

基于改进 FP-tree 的最大频繁项目集挖掘算法

马丽生*, 姚光顺, 杨传健

(滁州学院 计算机与信息工程学院, 安徽 滁州 239000)

(*通信作者电子邮箱 mls20041084@sina.com)

摘要: 针对已有算法为了减少 FP-tree 中路径被重复遍历的次数, 需要保存 FP-tree 中所有频繁 1-项集的条件模式基的问题, 对 FP-tree 的数据结构进行修改, 使得只需要保存 FP-tree 中每个叶子节点的父节点到根节点路径上项目组成的条件模式基, 降低了保存条件模式基的存储空间开销。在分析最大频繁项目集挖掘算法中搜索空间以及数据表示方法的基础上, 通过理论分析和证明, 设计了剪枝策略和压缩策略, 缩小了算法搜索空间, 压缩了 FP-tree 的规模, 提高了算法的执行效率。最后将新算法分别与 NHTFPG 算法、FpMAX 算法进行对比, 验证算法的正确性和有效性。实验结果表明, 新算法保存 FP-tree 条件模式基所需要的存储空间不到 NHTFPG 算法的 50%, 执行效率比 FpMAX 算法提高了 2~3 倍。

关键词: 频繁项目集; 最大频繁项目集; 条件模式基; 项头表; 剪枝策略; 压缩策略

中图分类号: TP311.13 **文献标志码:** A

Mining algorithm for maximal frequent itemsets based on improved FP-tree

MA Li-sheng*, YAO Guang-shun, YANG Chuan-jian

(College of Computer and Information Engineering, Chuzhou University, Chuzhou Anhui 239000, China)

Abstract: In order to reduce the repeated traversal times of path in the FP-tree, the conditional pattern bases of all frequent 1-itemsets in the FP-tree need to be saved in the existing algorithms. Concerning this problem, in the new algorithm, the data structure of FP-tree was improved that only the conditional pattern bases were saved which were constituted by the items in the path from every leaf node's parents to the root in the FP-tree, and the storage space of the conditional pattern bases was reduced. After studying search space and the method of data representation in the algorithm for mining maximal frequent itemsets, the pruning and compression strategies were developed through theoretical analysis and verification, which could decrease the search space and the scale of FP-tree. Finally, the new algorithm was compared with NHTFPG algorithm and FpMAX algorithm respectively in terms of accuracy and efficiency. The experimental results show that the new FP-tree algorithm saves the required conditions for model-based storage space more than 50% than NHTFPG algorithm, and the efficiency ratio improves by 2 to 3 times than FpMAX algorithm.

Key words: frequent itemset; maximal frequent itemset; conditional pattern base; item header table; pruning strategy; compression strategy

0 引言

从 IBM 的 Agrawal 博士等 1993 年提出 Apriori^[1-2] 方法以来, 许多研究人员对频繁项目集挖掘问题进行了大量的研究^[1-3]。但是由于挖掘频繁项目集要考虑太多的候选频繁项目集, 所以提出最大频繁项目集挖掘算法, 最大频繁项目集隐含了所有的频繁项目集且数量明显少于频繁项目集, 而且许多应用也只需要挖掘最大频繁项目集, 因此挖掘最大频繁项目集具有重要的意义。

目前已有的最大频繁项目集挖掘算法可分为宽度优先搜索算法^[4-5]和深度优先搜索算法^[6-12]两种。宽度优先搜索以 Apriori 算法为基础, 其中 Pincer-Search^[4] 和 MaxMiner^[5] 是两种典型的算法。深度优先算法以 DepthProject^[6]、MAFIA^[7]、FpMax^[8] 为代表, 其中 FpMax 算法是基于 FP-tree 挖掘最大频繁项目集的深度优先算法。但是基于 FP-tree 的算法在建立 FP-tree 时需要多次遍历共享前缀路径, 影响了算法的执行效率。文献[13]为解决上述问题提出了解决方法,

文献中建立无项头表的 FP-tree, 通过对 FP-tree 遍历一次求出 FP-tree 中每个频繁 1-项集的所有条件模式基, 从而避免了对共享前缀路径的重复遍历。但是由于一次遍历后需要保存频繁 1-项集的所有条件模式基, 当 FP-tree 较大时保存所有条件模式基时需要大量的存储空间。通过分析可以发现, 在最大频繁项目集搜索过程中以及 FP-tree 建立时可以进行优化, 进一步提高算法的效率, 在此基础上提出了一种最大频繁项目集挖掘算法, 称为 IMMFA。

