文章编号:1001 - 9081(2010)07 - 1922 - 04

关联规则中 FP-tree 的最大频繁模式非检验挖掘算法

惠 亮,钱雪忠
(江南大学 信息工程学院,江苏 无锡 214122)
(huiliangjd1@126.com)

摘 要:基于 FP-tree 的最大频繁模式挖掘算法是目前较为高效的频繁模式挖掘算法,针对这些算法需要递归生成条件 FP-tree、做超集检验等问题,在分析 DMFIA-1 算法的基础上,提出了最大频繁模式的非检验挖掘算法 NCMFP。该算法改进了 FP-tree 的结构,使挖掘过程中不需要生成条件频繁模式树也不需要超集检验。算法采用的预测剪枝策略减少了挖掘的次数,采用的求取公共交集的方式保证了挖掘结果的完整性。实验结果表明在支持度相对较小情况下,NCMFP 的效率是同类算法的 2 ~ 5 倍。

关键词:关联规则;数据挖掘;频繁模式树;最大频繁项集;超集检验
中图分类号: TP311.13 **文献标志码**:A

Non-check mining algorithm of maximum frequent patterns in association rules based on FP-tree

HUI Liang, QIAN Xue-zhong
(*School of Information Engineering, Jiangnan University, Wuxi Jiangsu* 214122, *China*)

Abstract: The algorithms based on FP-tree, for mining maximal frequent patterns, have high performance but with many drawbacks. For example, they must recursively generate conditional FP-trees, have to do the process of superset checking. In order to overcome these drawbacks of the existing algorithms, an algorithm Non-Check Mining algorithm of Maximum Frequent Pattern (NCMFP) for mining maximal frequent patterns was put forward after the analysis of DMFIA-1 algorithm. In the algorithm, neither constructing conditional frequent pattern tree recursively nor superset checking was needed through modifying the structure of FP-tree. This algorithm reduced the number of mining through early prediction before mining. The application of a method to get the public intersection sets could obtain a complete result. The experiment shows that the efficiency of NCMFP is two to five times as much as that of the similar algorithms in the case of a relatively small support.

Key words: association rule; data mining; Frequent Pattern Tree (FP-tree); maximum frequent itemsets; superset checking

0 引言

在关联规则挖掘中,频繁项集的挖掘是关键的步骤。根据挖掘结果的不同,关联规则挖掘可分为:完全频繁项集挖掘、频繁闭项集挖掘、频繁表示项集挖掘和最大频繁项集挖掘。根据频繁项集向上闭包性质,最大频繁项集已经包含了所有频繁项集,而且在多数的挖掘应用中,最大频繁项集可以满足要求,因此最大频繁项集挖掘具有更强的实用价值。

目前最大频繁项集挖掘算法主要有基于宽度优先的 MaxMiner[1]、DMFI[2]、DMFIA[3] 算法和基于深度优先的 Mafia[4]、GenMax[5]、SmartMiner[6]、FP-Max[7] 算法。上述算法在每次存储频繁项集时,都需要进行超集检验,实验证明在项数多、数据量大的情况下,上述算法的超集检验用时过多,而且大部分算法还要递归生成条件频繁模式树(Frequent Pattern Tree,FP-tree),这又消耗了大量的时空资源,从而影响了算法的效率。

鉴于目前以基于 FP-tree 的算法效果较好,故综合考虑上述各算法,为提高挖掘最大频繁项集的效率,可以从以下几个方面做工作:1)减少或不生成条件 FP-tree;2)如果要生成条件 FP-tree,则必须采用适当的方法提高其生成速度;3)采用剪枝策略,减少挖掘的次数;4)减少或避免超集检验;5)采用的改进策略不能过于复杂,以防其降低算法的效率。本文从以上五点入手,提出了最大频繁模式的非检验挖掘算法(Non-Check Mining algorithm of maximum Frequent Pattern,NCMFP)。实验证明该算法优于同类算法。

