计算机系统应用 ISSN 1003-3254, CODEN CSAOBN
Computer Systems & Applications,2019,28(9):154−161 [doi: 10.15888/j.cnki.csa.007058]
©中国科学院软件研究所版权所有.
E-mail: csa@iscas.ac.cn
http://www.c-s-a.org.cn
Tel: +86-10-62661041

改进的频繁模式挖掘算法①

魏恩超[1], 张德生[1], 安平平[2]

[1](西安理工大学 理学院, 西安 710054)
[2](西北师范大学 教育学院, 兰州 730070)
通讯作者: 魏恩超, E-mail: 1178044154@qq.com

摘　要: 为解决传统频繁模式挖掘算法效率不高的问题, 提出了一种改进的基于 FP-tree (Frequent pattern tree) 的 Apriori 频繁模式挖掘算法. 首先, 在 Apriori 算法的连接步加入连接预处理过程; 其次, 对 CP-tree (Compact Pattern tree) 进行扩展, 构造了一个新的树结构 ECP-tree (Extension of Compact Pattern tree), 新的树结构只需对数据库进行一次扫描就能构造出一棵紧凑的前缀树, 且支持交互式挖掘与增量挖掘; 然后, 将改进点与 APFT 算法结合, 用于挖掘频繁模式; 最后, 使用 UCI 数据库中两个数据集进行实验. 实验结果表明: 改进算法具有较高的挖掘效率, 频繁模式挖掘速度显著提升.

关键词: 频繁模式; 连接预处理; ECP-tree 结构; 交互式挖掘; 增量挖掘

引用格式: 魏恩超,张德生,安平平.改进的频繁模式挖掘算法.计算机系统应用,2019,28(9):154–161. http://www.c-s-a.org.cn/1003-3254/7058.html

Improved Frequent Patterns Mining Algorithm

WEI En-Chao[1], ZHANG De-Sheng[1], AN Ping-Ping[2]

[1](Faculty of Sciences, Xi'an University of Technology, Xi'an 710054, China)
[2](College of Education, Northwest Normal University, Lanzhou 730070, China)

Abstract: In order to solve the problem that the low efficiency of traditional frequent patterns mining algorithm, an improved Apriori algorithm based on FP-tree is proposed. Firstly, the join preprocessing process is added to the join step of Apriori algorithm. Secondly, the CP-tree is extended to construct a new tree structure, ECP-tree. The new tree structure can construct a compact prefix tree with only one scan of the database, and support interactive mining and incremental mining. Then, the improved points are combined with the APFT algorithm for mining frequent patterns. Finally, experiments are performed using two datasets in the UCI database. The experimental results show that the improved algorithm has higher mining efficiency and the frequent pattern mining speed is significantly improved.

Key words: frequent patterns; join preprocessing; ECP-tree structure; interactive mining; incremental mining

数据挖掘 (DM, Data Mining) 是近年来研究、开发和应用最活跃的一个领域, 它是在机器学习、神经网络和模式识别的理论基础上发展而来的, 其任务是从海量数据中提取隐含且用户感兴趣的知识和信息[1].

关联规则挖掘是数据挖掘中最重要的研究领域之一, 其本质是对频繁模式的挖掘, 最早由 Agrawal 等提出[2], 目的是发现模式之间隐含的内在联系, 为后续问题的解决提供决策支持. 关联规则挖掘主要包含两个子问题: 一是生成频繁模式集, 其任务是根据用户设置的最小支持度阈值生成所有频繁模式; 另一个是生成关联规则, 主要任务是从所有的频繁模式中提取满足最小置信度阈值的规则.

频繁模式挖掘算法大致可分为两类: 一种是类 Apriori 算法, 如文献[3]提出了 Inter-Apriori 频繁模式

① 收稿时间: 2019-02-27; 修改时间: 2019-03-22; 采用时间: 2019-03-29; csa 在线出版时间: 2019-09-05

© 中国科学院软件研究所 http://www.c-s-a.org.cn

挖掘算法, 该算法使用交集策略来减少扫描数据库的次数, 使算法达到了较高的效率; 文献[4]提出了一种基于三角矩阵和差集的垂直数据格式挖掘频繁模式的算法; 文献[5]提出了一种基于前缀项集的候选集存储结构, 并利用哈希表进行快速查找, 提高了经典 Apriori 算法在连接步和剪枝步中的效率; 文献[6]提出了一种基于矩阵的改进算法, 通过事务矩阵和候选项集项目矩阵相乘的矩阵操作改进频繁扫描数据库的问题, 优化了 Apriori 算法的连接步与剪枝步. 这些算法都是由经典的 Apriori 算法改进而来, 由于 Apriori 算法的固有缺陷, 使得这些算法的挖掘效率仍然不高. 另一类是基于频繁模式增长的算法, 如文献[7]提出了一种基于频繁模式树 (FP-tree) 的 FP-growth 算法; 文献[8]提出了一种基于 COFI-Tree 挖掘 N-最有兴趣项集算法, 该算法采用 COFI-Tree 结构, 无需递归构造条件子树 FP-Tree; 文献[9]提出了一种基于邻接表的改进 FP-Growth 算法, 该算法使用哈希表存储邻接表, 可以快速删除小于最小支持阈值的模式. 这类算法利用树结构对数据库进行压缩, 不产生候选项集, 减少了数据库的扫描次数, 但在对大型稠密数据集进行挖掘时会构造大量条件模式基和条件 FP-tree, 对内存消耗非常大.

