX-reduce 算法相对原 FPMAX 算法的提升效果来看，可认为 FPMAX-reduce 是一个高效的算法。

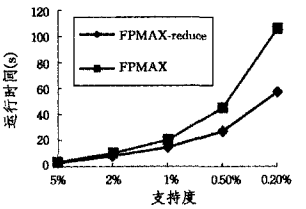

图 4 MUSHROOM 数据库执行时间比较

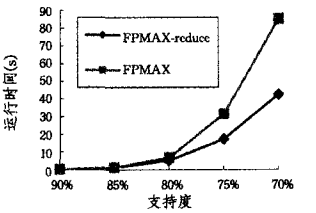

图 5 PUMSB 数据库执行时间比较

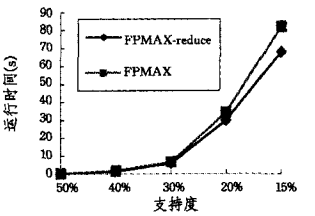

图 6 PUMSB-star 数据库执行时间比较

结束语 本文提出了最大频繁项集挖掘算法 FPMAX-reduce。该算法通过基于事务共同后缀的前瞻剪枝策略和选择性构造条件 MFI-tree 的超集检测优化策略，有效地改进了 FPMAX 算法在稠密数据库以及较低支持度下挖掘时存在的性能瓶颈问题。通过算法分析与实验比较验证，在稠密事务数据库以及较低支持度的情况下，本文的 FPMAX-reduce 算法可以有效减少 FPMAX 中冗余 FP-tree 的构建，在时间效率与空间效率方面均具有一定优越性，进而可被应用于其它数据挖掘应用如大规模数据下的聚类划分[11]等的改进中。

参 考 文 献

[1] BayardoJr R J. Efficiently mining long patterns from databases [C] // Proceedings of the 1998 ACM SIGMOD International Conference on Management of Data. New York, 1998: 85-93

[2] Burdick D, Calimlim M, Flannick J, et al. Mafia: A maximal fre-

quent itemsetalgorithm[J]. IEEE Transactions on Knowledge and Data Engineering,2005,17(11):1490-1504

[3] Gouda K, Zaki M J. Efficiently mining maximal frequent itemsets[C]// Proceedings of the 2001 IEEE International Conference on Data Mining. San Jose,2001:163-170

[4] 吉根林,杨明,宋余庆,等.最大频繁项目集的快速更新[J].计算机学报,2005,28(1):128-135

[5] Uno T,Kiyomi M,Arimura H. LCM ver. 2:Efficient mining algorithms for frequent/closed/maximal itemsets[C]// Proceedings of the 2004 IEEE ICDM Workshop on Frequent Itemset Mining Implementations. Brighton,2004:1-11

[6] Uno T,Kiyomi M,Arimura H. Lcm ver. 3:Collaboration of array,bitmap and prefix tree for frequent itemset mining[C]// Proceedings of the 1st International Workshop on Open Source Data Mining:Frequent Pattern Mining Implementations. New York,2005,77-86

[7] Grahne G,Zhu J. High performance mining of maximal frequent itemsets[C]// Proceedings of the 6th International Workshop on High Performance Data Mining. 2003:1-10

[8] Grahne G, Zhu J. Efficiently using prefix-trees in mining frequent itemsets[C]// Proceedings of the Third FIMI Workshop on Frequent Itemset Mining Implementations. Florida, 2003:123-132

[9] Goethals B,Zaki M J. Advances in frequent itemset mining implementations:report on FIMI'03[J]. ACM SIGKDD Explorations Newsletter,2004,6(1):109-117

[10] Han J,Pei J,Yin Y. Mining frequent patterns without candidate generation[J]. ACM SIGMOD Record,2000,29(2):1-12

[11] 牛新征,佘堃. 面向大规模数据的快速并行聚类划分算法研究[J].计算机科学,2012,39(1):134-137

(上接第 207 页)