1 频繁项目集和最大频繁项目集

设 $I = \{i_1, i_2, \dots, i_m\}$ 是不同项目的集合。设任务相关的数据 D 是数据库事务的集合, 其中每个事务 T 是项目的集合, 使得 $T \subseteq I$ 。每个事务有一个标识符, 称作 TID。对于项目集 $X \subseteq I$, 如果 $X \subseteq T$, 则称 X 被事务 T 包含。项目集 X 的支持数 $supp(X)$ 是 D 中包含 X 的事务 T 的个数。项目集 X 的支持度是 D 中包含 X 的事务 T 的个数与数据库中事务总数的百分比。项目集中包含的项目的个数称为项目集的长度, 如果项目集的

收稿日期: 2011-08-10; 修回日期: 2011-10-12。 基金项目: 安徽省高校省级自然科学基金项目 (KJ2010B421, KJ2011Z276); 安徽省高校省级优秀青年人才基金项目 (2010SQRL137, 2011SQRL123)。

作者简介: 马丽生 (1982-), 男, 山西柳林人, 讲师, 硕士研究生, 主要研究方向为: 数据挖掘; 姚光顺 (1982-), 男, 安徽肥东人, 讲师, 硕士研究生, 主要研究方向为: 人工智能; 杨传健 (1979-), 女, 安徽滁州人, 副教授, 硕士研究生, 主要研究方向为: 粗糙集、数据挖掘。

长度为 k , 则称为 k -项集。用户可以确定一个不大于数据 D 中的事务数的最小支持数阈值, 记为 \min_sup 。

定义 1 给定事务数据库 D 和最小支持数阈值 \min_sup , 对于项目集 $X \subseteq I$, 若 $supp(X) \geq \min_sup$, 则称 X 为 D 中的频繁项目集。

定义 2 给定事务数据库 D 和最小支持数阈值 \min_sup , 对于项目集 $X \subseteq I$, 若 $supp(X) \geq \min_sup$, 且对于项目集 X 任意超集都不是频繁项目集, 则称 X 为 D 中的最大频繁项目集。

定义 3 如果树中某节点的度大于 1, 则称该节点为分叉节点。

定义 4 如果树中某节点的度等于 0, 则称该节点为叶子节点。

性质 1 频繁项目集的任何真子集都不是最大频繁项目集。

2 最大频繁项目集挖掘算法

2.1 改进的 FP-tree (Frequent Pattern tree)

FP-tree 是将事务数据库中的每个事务中的频繁项目按照项目支持数从大到小排列后插入形成的一棵树, 树中的每个节点由六个域组成, 即项名 (item-name)、支持数计数 (count)、链指针 (Link)、孩子指针 (Child-Link)、兄弟指针 (Brother-Link) 以及父指针 (Parent-Link)。FP-tree 中还包括一个项头表 (Header-table), 每个单元由两部分组成, 即项名和指向树中与项名域中有相同项名的第一个节点的链指针。FP-tree 项头表中的每个频繁 1-项集在构建其对应的 FP-tree 时, 需要先求出该项目所对应的所有条件模式基, 即 FP-tree 中包含该项目的每个节点的父节点到根节点的路径上出现的项目的集合组成该频繁 1-项集的一条条件模式基, 文献[13]中通过遍历 FP-tree 一次求出项头表中每个项目的所有条件模式基并保存起来, 通过分析发现对于 FP-tree 而言只要保存每个叶子节点的父节点到根节点路径上项目组成的条件模式基即可, 树中其他节点中项目对应的条件模式基已经包含在该节点所对应的某条分支的叶子节点的父节点到根节点路径上项目组成的条件模式基中, 而不需要保存每个频繁 1-项集对应的所有条件模式基, 从而实现条件模式基的共享, 减少了保存条件模式基所需要的空间。为了实现条件模式基共享需要对 FP-tree 的结构进行修改, 在树中每个节点增加一个指针域称为条件模式基指针 (Condition-Link), 增加一个表示该节点在 FP-tree 中所处的层次的域, 称为层次域 (Level)。条件模式基指针或为空, 或指向一个包含该节点所共享的某个叶子节点的父节点到根节点路径上项目组成的条件模式基, 初始化时值为 NULL, FP-tree 中根节点层次为 0, 从上而下依次递增。

