文章编号：1674-8085(2018)04-0043-05

基于改进频繁模式树的最大频繁项目集更新挖掘算法

*赵群礼，郭玉堂，史君华

（合肥师范学院计算机学院，安徽，合肥 230061）

摘　要：在挖掘最大频繁项目集的过程中，通过改变最小支持度阈值可以挖掘更有用的最大频繁项目集，为此提出了一种最大频繁项目集更新挖掘算法 UAMMFI(Updating Algorithm for Mining Maximal Frequent Itemsets)。算法基于改进后的频繁模式树结构，在更新挖掘过程中，不需产生候选项目集和条件模式树，并且充分利用先前已挖掘的最大频繁项目集中包含的信息，快速更新挖掘出最小支持度阈值变化后的最大频繁项目集。实验结果表明，算法能够高效更新挖掘最大频繁项目集。

关键词：关联规则；最大频繁项目集；改进频繁模式树；最小支持度阈值

中图分类号：TP311　　**文献标识码**：A　　DOI:10.3969/j.issn.1674-8085.2018.04.008

AN UPDATING ALGORITHM FOR MINING MAXIMAL FREQUENT ITEM SETS BASED ON IMPROVED FREQUENT PATTERN TREE

*ZHAO Qun-Li, GUO Yu-Tang, SHI Jun-Hua

(The Department of Computer Science and Technology, Hefei Normal University, Hefei,Anhui 230061, China)

Abstract: In mining the maximal frequent item sets, we can get more useful maximal frequent item sets by changing the minimum support threshold. Furthermore, we proposed a new algorithm UAMMFI(updating algorithm for mining maximal frequent item sets) for mining maximal frequent item sets for the demand. The algorithm didn't generate candidate item sets and construct conditional pattern tree based on improved frequent tree. We also make full use of the information contained in maximal frequent item sets previously mined in the process of updating mining maximal frequent item sets. Finally, we can efficient mining new maximal frequent item sets after the minimum support threshold change. Based on experiment results, we can see that the algorithm has excellent performance.

Key words: association rules; maximal frequent item sets; improved frequent pattern tree; minimum support threshold

0　引言

在关联规则挖掘的概念提出以后，业界对关联规则挖掘算法进行了大量的研究，其中挖掘最大频繁项目集的算法是一个重要的研究方向。由于最大频繁项目集的子集都是频繁项目集，将挖掘频繁项目集的任务转化为挖掘最大频繁项目集，可以减少挖掘过程中的计算量，提高挖掘的

收稿日期：2018-03-11；修改日期：2018-05-17
基金项目：国家自然科学基金(61503116)；安徽省高校自然科学研究重点项目(KJ2016A585)
作者简介：*赵群礼(1974-)，男，安徽霍邱人，讲师，硕士，主要从事模式识别和数据挖掘研究(E-mail:zql@hfnu.edu.cn);
郭玉堂(1962-)，男，安徽安庆人，教授，博士，主要从事图像识别和数据挖掘研究(E-mail:guoyutang@hfnu.edu.cn);
史君华(1981-)，女，安徽合肥人，副教授，硕士，主要从事模式识别与数据挖掘方面的研究(E-mail:junhuash@163.com).