为克服前两类算法的局限性, 一些研究人员将 Apriori 算法和 FP-tree 结构结合[10–14]. 这类算法先将整个数据库投影到 FP-tree 上, 然后对 FP-tree 进行分区, 将数据库划分成若干个子数据集, 最后利用 Apriori 算法的候选生成-测试机制对子数据集进行挖掘. 文献[10]首次将 Apriori 算法与 FP-tree 结合, 提出了相应的 APFT 算法, 该算法不需要递归地生成条件模式基和条件 FP-tree, 有效提高了挖掘效率. 文献[11]在 FP-tree 的基础上提出了压缩频繁模式树 (CFP-tree), 并从最不频繁的项开始划分生成子树, 并在每棵子树上执行候选项集生成机制, 算法避免了递归生成大量子树, 提高了模式挖掘速度; 文献[12]利用频繁 1-项集对数据库进行分库, 统计候选项集支持度计数时只扫描分库, 减少了 Apriori 算法对数据库的遍历次数; 文献[13]提出了一种改进的基于 fp-tree 的 Apriori 算法, 该算法先用尾元将 fp-tree 分区, 生成数据量更小的子数据集, 再动态删除冗余数据将子数据集进一步压缩, 最后通过扫描子数据集进行支持数统计; 文献[14]将 Apriori 算法与 FP-growth 算法结合, 提出了一种基于事务映射区间求交的频繁模式挖掘算法 IITM, 只需扫描数据集两次来生成 FP-tree, 然后扫描 FP-tree 将每个项的 ID 映射到区间中, 通过区间求交进行模式增长. 但将 FP-tree 结构与 Apriori 算法结合后仍存在以下不足: (1) Apriori 算法在连接步会花费大量时间判断项集是否满足连接条件; (2) 构建 FP-tree 时需要对数据库进行两次扫描, 且不支持交互式挖掘与增量挖掘.

为解决这些问题, 本文在 APFT 算法的基础上提出了一种改进的频繁模式挖掘算法. 主要进行了如下改进: (1) 在算法的连接步前加入预处理过程, 减少比较次数; (2) 提出了一种新的紧凑前缀树结构, 只需对数据库进行一次扫描就可以构造出一棵紧凑的模式树, 且支持交互式挖掘与增量挖掘, 能有效处理数据流问题.

1 问题陈述

1.1 频繁模式挖掘

设$I=\{I_1,I_2,\cdots,I_m\}$是所有项的集合, 不失一般性, I中的项按字典顺序排列, 即$I_1<\cdots<I_m$. T 为一个事务, 每个事务都由唯一的标识符 TID 和与之对应的项集$Item(T)(Item(T)\subseteq I)$构成, 所有事务构成了事务数据库 D, D 中包含事务的个数称为数据库的大小, 记为$Size(D)$. 设$X\subseteq I$为一个项集, 项集中包含项的个数叫做项集的长度, 记为$Length(X)$, 即$Length(X)=|X|$, 长度为k的项集称为k-项集. 事务 T 包含项集 X, 当且仅当$X\subseteq Item(T)$, 项集 X 的支持度计数$\sigma(X)$定义为数据库D中包含项集X的事务数, 即$\sigma(X)=|\{T|T\in D\wedge X\subseteq Item(T)\}|$, 项集$X$的支持度$sup(X)$定义为包含项集$X$的事务在数据库$D$中所占的比例, 即$sup(X)=\sigma(X)/Size(D)$. 项集 X 为频繁模式, 当且仅当$sup(X)\geqslant minsup$, 其中$minsup$为给定的最小支持度阈值, 长度为k的频繁项集称为频繁k-项集, 记为F_k.

频繁模式挖掘定义为, 在给定的数据库 D 和最小支持度阈值$minsup$条件下, 找出所有频繁模式$FI=F_1\cup F_2\cup\cdots\cup F_l$, l为频繁模式的最大长度.

1.2 Apriori 算法

Apriori 算法[2]最早由 Agrawal 等提出, 该算法使用先验性质压缩搜索空间, 并使用一种逐层搜索的迭代方法, 其中k-项集用于探索 $(k+1)$-项集. Apriori 算法主要包含以下 3 个步骤:

连接步: 这一步主要是为了生成候选k-项集C_k, 设l_1和l_2是频繁$(k-1)$项集F_{k-1}中的项集, 记号$l_i[j]$表示l_i的第 j 项, 项集中的项按字典顺序排列, 如果有$(l_1[1]=$

© 中国科学院软件研究所 http://www.c-s-a.org.cn

$l_2[1]) \wedge \cdots \wedge (l_1[k-2] = l_2[k-2]) \wedge (l_1[k-1] < l_2[k-1])$，则连接 l_1 和 l_2 生成候选 k-项集 C_k，其中 $C_k = \{l_1[1], l_1[2], \cdots, l_1[k-1], l_2[k-1]\}$.