1 相关知识

1.1 DMFIA-1 算法[8]存在的问题

基于以上对算法的五点改进意见考虑,文献[8]中的 DMFIA-1 算法是一种较好的方法。该算法充分利用了最大频繁项集存在于 FP-tree 路径中的特性,直接从路径中搜寻最大频繁项集,避免了超集检验和递归生成条件 FP-tree 的操作。但是通过实验,发现该算法在支持度为 95%,对 connect-4 数据集进行挖掘时,只得到 2 个最大频繁项集,没有得到满足要求的所有最大频繁项目集。为了分析其原因,假设 p_i 为兄弟链表中的任意一节点,I_{pi} 代表从 p_i 节点到根节点的路径上所有节点组成的项集,那么以 p_i 为后缀的最大频繁项集一定在以下 3 种情况中产生:1)I_{pi} 可能是最大频繁项集;2)在 p_i 右

收稿日期:2010 - 01 - 04;**修回日期**:2010 - 03 - 10。 **基金项目**:江苏省自然科学基金资助项目(BK20003017)。

作者简介:惠亮(1983 -),男,江苏邳州人,硕士研究生,主要研究方向:数据库、数据挖掘; 钱雪忠(1967 -),男,江苏无锡人,副教授,硕士,主要研究方向:数据库、数据挖掘、网络安全。

侧的兄弟节点中存在节点 p_j,且 I_{pj} 是 I_{pi} 的子集,则该子集可能是最大频繁项集;3)在兄弟链表中存在节点 I_{pj}、…、I_{pk} 而且 I_{pi}、I_{pj}、…、I_{pk} 互不包含,则 $I_{pi} \cap I_{pj} \cap \cdots \cap I_{pk}$(i、j、…、k 互不相等)可能是最大频繁项集。很明显 DMFIA-1 算法只考虑了前两种情况,忽视了第三种情况。与此类似,MMFI 算法[9]也只考虑了前两种情况。本文汲取以上两种算法中的优点,改进 FP-tree 结构,使挖掘过程中不需要递归产生条件 FP-tree;同时采用预测挖掘的剪枝策略提高算法的效率,利用求两个项集最大公共交集的方式保证算法能够挖掘出所有的最大频繁项目集。

1.2 相关性质和定理

性质 1　如果 X 是最大频繁项目集,则 X 的任何真子集都不是最大频繁项目集。

性质 2　如果 X 是非频繁项目集,则 X 的任何超集都是非频繁项目集。

定理 1　在最大频繁项集挖掘算法中,假设 FP-tree 头表中各项按支持度从高到低排序为 a、b、c、d、e、f、h、k、g,当前挖掘 h 项,在 h 项挖掘出的结果中,有包含 a、b、c、d、e 的最大频繁项集,则在完成对项 f 挖掘后,算法可以结束。从性质 1 容易推断出此定理。

定义 1　p_i 是 FP-tree 的兄弟链表中任意一节点,I_{pi} 代表从 p_i 节点到根节点的路径上所有节点组成的项集。

定义 2　兄弟节点排序规则:p_i、p_k 是兄弟链表中的两个任意节点,ancester_p 是 p_i 和 p_k 的最近共同祖先节点,且 ancester_p_i、ancester_p_k 是 ancester_p 的子节点,ancester_p_i 是 p_i 的祖先节点或者就是 p_i;ancester_p_k 是 p_k 的祖先节点或者就是 p_k,当 ancester_p_i 在头表中位于 ancester_p_k 位置上方时,p_i 排在 p_k 的左侧。

定理 2　如果在建立的频繁模式树中,兄弟节点之间按照定义 2 排序,则任意一条兄弟链表中的两个节点 p_i 和 p_k,若 p_i 位于 p_k 左侧,I_{pk} 可能是 I_{pi} 的子集,但是不可能是 I_{pi} 的超集。

证明　假设 I_{pi} 和 I_{pk} 之间存在包含关系,根据 FP-tree 中的兄弟节点排序规则,因为 p_i 位于兄弟节点 p_k 左侧,则在头表中 ancester_p_i 位于 ancester_p_k 的上端,即 support(ancester_p_i) > support(ancester_p_k),根据 FP-tree 的性质,则由 p_k 到 ancester_p_k 所有节点组成的项集的长度小于由 p_i 到 ancester_p_i 所有节点组成的项集的长度,由于两者存在包含关系,故 I_{pk} 是 I_{pi} 的真子集。