效率。业界已提出的较为著名的最大频繁项目集挖掘算法有 GenMax[1]、SmartMiner[2]、DMFIA[3]、FpMax[4]、Mafia[5]等。在这些算法提出以后，研究人员又在上述算法的基础上进行了改进，提出了一些新的算法，如：Hybrid GeneticMax[6]、FPMax-reduce[7]、IFPmax[8]、Max-IFP[9]、MFPAS[10]等。这些算法在固定的最小支持度阈值（表示为 *minsup*）和数据库（表示为 TDB）中能有效挖掘最大频繁项目集。但在实际应用中，为了发现更有用的最大频繁项目集，用户要频繁地调整 *minsup*，而 TDB 也会随着时间发生变化，对于 *minsup* 和 TDB 变化以后，如何高效更新挖掘最大频繁项目集是一值得研究的课题。一种简单的解决方案是在 *minsup* 和 TDB 变化以后，调用最大频繁项目集挖掘算法再重新计算一遍，找出变化后的最大频繁项目集，但这种方法的不足之处是先前的挖掘结果和计算量都被浪费，而且对于海量数据来说，重新挖掘需要花费大量的计算，因此需要研究专门的算法用于最大频繁项目集的更新挖掘。在文献[3]中首先提出了一种在 TDB 变化时的更新挖掘最大频繁项目集算法 UMFIA，此后的很多研究也是针对 TDB 变化时的更新挖掘，而针对 *minsup* 变化后更新挖掘最大频繁项目集的研究很少。为此，本文提出了一种基于改进频繁模式树的最大频繁项目集更新挖掘算法 UAMMFI(Updating Algorithm for Mining Maximal Frequent Itemsets)，能够在 *minsup* 变化以后高效更新挖掘最大频繁项目集。

1 相关问题

1.1 相关定义

给定的事务数据库 TDB 和包含 n 个项目的项目集 $I=\{i_1,i_2,i_3,\cdots,i_n\}$，某个项目集 t（$t\subseteq I$）在 TDB 中的支持度 $s(t)$定义为：

$$s(t)=\frac{N_t}{|TDB|}\times 100\%$$

其中 N_t 表示 TDB 中包含 t 的记录数，|TDB| 表示数据库中的记录总数。t 在 TDB 中的支持数 $c(t)$与支持度 $s(t)$的关系为：$c(t)=s(t)\times|TDB|$。

定义 1：若项目集 t 的支持度 $s(t)\geq minsup$ ，则称 t 为频繁项目集。

定义 2：对于项目集 t，若 $s(t)\geq minsup$，而任意项目集 $u\supset t$，都有 $s(u)< minsup$，则称 t 为最大频繁项目集，表示为 MFI(Maximum Frequent Itemsets)，所有最大频繁项目集的集合表示为 MFS(Maximum Frequent Sets)。

1.2 频繁模式树

频繁模式树结构是由 Han 等人在文献[11]中提出的，基于频繁模式树结构挖掘 TDB 中的频繁项目集，在挖掘过程中不用产生候选项目集。在频繁模式树结构中，每个树节点包括四个数据项：node-name(存储节点的项目名)、node-count(存储到达此节点的子路径所对应的事务数)、node-link(指向树中同名节点)和 node-parent(指向父节点)。为便于遍历，建立一个头表 head-table，其中包含三个数据项：item-name、item-count 和 head-link，item-name 表示频繁项目名，item-count 表示项目的支持数，head-link 是指向同名节点链的头指针。建立频繁模式树的算法在文献[11]中有详细描述，本文不再赘述。图 1 是根据表 1 中的数据库在最小支持数等于 3 时构建的频繁模式树。

表 1 事务数据库 TDB

Table 1 A transaction database

序号	记录
1	a c e f x y
2	a b c f x z
3	b f j z
4	b c s y
5	a c d f i x
6	f k m
7	a c f l
8	a b c k i
9	f l s