剪枝步: 利用频繁项集的反单调性对候选 k-项集 C_k 进行剪枝, 生成频繁 k-项集 F_k. 如果候选 k-项集的一个 $(k-1)$ 项子集不在 F_{k-1} 中, 则该候选项集为非频繁项集, 否则需要遍历数据库进行支持数统计, 进一步验证 C_k 是否频繁.

统计支持度计数: 扫描数据库 D, 对候选 k-项集 C_k 在数据库中出现的次数进行累加, 根据给定的最小支持度阈值 $minsup$ 生成频繁 k-项集.

1.3 FP-tree 结构

为克服 Apriori 算法的缺陷, 韩家炜等提出了基于 FP-tree 的 FP-Growth 算法[7], 该算法只需对数据库进行两次扫描, 且不产生候选项集, FP-tree 结构能对数据库进行有效压缩.

FP-tree 中每个节点由 4 个域组成: 项标识 item-name、节点计数 item-count、节点链 item-link 和父节点指针 item-parent. 同时为了方便对树进行操作, FP-tree 还包含一个频繁项头表 header-table, 它由两个域组成: 项标识 item-name 和项元链链头 node-link, 其中 node-link 是指向 FP-tree 中具有相同项标识的首节点指针, FP-tree 中实线是父节点指针, 虚线是节点链. FP-tree 相关概念及结构细节见文献[1]. 使用表 1 中的示例数据生成的 FP-tree 如图 1 所示, 设最小支持度阈值 $minsup = 3$.

表 1 样例数据集

TID	Transaction	Frequent Items
001	f, a, c, d, g, i, m, p	f, c, a, m, p
002	a, b, c, f, l, m, o	f, c, a, b, m
003	b, f, h, j, o	f, b
004	b, c, k, s, p	c, b, p
005	a, f, c, e, l, p, m, n	f, c, a, m, p

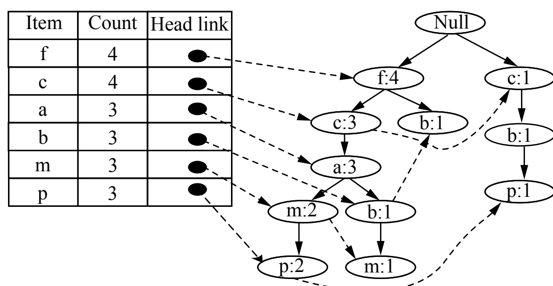

图 1 FP-tree 结构

1.4 APFT 算法

APFT 算法[10]使用 Apriori 算法挖掘基于 FP-tree 的频繁模式, 该算法仍采用分治策略. APFT 算法主要包括两个步骤, 第一步是利用 FP-growth 算法中构造树的方法构造 FP-tree; 第二步是利用 Apriori 算法挖掘 FP-tree. 在第二步中, 需要添加一个名为NTable的附加节点表, 表中的每个项由两个域组成: Item-name 和 Item-support. Item-name 表示在条件子树 $FPT_j(j = 1,2,\cdots,n)$ 中节点的名称, Item – support 表示与频繁 1-项集 $I_j(j = 1,2,\cdots,n)$ 一起出现的节点数, 即项的支持度计数. APFT 是最早将 Apriori 算法与 FP-tree 结合的算法, 但由于 Apriori 算法与 FP-tree 自身的缺陷导致该算法仍存在一些不足, 因此本文对 APFT 中的连接步与树构造方法进行了改进, 使算法能有效处理数据流问题. APFT 算法的伪代码如算法 1 所示.

算法 1. APFT

Input: FP-tree, *minsup*
Output: all frequent patterns L
1) L=L_1;
2) **for** each item I_j in header table, in top down order
3) L_{I_j}=Apriori−mining(I_j);
4) **return** L=$L \cup L_{I_1} \cup L_{I_2} \cup \cdots \cup L_{I_n}$;
Pseudocode Apriori−mining(I_j)
1) Find item p in the header table which has the same name with I_j;
2) q=p.tablelink;
3) **while** q is not null
4) **for** each node q_j!=root on the prefix path of q
5) **if** *NTable* has a entry *N* such that *N*.Item-name=q_j.item-name
6) N.Item−support=N.Item−support+q.count;
7) **else**
8) add an entry *N* to the *NTable*;
9) $N.Item\text{−}name$=$q_j.item\text{−}name$;
10) $N.Item\text{−}support$=$q.count$;
11) q=$q.tablelink$;
12) k=1;
13) F_k=$\{i | i \in NTable \wedge i.\text{item−support} \geq minsup\}$
14) **repeat**
15) k=k+1;
16) C_k=$apriori\text{−}gen(F_{k-1})$;
17) q=$p.tablelink$;
18) **while** q is not null
19) find prefix path t of q
20) C_t=$subset(C_k,t)$;
21) **for** each $c \in C_t$
22) $c.support$=$c.support$+$q.count$;
23) q=$q.tablelink$;
24) F_k=$\{c | c \in C_k \wedge c.support \geq min_Sup\}$
25) **until** F_k=$\emptyset$
26) **return** L_{I_i}=$I_i \cup F_1 \cup F_2 \cup \cdots \cup F_k$