表 1 中的事务在支持数为 3 时得到的 FP-tree 如图 1 所示。FP-tree 中每个节点上的箭头代表条件模式基指针, 每个节点中的 3 个值分别表示项目, 该节点到根节点路径上项目组成的项目集的支持数以及该节点在 FP-tree 中所在的层次。例如 $(f, 3, 2)$ 表示该节点中的项目为 f , 3 表示项目集 $\{c, f\}$ 的支持数, 2 表示该节点所处的层次。

表 1 事务数据集

TID	Itemsets
01	$\{a, c, d, f, g, i, m, p\}$
02	$\{a, b, c, f, l, m, o\}$
03	$\{b, c, h, m, o\}$
04	$\{b, f, k, p, s\}$
05	$\{a, c, e, f, l, m, n, p\}$

2.2 事务数据的表示

对于频繁项目集 $Head$, 设 $Head = \{i_1, i_2, \dots, i_n\}$, 其中 $0 \leq n \leq |I|$, $n = 0$ 表示 $Head$ 为空集, $i_j (1 \leq j \leq n)$ 表示频繁项目, 且 $supp(i_j) \leq supp(i_k) (1 \leq j < k \leq n)$ 。

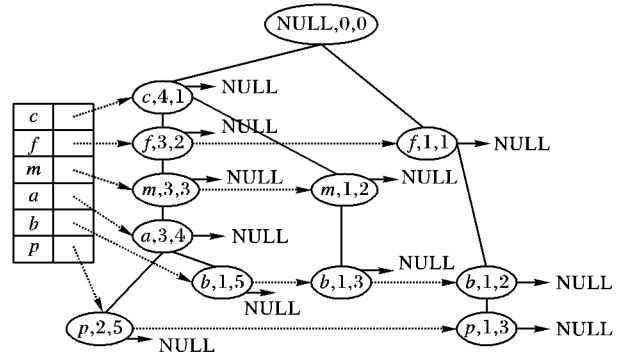


图 1 改进的 FP-tree

在已有基于 FP-tree 挖掘最大频繁项目集算法中, 在挖掘最大频繁项目集的搜索过程中, 对于挖掘得到的频繁项目集 $Head$ 对应一棵 FP-tree, 该 FP-tree 由频繁项目集 $Head$ 对应的条件模式基构成, 搜索构建频繁项目集 $Head$ 对应的 FP-tree 的条件模式基的过程为:

在频繁项目集 $Head = \{i_1, i_2, \dots, i_{n-1}\}$ 对应的 FP-tree 的项头表中找到项目 i_n 所在单元指针域所链接的单链表中的每个节点, 分两种情况:

1) 若节点 P 所对应的条件模式基指针为空, 则执行以下操作:

① 开辟一个大小等于 $P \rightarrow Level - 1$ 的数组 $Aarray_p$ 用于保存节点 P 的父节点到根节点路径上项目组成的节点 P 中项目的一条条件模式基, 并设置 $P \rightarrow Condition-Link = Aarray_p$ 。

② 从节点 P 的父节点开始向根节点遍历, 每个经过一个节点 Q (不包括根节点) 执行:

若 $Q \rightarrow Condition-Link = NULL$, 则将 Q 中的项目保存到数组 $Aarray_p$ 中, 并将数组下标值递增 1, 同时设置 $Q \rightarrow Condition-Link = Aarray_p$ 。

若 $Q \rightarrow Condition-Link \neq NULL$, 则该节点为分叉节点, 节点已经被遍历过了, 停止遍历。 $Q \rightarrow Condition-Link$ 指向其子孙节点中某个叶子节点的 Condition-Link 域所指示的保存条件模式基的数组 $Aarray_i$, 设数组的中元素个数为 n , 节点 Q 到根节点路径上的项目组成的条件模式基保存在数组 $Aarray_i$ 中第 $n - Q \rightarrow Level + 1$ 到第 n 个单元中, 将这些元素复制到数组 $Aarray_p$ 中。