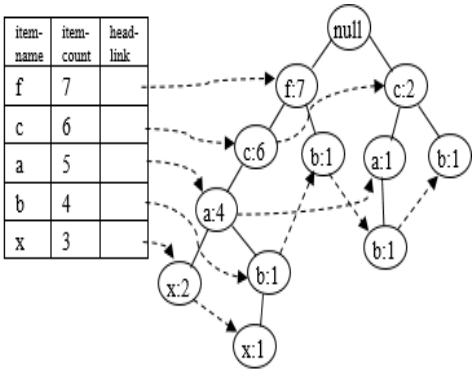

图 1 根据表 1 数据库建立的频繁模式树

Fig.1 A frequent pattern tree based on table 1 database

1.3 改进频繁模式树

基于频繁模式树结构虽然可以有效挖掘频繁项目集，但该结构不适合直接用于更新挖掘最大频繁项目集，需要对其进行改进。根据频繁模式树的构造过程，当一个节点的计数大于等于最小支持数时，以该节点为后缀的路径对应的项目集必然是频繁项目集，也有可能是最大频繁项目集，如果能知道某个频繁项目集不是最大频繁项目集，则其子集都可以不用再遍历，这样可以有效减少树中遍历的节点数。为此，在每个节点中增加一个 *flag* 域，用于表示树中以某节点为后缀的路径对应的项集是否是最大频繁项目集，0 表示不是最大频繁项目集，1 表示可能是最大频繁项目集。由于每个节点路径对应的项目集都有可能是最大频繁项目集，所以每个节点的 *flag* 域都初始化为 1。另外，在头表中增加一个 *visited* 域，用于标记对应项目是否已遍历，1 表示已遍历，0 表示没有遍历，并根据文献[12]提出的方法，减少挖掘最大频繁项目集过程中对超集的检测。在构造频繁模式树时，树中同层的节点按照头表中的顺序从左至右排列。图 2 给出了根据表 1 建立的改进后的频繁模式树，最小支持数为 3。在后面各部分所提到的频繁模式树，都指的是改进后的频繁模式树。

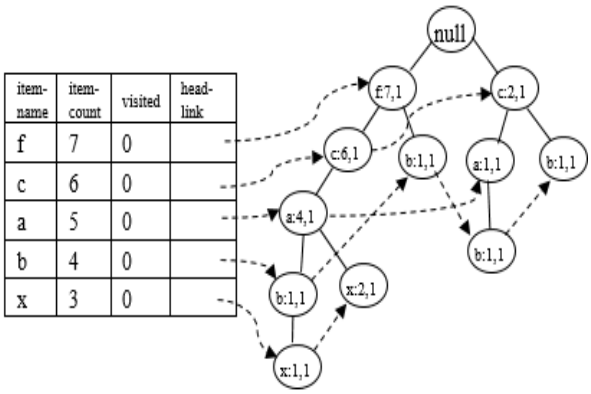

图 2 改进后的频繁模式树

Fig.2 An improved frequent pattern tree

根据改进后的频繁模式树，可以得出以下性质：

性质 1：设 m_i 和 m_j 是 head-table 中的两个频繁项目，且 m_j 在 m_i 的下方，则以 m_j 为后缀的项目集 I_{mj} 一定不会是以 m_i 后缀的项目集 I_{mi} 的子集，并且可能存在 $I_{mi} \subseteq I_{mj}$。

证明：根据改进后的频繁模式树的结构，频繁项目在 head-table 中是按照支持数降序排列的，若 m_i 和 m_j 出现在同一条路径中，则 m_j 必然在 m_i 的后缀路径中，性质 1 成立。

性质 2：在改进频繁模式树中的任意一个节点链上的两个节点 x_1 和 x_2，若 x_1 在 x_2 的左侧，且 x_1 和 x_2 有共同的祖先节点（null 根节点除外），则以 x_1 为后缀的项目集 I_{x1} 一定不是以 x_2 为后缀的项目集 I_{x2} 的子集。

证明：根据树的构造过程可知，项目集 I_{x1} 和 I_{x2} 不完全相同，而且 x_1 所在的层数一定不小于 x_2 所在的层数，因此 $|I_{x1}| \geq |I_{x2}|$，即 I_{x1} 一定不是 I_{x2} 的子集，性质 2 成立。

2 最大频繁项目集更新挖掘算法 UAMMFI

设 TDB 保持不变，只有最小支持度发生变化。设变化前的最小支持度表示为 *minsup*，变化后的最小支持度表示为 *minsup′*，对应的最大频繁项目集的集合分别表示为 MFS 和 MFS′，频繁项目集合表示为 *I* 和 *I′*。

性质 3：设事务数据库 TDB 不变，若 $minsup' < minsup$，则 $I \subseteq I'$；若 $minsup' > minsup$，则 $I \supseteq I'$。