© 中国科学院软件研究所 http://www.c-s-a.org.cn

2 改进算法

2.1 算法改进思想

为了提高基于 FP-tree 的 Apriori 算法的挖掘效率, 从进一步缩减数据库扫描次数以及增强算法的交互式挖掘与增量挖掘的方向改进. 首先, 将 Apriori 算法的 apriori_gen() 函数中的连接步进行优化. 由于连接步需要大量比较步骤, 并且会产生大量无用的候选项集, 因此增加了后续剪枝步和支持度计数步的时间开销. 其次, 将 CP-tree[15]进行扩展, 提出了一个新的树结构 ECP-tree(Extension of Compact Pattern tree). 由于在构造 CP-tree 的过程中包含所有项, 不论是频繁项还是非频繁项都参与了树的构造, 因此占用了大量的内存空间, 并且会影响频繁模式的挖掘效率. 在新构造的 ECP-tree 中只包含频繁项, 删除了非频繁项, 并且只经过一次数据库扫描就可以构造出一棵紧凑的前缀树, 新构造的树结构不仅支持交互式挖掘而且也支持增量挖掘.

2.2 优化连接步

在给出优化思想之前, 首先了解两个关于集合的性质.

性质 1. 包含 k 个不同项的 k-项集有 k 个不同的 $(k\text{–}1)$ 项子集;

性质 2. 假设 I 为 k-项集中的一个项, 根据性质 1 可得, 在 k 个子集中必有 $(k\text{–}1)$ 个子集包含项 I.

在 Apriori 算法的 apriori_gen() 函数的连接步中, 需要进行多次前 $(k\text{–}1)$ 项的比较, 这大大增加了算法的时间开销. 经研究发现, 存在这样一个性质:

性质 3. 假设I为频繁k-项集F_k中的一个项, 如果包含I的频繁k-项集还能产生频繁$(k+1)$-项集, 则包含I的频繁k-项集的总数不小于k, 如果项集总数小于k, 那么F_k就不用参与后续的连接步. 证明过程如下:

反证法: 假设包含项I的F_k参与了后续的连接步, 那么就会生成包含项 I 的候选 $(k+1)$-项集C_{k+1}. 如果候选项集C_{k+1}不被删除, 则必须满足C_{k+1}的 k+1 个 k 项子集必须都是频繁的, 即这 k+1 个 k 项子集必须在频繁 k-项集中, 根据性质 2 可得, 在 k+1 个 k 项子集中有 k 个子集包含项 I, 也就是说包含项 I 的项集总数不小于k. 反之, 如果包含项 I 的项集总数小于k, 那么连接生成的C_{k+1}必然会被剪掉.

根据性质 3, 可对连接步进行优化, 在连接之前进行预处理, 删除不满足条件的F_k. 预处理过程的伪代码如下所示:

算法 2. Pretreatment

```
Input: 频繁 k-项集 F_k
Output: 满足连接条件的频繁k-项集L_k
1) for each items F_i∈F_k
2)   for each item I∈F_i
3)     if I∈F_k
4)       I.count ++
5)   for each item I∈F_i
6)     if I.count < k
7)       delete F_i from F_k
8) return L_k
```

为了说明优化流程, 用以下示例说明具体过程. 假设表 2 是事务数据库产生的所有频繁 3-项集F_3.

表 2 所有频繁 3-项集

TID	F_3	TID	F_3
001	I_2,I_3,I_4	007	I_3,I_4,I_8
002	I_2,I_3,I_5	008	I_3,I_4,I_9
003	I_2,I_3,I_6	009	I_3,I_4,I_{10}
004	I_2,I_3,I_7	010	I_1,I_2,I_{10}
005	I_2,I_3,I_{10}	011	I_2,I_4,I_{10}
006	I_3,I_4,I_7	012	I_5,I_6,I_8

在原始 Apriori 算法中, 保留所有频繁 3-项集进入连接步, 连接产生的候选 4-项集C_4(未剪枝的候选项集) 如表 3 所示.