2) 若节点 P 所对应的条件模式基指针不为空, 则 $P \rightarrow Condition-Link$ 指向其子孙节点中某个叶子节点的 Condition-Link 域所指示的保存条件模式基的数组 $Aarray_i$, 设数组的中元素个数为 n , 节点 P 的父节点到根节点路径上的项目保存在数组 $Aarray_i$ 中第 $n - P \rightarrow Level + 2$ 到第 n 个单元中, 即数组 $Aarray_i$ 中第 $n - P \rightarrow Level + 2$ 元素到第 n 个元素组成的项目集构成了节点 P 中项目的一条条件模式基。例如: 以搜索图 1 所示 FP-tree 项头表中项目 p 的条件模式基为例, 首先通过 FP-tree 的项头表中的包含项目 p 的单元中链指针找到 FP-tree 包含项目 p 的节点, FP-tree 中有两个包含项目 p 的节点且 Condition-Link 都为 NULL, 根据上述搜索方法, 对于节点 $(p, 2, 5)$, 构建一个包含 4 个元素的数组, 数组中元素为 $\{a, m, f, c\}$, 其前缀路径上每个节点的 Condition-Link 域指向该数组, 如项目 a 前缀路径上项目构成的条件模式基保存在数组第 2 个元素开始的单元中, 即 $\{m, f, c\}$ 。

根据 FP-tree 构建方法可知,任一条件模式基中支持数最小的项目出现在叶子节点上,由于基于 FP-tree 的频繁项目集挖掘算法采用自下而上的挖掘方法,即按照频繁 1-项集的支持数从小到大的顺序产生频繁项目集,因此在 FP-tree 任一分支中必先挖掘包含叶子节点中项目的频繁项目集,根据上述条件模式基的搜索方法,任一叶子节点的 Condition-Link 域不为空值 NULL,其所在分支的非叶子节点的 Condition-Link 域一定不为空值 NULL。因此只需要保存 FP-tree 中每个叶子节点的父节点到根节点路径上项目组成的条件模式基即可,其他节点可以共享某个叶子节点的父节点到根节点路径上项目组成的条件模式基,而没有必要保存 FP-tree 项头表中每个频繁 1-项集的所有条件模式基,而且采用上述方法求条件模式基也只需要遍历 FP-tree 一次。

如图 1 所示 FP-tree 中只需要保存 4 个叶子节点的父节点到根节点路径上项目组成的条件模式基。

假设 FP-tree 中有 n 条分支,每条分支的平均长度为 m ,使用文献[13]中的方法,保存 FP-tree 中所有条件模式基需要 $m \times n \times (m-1)/2$ 个存储单元;而本文提出的方法只需要 $m \times n$ 个存储单元,另外每个节点中增加的两个域所需要存储空间为 $2m \times n$,共需要 $3m \times n$ 存储单元,当 m, n 值较大时, $m \times n \times (m-1)/2 \gg 3m \times n$ 。

2.3 优化策略

对于频繁项目集 $Head$, 设其对应的 FP-tree 中能够和 $Head$ 构成频繁项目集的项目的集合为 $Tail_{Head}$ 。

定理 1 设 $i_j \in Tail_{Head}, i_k \in Tail_{Head \cup i_n}$, 其中 $i_j, i_k, i_n \in I$ 。如果 $i_j = i_k$ 且 $supp(Head \cup i_j) = supp(Head \cup i_n \cup i_k)$, 则由频繁项目集 $Head \cup i_j$ 为前缀构成的频繁项目集一定不是最大频繁项目集。

证明 由 $i_j = i_k$ 可知, $Head \cup i_j \subset Head \cup i_n \cup i_k$, 所以 $supp(Head \cup i_j) \geq supp(Head \cup i_n \cup i_k)$, 即事务数据库中包含频繁项目集 $Head \cup i_j$ 的事务一定包含频繁项目集 $Head \cup i_n \cup i_k$, 又因 $supp(Head \cup i_j) = supp(Head \cup i_n \cup i_k)$ 可得, 事务数据库中包含频繁项目集 $Head \cup i_j$ 的事务和包含频繁项目集 $Head \cup i_n \cup i_k$ 的事务相同, 因此, 以频繁项目集 $Head \cup i_j$ 为前缀构成的频繁项目集是以频繁项目集 $Head \cup i_n \cup i_k$ 为前缀的频繁项目集的子集, 根据性质 1, 定理结论成立。