2 NCMFP 算法

2.1 改进的 FP-tree

在构建 FP-tree 的过程中,如果兄弟节点按照定义 2 中的规则排序,就需要比较 ancester_p_i 和 ancester_p_k 在头表中的位置。以往的 FP-tree 节点中存储的是项名,因此在比较之前需要在头表中搜索到各自的位置,然后进行比较。由于每次插入一个新的兄弟节点都需要作上述操作,当数据集庞大时,这种查找过程要花费很多时间。如果树的节点中存储项在头表中的位置编号,则在排序时直接比较。因此本文用数字频繁模式树 Digital FP-tree(DFP-tree)代替常规的 FP-tree,这也是本文的预测剪枝策略的需要。

DFP-tree 的生成步骤如下:

1)扫描事务集,得到所有满足要求的频繁项,并按照支持度从高到低排序、建立头表。

2)对于事务集中的任意一条事务 t_i,删除 t_i 中的非频繁项,剩下的项用各自在头表中的位置编号代替,并按从小到大排序,存储为[item_position | Item_Position]。其中 item_position 代表排序后第一个项的位置编号,Item_Position 代表排序后的剩余项位置编号。调用 insert_tree([item_position | Item_Position],T),把排序后的数字串插入到 DFP-tree 中,并且保证树中各兄弟节点按照定义 2 中的规则排序。

2.2 预测剪枝策略

根据定理 1,假设当前的 DFP-tree 头表中的项从底向上分别为:7,6,5,4,3,2,1,0,采用自底向上的策略对以 7 为后缀的项集挖掘时,如果得到结果:{7,6,5,4,3,2,1,0:min-support},则对于后续的 6 到 0 各项都无需挖掘,这就减少了挖掘的次数。此策略可以通过算法 1 实现。

算法 1　search_result(search_head,finally_head)

输入:存储结果的链表头节点指针 finally_head,存储当前待挖掘项集的链表头指针 search_head;

输出:-1 或者其他数值。

```
for(当前所有最大频繁项集)
{search_result 指向一个最大频繁项集链表表头;
 result_move 指向待挖掘的链表表头,即 result_move = search_
 head;
 while(指针 search_result、result_move 均为非空)
 //两个链表中的项逐个匹配
 {If(search_result -> item_position 与 result_move ->
     item_position 相等)
    两个指针各自向后移动一个位置;
    Else if(search_result -> item_position > result_move ->
          item_position)
    search_result 指针后以一个位置,result_move 不变
    Else if(search_result -> item_position < result_move ->
          item_position)
    两个链表不匹配,跳出此循环}
 If(两个链表匹配) min 为 result_move -> item_position 中最小的
 值,检验当前 search_result 指向的链表中是否顺序存在 min-1、
 min-2、min-3 直到 0 的项集
 If(min-1、min-2、min-3 直到 0 的项集存在)返回-1;
 //表明挖掘可以结束
 Else 取下一个最大频繁项集的链表表头;          //继续循环
}
```

2.3 NCMFP 算法思想

算法在纵向上采用自底向上的策略,横向的兄弟节点链表采用从左到右的方式。指针 p 从兄弟链表头开始,如果 support(p)≥min-support,则 I_p 是最大频繁项集。存储 I_p,并在 p 右侧搜索其子集且标记为不可挖掘(在树的节点中添加 YorN_sign 域,YorN_sign 为 1 表示可以挖掘,为 0 表示不可挖掘,初始值全为 1),然后指针移向下一个节点。如果 support(p) < min-support,则分为以下两种情况:

1)在 p 节点的右侧存在节点 p_j,且 I_{pj} 是 I_p 的子集,该子集 I_{pj} 可能是最大频繁项集。此时需要把 I_p 的支持度信息转移到其子集中(在树的节点中添加 move_count 域,初始值为 0)。

2)兄弟链表中存在 p_i,且 I_{pi} 不是 I_p 的子集,令 $I_{pj} = I_{pi} \cap I_p$,如果 I_{pj} 非空,则 I_{pj} -> move_count = I_p -> count,并把 I_{pj} 按