表 3 所有候选 4-项集

TID	C_4	TID	C_4
001	I_2,I_3,I_4,I_5	009	I_2,I_3,I_6,I_{10}
002	I_2,I_3,I_4,I_6	010	I_2,I_3,I_7,I_{10}
003	I_2,I_3,I_4,I_7	011	I_3,I_4,I_7,I_8
004	I_2,I_3,I_4,I_{10}	012	I_3,I_4,I_7,I_9
005	I_2,I_3,I_5,I_6	013	I_3,I_4,I_7,I_{10}
006	I_2,I_3,I_5,I_7	014	I_3,I_4,I_8,I_9
007	I_2,I_3,I_5,I_{10}	015	I_3,I_4,I_8,I_{10}
008	I_2,I_3,I_6,I_7	016	I_3,I_4,I_9,I_{10}

如表 3 所示, 表 2 中的 12 个频繁 3-项集连接后共产生 16 个候选 4-项集, 根据剪枝原理, 剪枝后剩下的候选 4-项集只有$\{I_2,I_3,I_4,I_{10}\}$. 只有这一个候选 4-项集进入支持度计数步. 连接步的预处理过程如下所示:

第一步: 计算表 2 中 12 个频繁 3-项集中每一个项的计数, 结果如表 4 所示.

第二步: 由表 4 可知, 项 I_1、I_5、I_6、I_7、I_8、I_9 的

© 中国科学院软件研究所 http://www.c-s-a.org.cn

计数小于 k(即, 3), 因此对所有包含项 I_1、I_5、I_6、I_7、I_8、I_9 的频繁 3-项集F_3进行剪枝, 剪枝后的频繁 3-项集L_3如表 5 所示.

表 4 频繁 3-项集中每个项计数

项 (I)	计数 (I.count)	项 (I)	计数 (I.count)
I_1	1	I_6	2
I_2	7	I_7	2
I_3	9	I_8	2
I_4	6	I_9	1
I_5	2	I_{10}	4

表 5 预处理后的频繁 3-项集

TID	L_3	TID	L_3
001	I_2,I_3,I_4	003	I_3,I_4,I_{10}
002	I_2,I_3,I_{10}	004	I_2,I_4,I_{10}

预处理后只剩 4 个频繁 3-项集, 按照连接规则将这 4 个频繁项集进行连接, 生成一个候选 4-项集$\{I_2,I_3,I_4,I_{10}\}$, 对这个候选项集进行剪枝操作, 没有被剪掉, 进入支持度计数步. 预处理后的结果与原始 Apriori 算法结果相同, 故预处理步骤是合理的.

2.3 新树的构造

新树的构造主要包括两个阶段: 1) 插入阶段; 2) 重构阶段. 在插入阶段, 通过逐一扫描数据库中的事务项集, 根据项在事务中出现的先后顺序将它们插入树中. 由于将事务插入树中时, 它维护I–list并更新I–list中各个项的支持度计数, 因此, 在插入所有事务后, 树结构在I–list中具有数据库中所有项的支持度计数. 在重构阶段, 根据项的支持计数以降序方式重新排列I–list, 只保留频繁项, 即支持度大于等于用户指定的最小支持度阈值的项, 并根据新排列的I–list重新构造树节点. 在树重构中, 使用分支排序方法 (BSM, Branch Sorting Method)[15], 该方法逐一对未排序的路径进行排序并按项的支持度计数降序排列I–list实现重构.

本文对分支排序方法 (BSM) 进行了改进, 使该算法能够适应新提出的树结构. 在改进方法中, 如果路径没有按照新的排序顺序进行排序, 则将该路径删除, 并删除非频繁的项, 将剩余的频繁项按排序顺序排列到一个临时数组中, 最后再按顺序插入树中, 重复此过程最终将得到一棵紧凑的模式树. 改进的分支排序方法 (IBSM, Improve Branch Sorting Method) 的伪代码如算法 3 所示.

算法 3. IBSM

Input: T 和 I
Output: 仅包含频繁项的 Tsort 和 Isort
1) Compute I_{sort} from I in frequency-descending order using merge sort technique
2) **for** each branch B_i in T
3) **for** each unprocessed path P_j in B_i
4) **if** P_j contains only frequent items in order according to I_{sort}
5) Process_Branch(P_j)
6) **else** Sort_path(P_j)
7) Terminate when all the branches are sorted and output T_{sort} ang I_{sort}.
8) **Process_Branch**(P){
9) **for** each branching node n_b in P from the $leaf_P$ node
10) **for** each sub-path from the $leaf_k$ with $k \neq p$
11) **if** item ranks of all nodes between n_b and $leaf_k$ are greater than that of n_b
12) P=sub-path from n_b to $leaf_k$
13) **if** P is contains only frequent items in order according to I_{sort}
14) Process_Branch(P)
15) **else** P=path from the root to $leaf_k$
16) Sort_path(P)
17) }
18) **Sort_ path**(Q){
19) Reduce the count of all node of Q by the value of $leaf_Q$ count
20) Delete all nodes contains infrequent items and sort remaining nodes in an array according to I_{sort} order
21) Delete all nodes having count zero from Q
22) Insert sorted Q into T at the location from where it was taken
23)}