证明：对于任意项目 $m \in I$，m 的支持度 $sup(m) \geq minsup$。如果 $minsup' < minsup$，则 $sup(m) > minsup'$，即 $m \in I'$，故有 $I \subseteq I'$成立；同理可证如果 $minsup' > minsup$，则 $I \supseteq I'$，性质 3 成立。

性质 4：设 TDB 不变，当 $minsup' < minsup$ 时，对 MFS 中任意一个最大频繁项目集 $mfi \in MFS$，则 $\exists\, mfi' \in MFS'$，使得 $mfi \subseteq mfi'$；同理，当 $minsup' > minsup$ 时，对于 $\forall\, mfi' \in MFS'$，则 $\exists\, mfi \in MFS$，使得 $mfi \supseteq mfi'$。（证略）

性质 5：设 *T* 和 *T′*分别是对应于 *minsup* 和 *minsup′* 的改进后的频繁模式树，当 $minsup' < minsup$ 时，有 $I''=I'-I$，则 *I″*中的项目一定为 *T′* 的叶节点或叶节点的前缀节点，且在 *I* 中的项目之后；当 $minsup' > minsup$ 时，有 $I''=I-I'$，则 *I″*中项目在 *T′*中无对应的节点。

证明：当 $minsup' < minsup$ 时，I''中项目的支持度小于 I 中所有项目的支持度，根据频繁模式树的构造过程可知，I''中的项目必然排在 I 中的项目之后，在 T'中作为 I 中项目的后缀，所以 I''中的项目或者为叶子节点，或者为 I''中其他项目的前缀；当 $minsup' > minsup$ 时，因 I''中的项目均为 $minsup'$下的非频繁项目，所以在 T'中没有 I''中项目对应的节点，性质 5 成立。

性质 6：对于频繁模式树中某节点 N，若存在 N. node-count ≥ minsup×|TDB|，则以 N 为后的路径$\{p_1,p_2,\cdots,p_n\}$对应的项目集 $I_p=\{I_{p1},I_{p2},\cdots,I_{pn}\}$必为频繁项目集。

证明：根据频繁模式树的构造过程可知，树中的节点是按照支持数降序排列的，如果节点的计数大于等于最小支持数，则其前缀中的所有节点的计数也必然大于等于最小支持数，因此以该节点为后缀的路径对应的项目集必为频繁项目集，性质 6 成立。

由于最小支持度发生变化分为 $minsup' > minsup$ 和 $minsup' < minsup$ 两种情况，在这两种情况中，都需要对原有的频繁模式树进行调整，以用于挖掘 MFS′，下面分别进行阐述：

（1）当 $minsup' > minsup$ 时，由于频繁模式树中已经包含所需信息，因此不需再扫描数据库，但要对频繁模式树中的项目和节点信息进行标定。根据性质 3 和 5，$minsup$ 下的频繁项目在 $minsup'$可能会成为非频繁项目，因此需要对原频繁模式树头表中不满足 $minsup'$的项目加上已处理标记，即将头表中不满足 $minsup'$的项目的 visited 标记都设为 1，相当于将这些项目从头表删除。再根据性质 4，MFS′中的最大频繁项目集一定是 MFS 中的子集，若 $mfi \in$ MFS 中，且 $sup(mfi) \geq minsup'$，则 mfi 必为 $minsup'$下的最大频繁项目集，可以直接并入 MFS′中，即 $MFS'=MFS' \cup mfi$，然后遍历项目 $mfi[k]$(k 等于 mfi 的长度，表示为$|mfi|$)的节点链，若节点链中某个以 $mfi[k]$为后缀的路径是 mfi 的子集，则将该路径中所有节点的 *flag* 标记都设为 0，若所有以 $mfi[k]$为后缀的路径都是 mfi 的子集，除了将所有路径中节点的 *flag* 标记设为 0，还要将头表中项目 $mfi[k]$的 visited 标记设为 1。