照定义2中的规则添加到兄弟链表中，则 I_{pj} 可能是最大频繁项集。

2.4 NCMFP 算法流程

输入：数字频繁模式树 DFP-tree 和最小支持度 min-support；

输出：最大频繁项集 MFS。

```
For( toubiao_max_position; toubiao_max_position  > =0; toubiao_max
_position - - )
{ end_sign = search_result( search_head, finally_head) ;
                                        //检验是否可以结束挖掘
 If( end_sign = -1)  Break;                    //结束挖掘
 Else{
   sibling_p = toubiao[ toubiao_max_position] . node_link;
   //取得当前最底层的兄弟链表头
   While( sibling_p! = NULL) {
     If ( sibling_p -> YorN_sign == 1) {
       //Ip 可能是最大频繁项集
       If ( ( sibling_p -> count + sibling_p -> move_count) > =
         min-support) {
         //Ip 是最大频繁项集
         I sibling_p -> parent -> YorN_sign = 0;
                                        //标记其父节点不可挖掘
         将 Isibling_p 存入 MFS;
         While( 在 sibling_p 节点右侧沿兄弟链表搜索 YorN_
         sign = 1 的所有节点 pi) {
         If( Ipi 是 Isibling_p 的子集) pi -> YorN_sign = 0;
                                        //标记其子集为不可挖掘
         }
         Isibling_p = Isibling_p -> node_link;
         //处理下一个节点
       }
       Else{
         //Ip 不是最大频繁项集，其子集可能是最大频繁项集
         While( 在 sibling_p 节点右侧沿兄弟链表搜索 YorN_
             sign = 1 的所有节点 pi) {
             If ( Ipi 是 Isibling_p 的子集)
               pi -> move_count = sibling_p -> count + pi ->
               move_count;
             Else{ Ipj = Ipi ∩ Isibling_p; Ipj -> move _ count =
               Isibling_p -> count;
             }
         }
         Isibling_p = Isibling_p -> node_link;
         //处理下一个节点
       }
     }
     Else{ //Ip 和它的子集不是最大频繁项集
       While( 在 sibling_p 节点右侧沿兄弟链表搜索
             YorN_sign = 1 的所有节点 pi) {
         If( Ipi 是 Isibling_p 的子集) pi -> YorN_sign = 0;
         Isibling_p -> parent -> YorN_sign = 0;
                                        //标记其父节点不可挖掘
         Isibling_p = Isibling_p -> node_link;    //处理下一个节点
     }
   }
 }
}
```

2.5 NCMFP 算法实例

表1表示的是事务数据集，图1是对应的 DFP-tree，图中左侧第一列为头表中位置编号，第二列为对应的频繁项名。设 min-support = 2，首先 toubiao_max_position = 9，由于项集{9,4,2:1}和项集{9,7,4,3:1}互不相容，故产生新的项集{9,4}，按照顺序将其插入到合适位置，且其支持度为2，由此得到以9为后缀的最大频繁项集的位置编号集合{9,4:2}。当 toubiao_max_position = 8 时：

1）由于三个项集互不包含，于是由{8,5,4,1,0:count = 1}和{8,7,5,1,0:count = 1}产生新的集合{8,5,1,0:move_count = 1}；由{8,5,4,1,0:count = 1}和{8,5,3,0:count = 1}产生新的集合{8,5,0:move_ count = 1}，依次排序为{8,5,4,1,0:count = 1}、{8,7,5,1,0:count = 1}、{8,5,1,0:move_count = 1}和{8,5,0:move_count = 1}，指针右移。

2）当前指针指向{8,7,5,1,0:count = 1}，因为{8,5,1,0:move_count = 1}是其子集，所以支持度要传递过来，即其 move_count = 1 + 1，由于{8,7,5,1,0:count = 1}与{8,5,0:move_count = 1}相交得到集合已经存在，故无需存储，只转移支持度信息。按此顺序进一步挖掘得到以8为后缀的最大频繁项集的位置编号集合为{8,5,1,0:move_count = 2}。按照以上的方式最终会得到所有的最大频繁项集位置编号集合，对照头表即可得到所有的最大频繁项集。而按文献[2]的 DMFIA-1 算法，挖掘结果会缺失最大频繁项集：{8,2:2}，{7,6,1,5:2}，{11,6,5:2}，{11,1:2}，{11,3:2}，{11,2:2}，{19,3,1:2}。