为了更好地理解改进树重构方法的工作原理, 使用表 1 给出的样例数据库进行详细说明, 设最小支持度阈值$minsup$=3. 图 2 为插入阶段后的树结构 T 和项列表 I, 此过程与原始分支排序方法中的插入阶段略有不同. 由于项列表 I 和前缀树 T 没有按项的支持度计数降序排列, 因此, 为了使用 IBSM 以项的降序重构树, 首先对 I 进行排序以生成只包含频繁项的新列表 I_{sort}, 如图 3 所示. 重构从树的第一个分支开始, 即最左边的分支, 从图 2 所示树的根节点开始. 由于分支的第一个路径{f : 1, a : 1, c : 1, d : 1, g : 1, i : 1, m : 1, p : 1}是一个未排序的路径, 因此将该路径从树中移除, 移除后的树如图 4 所示, 将移除的路径存放在一个临时数组中并按 I_{sort} 中项的顺序对余下的频繁项进行重新排列, 排序后的路径为{f : 1, c : 1, a : 1, m : 1, p : 1}, 如图 5 所示. 然后再按顺序插入树中, 图 6 显示了第一个路径排序完成后的树结构. 重复此过程, 直到所有路径都完成排序, 图 7 显示了重构后的最终树.

© 中国科学院软件研究所 http://www.c-s-a.org.cn

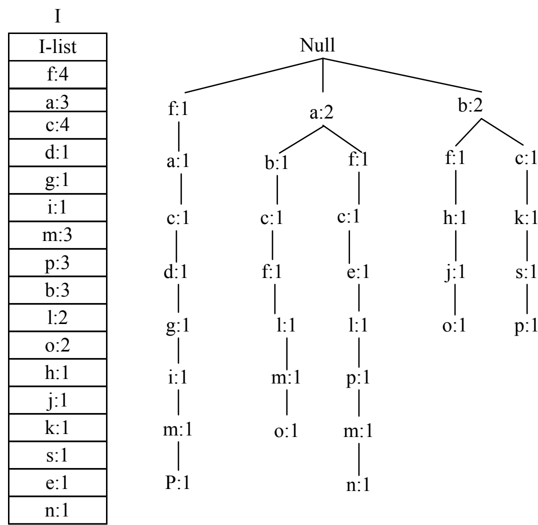

图 2　插入阶段后的 I 和 T

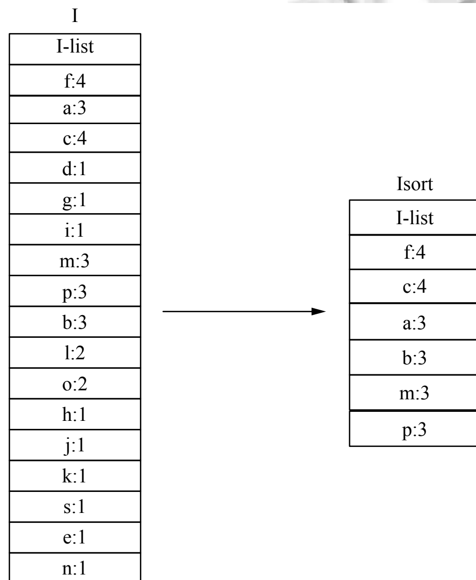

图 3　重排 I 构造 I_{sort}

新提出的树结构只需对数据库进行一次扫描, 就可以构造出一棵与 FP-tree 完全相同的树. 新的树结构支持交互式挖掘, 在交互式挖掘中, 用户可以为同一个数据库更改指定的最小支持度阈值. 在这种情况下, 我们可以在插入所有事务后保存树, 然后根据用户指定的最小支持度阈值重构树, 在I – list中保存着所有项的总支持度计数, 可以根据不同的最小支持度阈值只保留频繁项, 而不需要重新扫描数据库. 因此, 新提出的树结构不需要像 FP-tree 那样重新扫描数据库以实现交互式挖掘. 新的树结构也支持增量挖掘, 在增量挖掘中, 可以对原始数据库中的事务进行添加和/或删除. 在这种情况下, 我们可以在原始数据库中插入所有事务后保存树, 当添加新事务和/或删除一些旧事务时, 可以在树中添加新事务的分支并且删除那些被删除的事务分支, 重构树不需要像 FP-tree 那样再次扫描原始数据库.