（2）当 $minsup' < minsup$ 时，根据性质 4，MFS 中的项目集一定是 MFS'中项目集的子集，根据性质 3，在 $minsup$ 下的非频繁项目可能变成 $minsup'$ 下的频繁项目，因此需要扫描数据库 TDB 将新增的 $minsup'$ 下的频繁项目找出来，并调整原频繁模式树，方法如下：（1）扫描一次 TDB，找出新增的频繁项目 I''，并按照降序插入到原有的 head-table 中，并将 *visited* 置为 0；（2）再次扫描 TDB，将每个记录中新增的频繁项目按照头表中的顺序从左至右依次插入到频繁模式树中，根据性质 5，新增的频繁项目可能是树中的叶子节点，也可能是其他新增项目的前缀节点。

在频繁模式树调整好之后就可在其中挖掘 $minsup'$ 下的 MFS′，整个挖掘过程按照自下而上从左至右的顺序进行。根据性质 1 和 2，某项目 p_i 的节点链中，若左侧节点对应的项集 I_{Lpi} 是最大频繁项目集，右侧节点对应的项集 I_{Rpj} 是 I_{Lpi} 的子集，则 I_{Rpj} 对应路径中的所有节点的 *flag* 都设为 0，由此可以减少超集检测。若 I_{Lpi} 不是频繁项目集，且 $I_{Rpj} \subset I_{Lpi}$，则 I_{Rpj} 可能会是最大频繁项目集，需将 p_i 的节点计数累加到节点 p_j，即 $p_j.node\text{-}count= p_j.node\text{-}count+ p_i.node\text{-}count$。根据性质 6，若 I_{pi} 是频繁项目集但不是最大频繁项目集，则的子集都不是最大频繁项目集，所以在 p_i 右侧的节点链中如果存在的 I_{pi} 子集 I_{pj}，则应将 I_{pj} 对应节点的 *flag* 都设为 0。

通过以上分析，给出 UAMMFI 算法描述如下：

输入：$minsup'$、$minsup$、MFS、改进的频繁模式树 T。

输出：MFS′。

（1）MFS′={ }；

（2）*if*(minsup′ >*minsup*){

（3）将 T 的头表中不满足 $minsup'$的项目的 visited 设为 1;

（4）for each $mfi \in MFS$ *do* {

（5）*if*(sup(mfi)≥$minsup'$) {

（6）MFS′=MFS′∪mfi;

（7）搜索项目 $mfi[k]$的节点链中的节点 pi，

若 *Ipi* 是 *mfi* 的子集，则将 *Ipi* 对节点的 *flag* 都设为 0，如果节点链中所有节点的 *flag* 都为 0，则将项目 *mfi*[*k*]的 *visited* 标记设为 1; }

（8） }

（9） 调用 UAMMFI(T);

（10）}

（11）*else if*(*minsup'*<*minsup*) {

（12）扫描 TDB，找出满足 *minsup'*的新增项目，并按支持度降序放入 *T* 的 head-table 中;

（13）再次扫描 TDB，将每个记录中新增项目对应的路径添加到 *T* 中;

（14）调用 UAMMFI(T);

（15）}

Procedure UAMMFI(Tree){

（1）*for* (*i*=*k*;*i*>0;*i*--) *do* { //k 等于 Tree 中 head-table 的长度

（2）*if* (head-table[i].visited≠1) then { //遍历树中未访问项目的节点链

（3）*head-table*[*i*].*visited*=1;

（4）*p* = *head-table*[*i*].*head-link*;

（5）*while*(p≠*null*) do {

（6）*if*(p.*flag* = 0) { p = *p.node-link*; }

（7）*else if*(*p.node-count*≥*minsup'*) { //*Ip* 是最大频繁项目集

（8）搜索节点链中 *p* 右侧的其他节点 *pi*，如果 *pi.flag* 等于 1，并且以 *pi* 为后缀的项集 *Ipi* 是以 *p* 为后缀的项集 *Ip* 的子集，则将 *pi.flag* 设为 0；

（9）p. node-parent.flag = 0;

（10）MFS′=MFS′∪Ip;