根据定理 1, 最大频繁项目集搜索过程中, 像满足定理 1 条件的频繁项目集 $Head \cup i_j$ 停止继续搜索, 这样缩小了搜索空间, 加快了搜索的速度, 称这种优化策略为剪枝策略。

以图 1 所示 FP-tree 中项目 a 为例(支持数为 3), 当 $Head = \{a\}$ 时, $Tail_{Head} = \{m, f, c\}, Tail_{Head \cup m} = \{f, c\}, f, c \in Tail_{Head}$ 和 $f, c \in Tail_{Head \cup m}$, 且 $supp(Head \cup f) = supp(Head \cup m \cup f), supp(Head \cup c) = supp(Head \cup m \cup c)$, 根据定理 1, 频繁项目集 $Head \cup f, Head \cup c$ 停止扩展搜索。

定理 2 设频繁项目集 $Head$ 对应的 FP-tree 为 T , 项目 $i \in Tail_{Head}$, 如果在 T 中包含项目 i 的某节点 P 的支持数不小于 min_sup , 且从该节点到根节点路径上项目的集合为 P_r , 则对于节点 P 的前缀路径上任意节点到根节点路径上的项目的集合 P_s , 有 $Head \cup P_s$ 不是最大频繁项目集。

证明 由于 T 中节点 P 的支持数不小于 min_sup , 可得 $supp(Head \cup P_r) \geq min_sup$, 又因 $P_s \subset P_r, (Head \cup P_s) \subset (Head \cup P_r)$, 根据性质 1 可知定理 2 成立。

定理 3 设频繁项目集 $Head$ 对应的 FP-tree 为 T , 项目 $i \in Tail_{Head}$, 如果在 T 中包含项目 i 的某节点 P 支持数不小于 min_sup , 且从该节点到根节点路径上项目的集合为 P_r , 则对

于 T 中包含项目 i 的其他节点到根节点路径上项目的集合 P_l , $P_l \subset P_r$, $Head \cup P_l$ 不是最大频繁项目集。

证明 由于 T 中包含项目 i 的节点的支持数不小于 min_sup , 可得 $supp(Head \cup P_r) \geq min_sup$, 又因 $P_l \subset P_r, (Head \cup P_l) \subset (Head \cup P_r)$, 根据性质 1 可知定理 3 成立。

根据定理 2, 在构建频繁项目集 $Head \cup i' (i' \in Tail_{Head})$ (设 FP-tree 中节点 P 包含项目 i' 且节点 P 在满足定理 2 条件的节点 i 所在节点的前缀路径上) 对应的 FP-tree 时, 节点 P 的前缀路径上的项目不作为构建频繁项目集 $Head \cup i'$ 对应的 FP-tree 的条件模式基。根据定理 3, 对于 FP-tree 中包含项目 i 的节点 P 到根节点路径上项目的集合 P_l , 满足 $P_l \subset P_r$, 则节点 P 的父节点到根节点路径上的项目的集合不作为构建频繁项目集 $Head \cup i$ 对应的 FP-tree 的条件模式基, 这样可以压缩 FP-tree 的规模, 降低存储空间, 提高算法效率, 称利用定理 2, 3 的优化策略为压缩策略。

例如, 以图 1 所示 FP-tree 为例, $Head = \{a\}, a \in Tail_{Head}$, 在构建项目 a 对应的 FP-tree 时, 搜索 a 的条件模式基, 由于 FP-tree 中存在包含项目 a 的节点的支持数为 3, 根据定理 2, FP-tree 中包含该节点的前缀路径上的项目集合 $\{f, c\}$ 不作为构建项目 m 对应 FP-tree 的条件模式基, 项目集合 $\{c\}$ 不作为构建项目 f 对应 FP-tree 的条件模式基。

$Head = \{m\}, m \in Tail_{Head}$, 在构建项目 m 对应的 FP-tree 时, 由于 $\{f, c\}$ 不作为条件模式基, FP-tree 还有一条条件模式基 $\{c\}$, 由于 $\{f, c\}$ 的支持数为 3 且 $\{c\} \subset \{f, c\}$, 根据定理 3, $\{c\}$ 也不作为条件模式基, 所以项目 m 对应的 FP-tree 为空。

2.4 IMMFA 实现步骤

在上面讨论的基础上给出新算法 IMMFA 伪码如下:

IMMFA (T, min_sup)

Input: an FP-tree T, min_sup

Global:

$MFIS$: a set of maximal frequent itemsets

$Head$: a linked list of items

Output: The $MFIS$ that contains all maximal frequent itemsets.