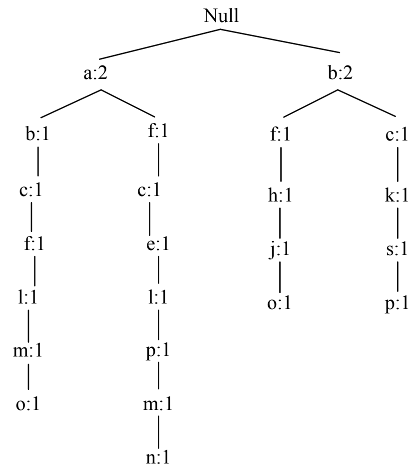

图 4　移除未排序路径

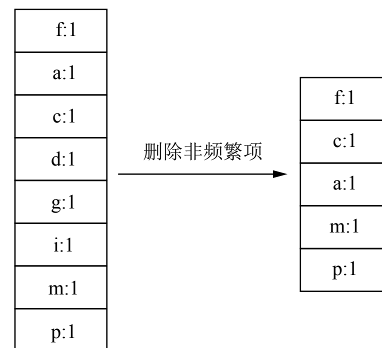

图 5　在临时数组中对路径排序

3　实验分析

为验证所提方法的有效性, 将改进方法运用到 APFT 算法中来挖掘频繁模式, 且与原始 APFT 算法、FP-Growth 算法以及 Apriori 算法进行比较. 本文算法实验平台为: Intel Core i7-4790 CPU 3.6 GHz 处理器和 8 GB 内存, Windows 10 64 位操作系统, 开发软件为 Matlab R2014a. 实验所用的数据集有两个, mushroom 数据库和 T10I4D100K 数据库, 均可从 http://fimi.ua.ac.be/data/.下载, 数据集的特征如表 6 所示. 本文分别在这两种数据集上进行对比试验, 每组试验的参数都是不变的, 都是通过改变最小支持度阈值从而记录算法的运行时间, 算法运行的时间越小代表算法的挖掘速度越快.

© 中国科学院软件研究所 http://www.c-s-a.org.cn

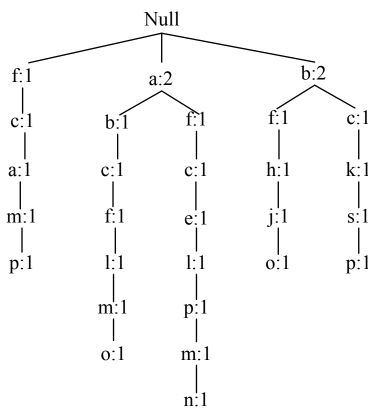

图 6 将排序后的路径插入 T 中

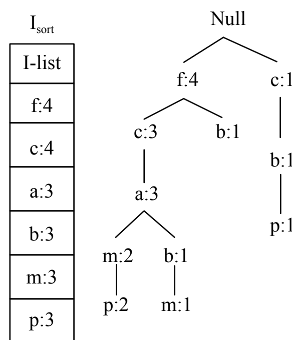

图 7 重构后的最终树

表 6 数据库特征

Database	Items	Rcords	Max|T|	Avg|T|
mushroom	119	8124	23	23
T10I4D100K	870	100 000	29	10

在相同条件下, 算法独立实验 10 次, 取其运行时间的均值作为算法最终结果. 两个数据集在不同算法和不同最小支持度阈值下的运行时间对比如表 7 和表 8 所示, 单位为秒 (即, s). 从表中可以看出, 本文改进算法在运行时间上明显优于其他几个算法, 说明本文改进算法有效提高了频繁模式的挖掘速度. 从图 8 和图 9 中可以看出本文改进算法比原始 APFT 算法更高效, 主要原因有两个: 1) 在算法的 apriori_gen() 函数前加入了连接预处理步骤, 对进入连接步的频繁 k-项集进行剪枝, 减少了两两频繁项集前 $(k-1)$ 项的比较次数, 因此减小了连接步的时间消耗, 并且候选项集的数量也减少了, 减小了剪枝步的时间开销; 2) 新提出的树构只需对数据库进行一次扫描就可以构造出一颗紧凑的模式树, 虽然增加了树重构过程但这一过程基本上不会花费太多时间, 并且在挖掘过程中不需要额外的空间, 因此本文改进算法具有更好的空间可扩展性.

表 7 mushroom 上不同算法运行时间对比

最小支持度阈值 (%)	4	5	6	7	8	9
Apriori 算法	675	300	130	90	75	45
FP-Growth 算法	110	80	65	40	35	20
APFT 算法	90	75	50	30	15	9
本文改进算法	**60**	**53**	**33**	**24**	**10**	**5**

表 8 T10I4D100K 上不同算法运行时间对比

最小支持度阈值 (%)	1	2	3	4	5	6
Apriori 算法	6.8	4.3	3.4	3.1	2.9	2.7
FP-Growth 算法	5.7	3.5	2.9	2.7	2.5	2.2
APFT 算法	4.8	3.1	2.9	2.8	2.4	2.2
本文改进算法	**3.8**	**2.7**	**2.3**	**1.9**	**1.7**	**1.3**