（11）p = p.node-link; }

（12）else{ //*Ip* 不是频繁项目集

（13）搜索节点链中 *p* 右侧的其他节点 *pi*，如果 *pi.flag* 等于 1，并且以 *pi* 为后缀的项集 I_{pi} 是以 *p* 为后缀的项集 I_p 的子集，则 *pi.count*= *pi.count*+ *p.count*;}

（14）}//end while

（15）}//end if

（16）}//end for

（17）}//end Procedure

3 算法实验对比和性能分析

3.1 实验对比

在实验中，用 C++语言在酷睿 i7、2GHZ 处理器、8GB 内存、Windows 10 系统的计算机上实现了 UAMMFI 算法和 FPMax 算法，并采用了与文献[7]相同的 mushroom 测试数据库。实验方案为：我们先设定最小支持度为 0.3，并用 Fpmax 算法挖掘出此时的 MFS，然后用 UAMMFI 算法在最小支持度依次变化为 0.2、0.1、0.05、0.02、0.01 时分别进行更新挖掘，得到对应的 MFS′，最后最小支持度再按照 0.02、0.05、0.1、0.2、0.3 依次变化，再用 UAMMFI 算法依次进行更新挖掘。为了对比，我们用 Fpmax 算法分别在 0.2、0.1、0.05、0.02、0.01 时重新挖掘一遍，表 2 和表 3 给出了实验结果数据，图 3 和图 4 给出了实验结果对比图。从实验结果可以看出，UAMMFI 算法在进行最大频繁项目集更新挖掘时的效率要好于 Fpmax 算法，而且算法的性能稳定，在不同的支持度阈值下，算法执行时间的变化幅度不大，也说明算法具有较好的执行性能。

表 2 最小支持度由大变小的实验数据

Table 2 Experimental data with minimal support from large to small

Minsup	FpMaxa(s)	UAMMFI(s)
20%	1.79	0.647
10%	2.65	0.816
5%	3.86	1.035
2%	5.12	1.55
1%	6.45	2.203

表 3 最小支持度由小变大的实验数据

Table 3 Experimental data with minimal support from small to large

Minsup	FpMaxa(s)	UAMMFI(s)
2%	5.12	1.25
5%	3.86	0.935
10%	2.65	0.616
20%	1.79	0.447
30%	1.1	0.31