Setup 算法,通过获得中央节点分发的全局参数 GP,生成各节点的公钥组和私钥组,并根据输入的 GID 值,为属于本授权机构的每个用户属性、身份对输出密钥 $K_{i,GID}$。

(3)终端客户节点。客户节点拥有分属于不同分机构节点的属性,访问数据时通过登录提交相应的属性,并提交密钥 $K_{i,GID}$,若属性符合访问矩阵则可获得所需的数据明文。

结束语 本文将属性加密机制应用到气象云数据的管理研究中,给出了一种基于多方授权的属性加密的访问控制模型。通过引入多方授权方案,很好地解决了实际使用中需要有多个授权机构情况下的文件共享访问问题,同时通过引入系统用户全局唯一的身份标识 GID 来有效避免不同用户间的合谋攻击问题。因此,本系统方案具有较高的安全性和很好的实用价值。

参 考 文 献

[1] Hur J,Noh D K. Attribute-Based Access Control with Efficient Revocation in Data Outsourcing Systems[J]. IEEE Transactions on Parallel and Distributed Systems,2011,22(7)

[2] Yang Ming. An Efficient Attribute based Encryption Scheme with Revocation for Outsourced Data Sharing Control[C]//2011 International Conference on Instrumentation, Measurement, Computer,Communication and Control. 2011

[3] Jia Hong-yong. Efficient and Scalable Multicast Key Management Using Attribute Based Encryption[C]// 2010 International Conference on Information Theory and Information Security. 2010

[4] 苏金树,曹丹,王小峰,等.属性基加密机制[J].软件学报,2011,22(6)

[5] Subashini S,Kavitha V. A survey on security issues in service delivery models of cloud computing[J]. Journal of Network and Computer Applications,2011,34

[6] 张磊,曹珍富.一个适合分布式网络的属性基加密方案[J].上海交通大学学报,2010,44(11)

[7] 陈勤,党正芹,张金漫,等.一种多认证机构可验证的属性基加密方案[J].计算机应用研究,2012,29(1)

[8] Liu Yu-chao. A Method for Trust Management in Cloud Computing:Data Coloring by Cloud Watermarking[J]. International Journal of Automation and Computing,2011,8(3)

[9] Sakr S,Liu A. A Survey of Large Scale Data Management Approaches in Cloud Environments[J]. IEEE Communications Surveys & Tutorials,2011,13(3)

[10] Yu Shu-cheng, Wang Cong. Achieving Secure, Scalable, and Fine-grained Data Access Control in Cloud Computing[C]// IEEE INFOCOM. 2010

[11] Zissis D,Lekkas D. Addressing cloud computing security issues[J]. Future Generation Computer Systems,2012,28(3)

[12] Hay B, Nance K, Bishop M. Storm Clouds Rising: Security Challenges for IaaS Cloud Computing[C]//Proceedings of the 44th Hawaii International Conference on System Sciences. 2011

[13] Rodero-Merino L. Building safe PaaS clouds:A survey on security in multitenant software platforms[J]. Computers & Security, 2012,31

[14] Blandford R. Information security in the cloud[M]. Network Security,April 2011

[15] Walters R. Managing privileged user activity in the datacentre [M]. Network Security,November 2010

[16] Zhuo Hao. A Privacy-Preserving Remote Data Integrity Checking Protocol with Data Dynamics and Public Verifiability[J]. IEEE Transactions on knowledge and data engineering,2011,23(9)

[17] Jia Wei-wei. SDSM:A Secure Data Service Mechanism in Mobile Cloud Computing[C]// The First International Workshop on Security in Computers,Networking and Communications. 2011

[18] Pearson S, Mont M C. Sticky Policies: An Approach for Managing Privacy across Multiple Parties[J]. Computer, 2011, 44(9):60-68

[19] Garber L. Serious Security Flaws Identified in Cloud Systems [J]. Computer,2011,44(12):21-23

[20] Song D,Shi E,Fischer I. Cloud Data Protection for the Masses [J]. Computer,2012,45(1):39-45