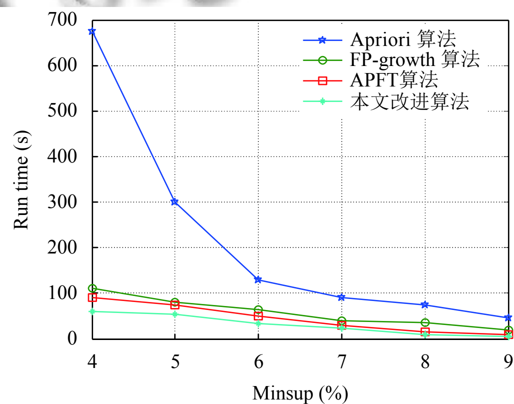

图 8 mushroom 上运行时间对比

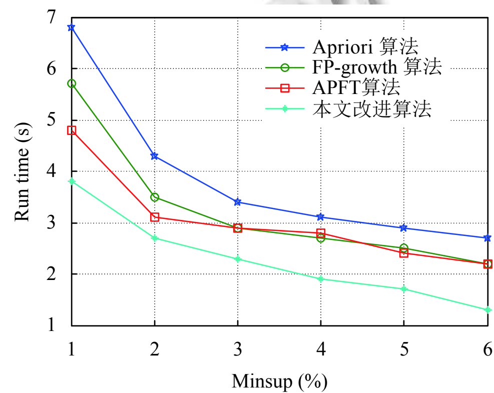

图 9 T10I4D100K 上运行时间对比

4 结束语

本文针对传统频繁模式挖掘算法存在的固有缺陷, 提出了一种基于 APFT 算法的改进频繁模式挖掘算法. 首先, 在算法的连接步前加入预处理过程, 对参与连接的频繁 k-项集进行有效剪枝, 大大减少了连接步与剪枝步的时间开销; 其次, 对 CP-tree 进行扩展提出了一

© 中国科学院软件研究所 http://www.c-s-a.org.cn

种新的树结构 ECP-tree, 新的树结构是一棵紧凑的前缀模式树; 然后, 再将改进点与 APFT 算法结合用于频繁模式挖掘; 最后, 利用实验验证了改进算法的有效性.

为了有效对频繁模式进行挖掘, 本文将改进点与 APFT 算法结合, 由于新提出的连接预处理方法与紧凑树结构具有较好的可移植性, 因此改进点可与其它类似算法结合, 且并不影响挖掘效率, 相应地还能增强原始算法处理数据流问题的能力.

下一步的研究工作包括: 1) 继续完善算法, 往并行计算方向扩展; 2) 将该算法的思想扩展到最大、闭频繁模式的挖掘领域.

参考文献

1 Han JW, Kamber M, Pei J. 数据挖掘: 概念与技术. 范明, 孟小峰, 译. 3 版. 北京: 机械工业出版社, 2012.

2 Agrawal R, Imieliński T, Swami A. Mining association rules between sets of items in large databases. ACM SIGMOD Record, 1993, 22(2): 207–216. [doi: 10.1145/170036]

3 刘步中. 基于频繁项集挖掘算法的改进与研究. 计算机应用研究, 2012, 29(2): 475–477. [doi: 10.3969/j.issn.1001-3695.2012.02.019]

4 邢长征, 安维国, 王星. 垂直数据格式挖掘频繁项集算法的改进. 计算机工程与科学, 2017, 39(7): 1365–1370. [doi: 10.3969/j.issn.1007-130X.2017.07.025]

5 于守健, 周羿阳. 基于前缀项集的 Apriori 算法改进. 计算机应用与软件, 2017, 34(2): 290–294. [doi: 10.3969/j.issn.1000-386x.2017.02.052]

6 王蒙, 邹书蓉, 方睿. 一种基于矩阵的 Apriori 改进算法. 信息技术, 2018, (3): 150–154.

7 Han JW, Pei J, Yin YW. Mining frequent patterns without candidate generation. Proceedings of 2000 ACM SIGMOD International Conference on Management of Data. Dallas, Texas, USA. 2000. 1–12.

8 肖继海, 崔晓红, 陈俊杰. 基于 COFI-Tree 的 N-最有兴趣项目集挖掘算法. 计算机技术与发展, 2012, 22(3): 99–102. [doi: 10.3969/j.issn.1673-629X.2012.03.026]

9 Yin M, Wang WJ, Liu Y, *et al*. An improvement of FP-Growth association rule mining algorithm based on adjacency table. MATEC Web of Conferences, 2018, 189(1): 10012.

10 Lan QH, Zhang DF, Wu B. A new algorithm for frequent itemsets mining based on apriori and FP-tree. Proceedings of 2009 WRI Global Congress on Intelligent Systems. Xiamen, China. 2009. 360–364.

11 吴倩. 基于压缩 FP-tree 的频繁项集快速挖掘算法研究[硕士学位论文]. 上海: 华东理工大学, 2015.

12 张宁. 基于 FP-tree 的 Apriori 算法的改进. 信息通信, 2015, (2): 94–95. [doi: 10.3969/j.issn.1673-1131.2015.02.056]

13 倪政君, 夏哲雷. 一种基于 fp-tree 的 Apriori 算法改进研究. 中国计量大学学报, 2018, 29(1): 50–54. [doi: 10.3969/j.issn.2096-2835.2018.01.009]

14 吴磊, 程良伦, 王涛. 基于事务映射区间求交的高效频繁模式挖掘算法. 计算机应用研究, 2019, 36(4). [doi: 10.19734/j.issn.1001-3695.2017.10.0972.]

15 Tanbeer SK, Ahmed CF, Jeong BS, *et al*. Efficient single-pass frequent pattern mining using a prefix-tree. Information Sciences, 2009, 179(5): 559–583. [doi: 10.1016/j.ins.2008.10.027]

© 中国科学院软件研究所 http://www.c-s-a.org.cn