Method:

1) if T only contains a single path P

2) Insert $Head \cup P$ into $MFIS$

//如果当前的 FP-tree 只包含一条分支, 则项目集 $Head$ 与

//FP-tree 单分支 P 中的项目的并集 $Head \cup P$ 为最大频繁项目集

3) else {

4) for each i in Header-table of T {

5) Append i to $Head$

//将当前 FP-tree 项头表中的项目 i 加入到项目集 $Head$ 中

6) Use compression strategy constructing the conditional pattern bases

//搜索构建项目集 $Head$ 对应的 FP-tree 的条件模式基,

//并利用定理 2, 3 的结论对条件模式基进行处理

7) $Tail = \{ \text{frequent items in base} \}$

8) Use pruning strategy in $Tail$

//利用步骤 6) 得到的条件模式基中的频繁项目,

//根据定理 1 的结论对搜索空间进行剪枝处理

9) if $Head \cup Tail$ is not a subset of maximal frequent itemset in $MFIS$ {

10) Construct the FP-tree T_{Head}

11) Call IMMFA (T_{Head}, min_sup)

//构建项目集 $Head$ 对应的 FP-tree, 并递归调用算法

}

12) remove i from $Head$

//从当前递归调用返回继续执行,

//直到搜索到所有最大频繁项目集

}

3 实验结果及分析

3.1 实验环境及实验数据来源

本实验是在 AMD Athlon(tm) 64 X2 1.71 GHz, 1 GB 内存, 120 GB 硬盘, 运行 Microsoft WindowsXP 操作系统的机器上完成。IMMFIA 由 C++ 实现, FpMAX 算法实现代码是由文献作者提供 (<http://users.encs.concordia.ca/~bibdb/dbdm/dm.html>), 实验数据为密集型数据集 connect 和 mushroom。其中: mushroom 数据集有事务数 8 124, 项目个数 119, 最大事务长度 23, 平均长度 22; connect 数据集有事务数 67 557, 项目个数 129, 最大事务长度 43, 平均长度 43。

3.2 实验结果及分析

为了验证算法的正确性和有效性,进行了以下两组实验:

1) 比较 IMMFIA 与 NHTFPG 算法^[13]在 mushroom 数据集上不同支持度下保存由原始事务数据库建立的 FP-tree 的条件模式基所需要的存储单元的比值 S , 结果如图 2 所示。NHTFPG 算法在建立 FP-tree 时不需要项头表, 在实验时, IMMFIA 中所需存储单元包含了保存项头表的空间。

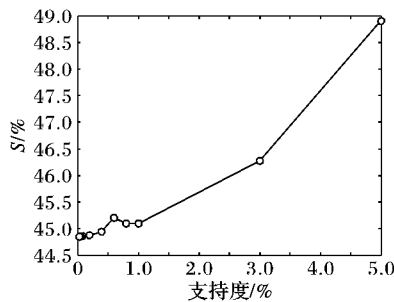


图2 IMMFIA 与 NHTFPG 在 mushroom 数据集上保存条件模式基所需存储单元的比值

从图 2 的实验结果可以看出, IMMFIA 所需保存 FP-tree 的条件模式基的存储单元不到 NHTFPG 算法所需存储单元的 50%, 而且随着支持度的下降优势更加明显。由于基于 FP-tree 的最大频繁项目集挖掘算法在执行过程中需要建立大量的 FP-tree, 因此 IMMFIA 可以有效降低存储条件模式基的存储开销。