表1 事务数据库

事务编号	单个事务包含的项	排序后的事务集	在头表中对应的位置
1	1,2,5,6,7	5,1,2,6,7	0,1,4,5,8
2	2,4,8,11	4,2,11,8	3,4,7,9
3	2,5	5,2	0,4
4	1,2,3	1,3,2	1,2,4
5	3,4,5,19	5,3,4,19	0,2,3,6
6	1,4,6	1,4,6	1,3,5
7	1,2,3,4,5	5,1,3,4,2	0,1,2,3,4
8	1,3,4	1,3,4	1,2,3
9	3,5,6,11	5,3,6,11	0,2,5,7
10	2,3,5,4	5,3,4,2	0,2,3,4
11	1,2,3,11	1,3,2,11	1,2,4,7
12	3,4,5,6	5,3,4,6	0,2,3,5
13	4,5,6,7	5,4,6,7	0,3,5,8
14	1,5,6,7,11	5,1,6,11,7	0,1,5,7,8
15	1,3,6,19	1,3,6,19	1,2,5,6
16	1,2,5	5,1,2	0,1,4
17	1,3,4,5,19	5,1,3,4,19	0,1,2,3,6
18	1,4,5,19	5,1,4,19	0,1,3,6
19	2,3,5,8	5,3,2,8	0,2,4,9
20	1,6,19	1,6,19	1,5,6

3 实验分析

为检测该算法的性能，在内存为480 MB，CPU 为 Pentium Ⅳ 2.93 GHz，操作系统为 Windows XP 的机器上用VC++6.0实现了 FP-Max、NCMFP 算法，FP-Growth 算法由 B. Goethals 提供。图2中采用的是 connect-4 数据库，该数据有67 558个事务，平均事务长度为43，包含130个项，图中显示了在不同的最小支持度情况下（分为6档：65%、70%、75%、80%、85%、90%）挖掘消耗的时间。在支持度小于70%时，NCMFP 算法用时大约为 FP-Max 算法的40%，且耗时随着支持度的减小，

增长的趋势平缓,这是由于支持度越低,最大频繁项集的长度就越长,采用的预测剪枝策略就越有效,挖掘的次数也就越少。

图 3 中采用的是随机生成的稀疏数据集,该数据集共 10000条事务,事务平均长度为 20,包括 15 个属性,图中显示了不同的支持度(分为 5 档:1%、3%、5%、10%、20%)下的挖掘时间,性能为 NCMFP > FP-Max ≥FP-Growth,本文算法的效率是 FP-Max 效率的 2 倍以上。