2) 比较 IMMFIA 与 FpMax 算法^[8]在两个数据集上不同支持度下的运行时间, 结果如图 3、4 所示。

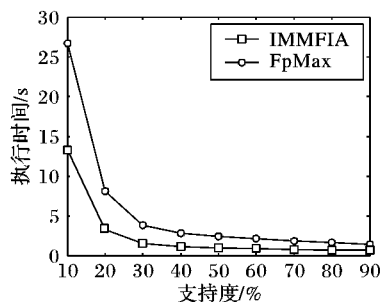


图3 IMMFIA 与 FpMax 在 connect 数据集上的执行时间

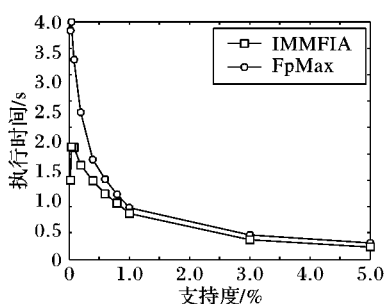


图4 IMMFIA 与 FpMax 在 mushroom 数据集上的执行时间

由于 IMMFIA 对 FP-tree 结构进行修改, 减少了 FP-tree 分支重复遍历的次数, 同时利用剪枝策略和压缩策略, 缩小搜索空间和压缩 FP-tree 的规模, 因此减少了算法的执行时间, 降低了存储空间需求。从图 3、4 的实验结果可以看出随着支持度的下降, IMMFIA 比 FpMax 算法的执行效率提高了 2~3 倍, 因此算法在低支持度下优势更加明显。

4 结语

挖掘最大频繁项目集是频繁项目集挖掘的一个主要研究内容之一, 提出了基于改进 FP-tree 的最大频繁项目集挖掘算法 IMMFIA, 算法对 FP-tree 进行修改, 减少了 FP-tree 重复遍历的次数, 降低了保存条件模式基的存储空间; 同时提出了两种优化策略, 减少了算法执行的时间和空间开销, 提高了算法执行的效率。

参考文献:

- [1] AGRAWAL R, IMIELINSKI T, SWAMI A. Mining association rules between sets of items in large database [C] // Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data. New York: ACM, 1993: 207-216.
- [2] AGRAWAL R, SRIKANT R. Fast algorithms for mining association rules in large database [C] // Proceedings of the 20th International Conference on Very Large Data Bases. San Francisco: Morgan Kaufman, 1994: 478-499.
- [3] HAN J, PEI J, YIN Y. Mining frequent patterns without candidate generation [C] // Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data. New York: ACM, 2000: 1-12.
- [4] LIN D-I, KEDEM Z M. Pincer-search: A new algorithm for discovering the maximum frequent set [C] // Proceedings of the 6th International Conference on Extending Database Technology. Berlin: Springer-Verlag, 1998: 105-119.
- [5] BAYARDO R J. Efficiently mining long patterns from database [C] // Proceedings of the 1998 ACM SIGMOD International Conference on Management of Data. New York: ACM, 1998: 85-93.
- [6] AGGARWAL C, AGRAWAL R, PRASAD V V V. Depth first generation of long patterns [C] // Proceedings of the 6th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining. New York: ACM, 2000: 108-118.
- [7] BURDICK D, CALIMLIM M, GEHRKE J. MAFIA: A maximal frequent itemset algorithm for transactional databases [C] // Proceedings of the 17th International Conference on Data Engineering. Washington, DC: IEEE Computer Society, 2001: 443-452.
- [8] GRAHNE G, ZHU J F. High performance mining of maximal frequent itemsets [C] // Proceedings of the 6th SIAM International Workshop on High Performance Data Mining: Pervasive and Data Stream Mining. San Francisco, CA: [s. n.], 2003: 135-144.
- [9] 刘君强, 孙晓莹, 王勋, 等. 挖掘最大频繁模式的新方法 [J]. 计算机学报, 2004, 27(10): 1328-1333.
- [10] GRAHNE G, ZHU J F. Fast algorithm for frequent itemset mining using FP-trees [J]. IEEE Transactions on Knowledge and Data Engineering, 2005, 17(10): 1347-1362.
- [11] 颜跃进, 李舟军, 陈火旺. 基于 FP-tree 有效挖掘最大频繁项集 [J]. 软件学报, 2005, 16(2): 215-221.
- [12] 任永功, 张亮, 付玉. 一种基于频繁模式树的最大频繁项目集挖掘算法 [J]. 小型微型计算机系统, 2010, 31(2): 317-321.
- [13] 凌绪雄, 王社国, 李洋, 等. 无项头表的 FP-Growth 算法 [J]. 计算机应用, 2011, 31(5): 1391-1394.
- [14] 王丽珍, 周丽华, 陈红梅, 等. 数据仓库与数据挖掘原理及应用 [M]. 北京: 科学出版社, 2005.