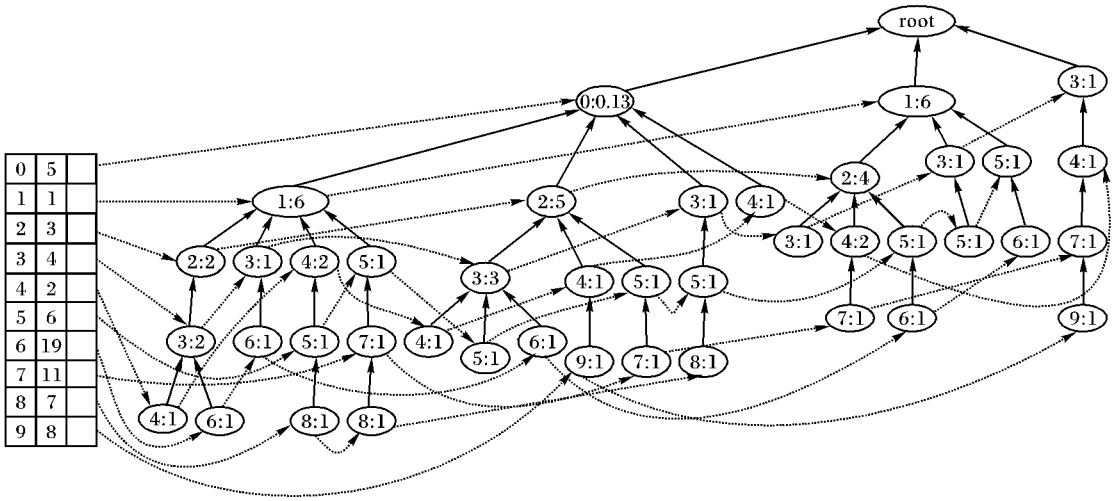

图 1　头表,数字频繁模式树 DFP-tree

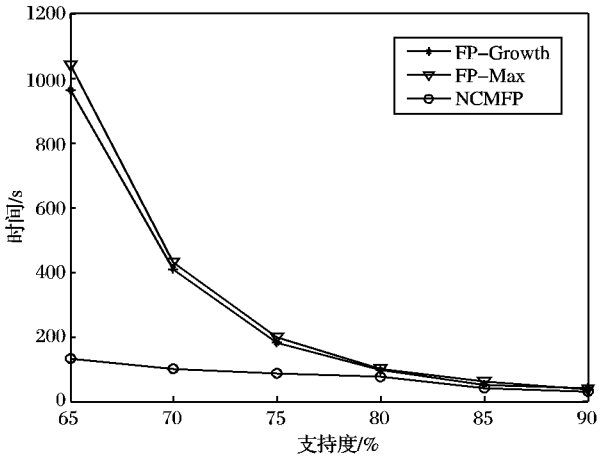

图 2　connect-4 数据集挖掘对比

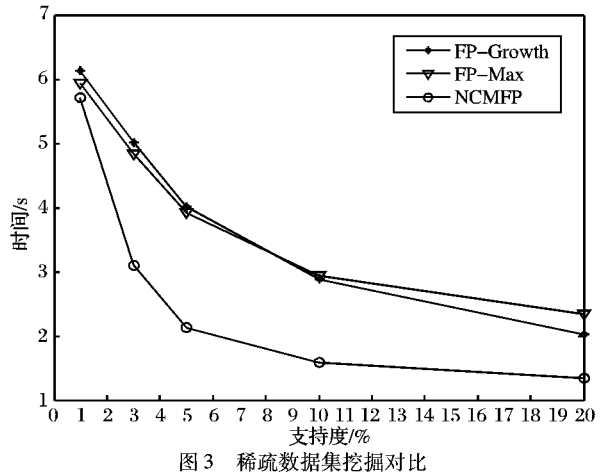

图 3　稀疏数据集挖掘对比

4　结语

本文提出的 NCMFP 算法,采用了交集的方法解决了 DMFIA-1 算法无法发现所有最大频繁项集的问题;改传统的 FP-tree 结构为 DFP-tree 有序结构,使整个挖掘过程无需生成条件模式树,也不需要超集检验;采用的预测剪枝策略减少了挖掘的次数,从而提高了算法的效率。实验证明,与同类算法相比,该算法有明显的优越性。从 NCMFP 算法流程可以看出,该算法的主要工作是遍历兄弟节点的前缀路径,这消耗了算法的绝大部分时空资源。因此,如何更有效地遍历节点的前缀路径是一个值得进一步探讨的问题。

参考文献:

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

[2] 路松峰, 卢正鼎. 快速开采最大频繁项目集[J]. 软件学报, 2001, 12(2): 293 - 297.

[3] 宋余庆, 朱玉全, 孙志挥, 等. 基于 FP-tree 的最大频繁项目集挖掘及更新算法[J]. 软件学报 2003, 14(9): 1586 - 1592.

[4] BURDICK D, CALIMLIM M, GEHRKE J. MAFIA: A maximal frequent itemsets algorithm for transactional databases[C]// Proceedings of the 17th International Conference on Data Engineering. Washington, DC: IEEE Computer Society, 2001: 443 - 452.

[5] GOUDA K, ZAKI M J. Efficiently mining maximal frequent itemsets [C]// Proceedings of the IEEE International Conference on Data Mining. Washington, DC: IEEE Computer Society, 2001: 163 - 170.

[6] ZHOU Q H, WESLEY C, LU B J. SmartMiner: A depth 1st algorithm guided by tail information for mining maximal frequent itemsets [C]// Proceedings of the IEEE International Conference on Data Mining. Washington, DC: IEEE Computer Society, 2002: 570 - 577.

[7] GRAHNE G, ZHU J F. High performance mining of maximal frequent itemsets[C]// Proceedings of the 6th SIAM International Workshop on High Performance. New York: HPDM Press, 2003: 135 - 143.

[8] 刘乃丽, 李玉忱, 马磊. 一种基于 FP-tree 的最大频繁项目集挖掘算法[J]. 计算机应用, 2005, 25(5): 998 - 1000.

[9] 陈晨, 鞠时光. 基于改进 FP-Tree 的最大频繁项集挖掘算法[J]. 计算机工程与设计, 2008, 29(24): 6236 - 6239.

[10] 王现君. 在单向 FP-tree 上挖掘频繁闭项集[J]. 计算机工程与应用, 2008, 44(10): 150 - 153.

[11] 马丽生. 快速挖掘频繁项目集的算法[J]. 计算机工程与设计, 2009, 30(8): 1903 - 1906.