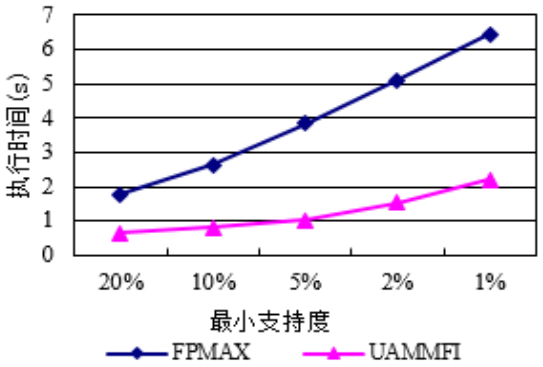

图 3 最小支持度阈值由大变小

Fig.3 Minimum support threshold from large to small

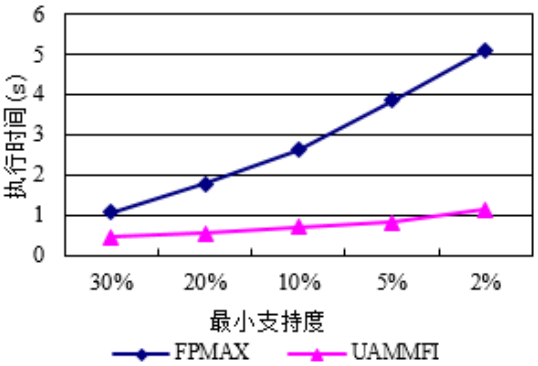

图 4 最小支持度阈值由小变大

Fig.4 Minimum support threshold from small to large

3.2 性能分析

UAMMFI 算法充分利用改进后的频繁模式树的特点和先前已挖掘的 *MFS* 中包含的信息，在挖掘过程中不产生候选项目集和条件模式树。算法所需的存储空间仅为存储频繁模式树的容量，计算过程中不需要占用额外的存储空间，算法所需的计算时间主要消耗在自底向上自左至右搜索最大频繁项目集上，根据改进后的频繁模式树，在最好情况下，算法的性能可以达到 $O(n)$（n 为头表的长度），在最坏情况下算法性能要达到 $O(nm)$（n 为头表的长度，m 为最长节点链的长度），而且由于在树中的每个节点中设置有 *flag* 标记，对于不可能成为最大频繁项目集的节点路径，不需再进行搜索，因此有效减少了搜索过程中的计算量。

另外，当 $minsup' < minsup$ 时，虽然需要扫描数据库，将满足 $minsup'$ 的新项目添加到改进后的频繁模式树中，由于不需要重新建立频繁模式树，仅需将新项目按顺序插入到树中现有路径的后面，所以所需的计算量要比新建频繁模式树所需的计算量小很多。当 $minsup' > minsup$ 时，由于 MFS' 中项集是 *MFS* 中项集的子集，因此不需要再扫描数据库，仅需在原有频繁模式树中即可挖掘出满足 $minsup'$ 的最大频繁项目集 MFS'，由于树中不满足 $minsup'$ 的节点不会再被遍历，所以挖掘 MFS' 所花费的计算量要比 $minsup' < minsup$ 时所需的计算量小，实验结果也证明了这一点。

4 结束语

在最小支持度阈值变化时更新挖掘最大频繁项目集，不需要在数据库中重新挖掘一遍，因此可以减少挖掘过程中的计算量。本文提出的最大频繁项目集更新挖掘算法 UAMMFI，基于改进后的频繁模式树结构，挖掘过程中不会递归构建条件模式树，并通过树中节点设置 *flag* 标记，减少了遍历节点的数量，节约了计算时间，从而提高了算法的效率。根据实验对比结果和性能分析可以看出，该算法可以有效解决在最小支持度阈值发生变化时更新挖掘最大频繁项目集的问题。

参考文献：

[1] Gouda K, Zaki M J. Efficiently Mining Maximal Frequent Itemsets[C]. IEEE International Conference on Data Mining. IEEE, 2002:2405-2409.

[2] Zou Q, Chu W W, Lu B. SmartMiner: A Depth First Algorithm Guided by Tail Information for Mining Maximal Frequent Itemsets[C]. Proceeding of the IEEE International Conference on Data Mining, 2002:570-577.

[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] Grahne G, Zhu J F. High performance mining of maximal frequent itemsets[C].Proceeding of the 6th SIAM International Workshop on High Performance Data Mining,2003:135-143.

[5] Burdick D, Calimlim M, Flannick J, et al. MAFIA: a maximal frequent itemset algorithm[J]. IEEE Transactions on Knowledge & Data Engineering, 2005, 17(11):1490-1504. (参考文献[6]- [12]转第 64 页)

5 结论

采用有限元方法研究了某汽车白车身静态弯曲刚度，并通过试验进行了验证，得到下述结论：

1） 左右纵梁 Z 向位移变化情况基本相同，均在加载点附近最大，前后约束点位置最小。且从前约束点至车身前端和从前约束点至加载点，左右纵梁 Z 向位移逐渐增大；从后约束点至车身后端和从后约束点至加载点，左右纵梁 Z 向位移逐渐增大；

2） 车身整体弯曲刚度有限元分析和试验结果分别为 13108.88 N/m 和 14421.21 N/m，误差 9.10%，小于可接受误差范围 10%，有限元分析结果和试验结果基本吻合；

3） 较高的车身弯曲刚度可以获得更好的整车可靠性，车身弯曲刚度的目标设定需兼顾车身 NVH 和操稳性要求以及轻量化的性价比。

参考文献：

[1] 殷良艳,郭健忠,杨二卫. 白车身模态与静刚度关联性的研究—扭转[J].公路与汽运,2013(6):1-5.

[2] 夏国林.轿车白车身静刚度分析[J].汽车科技,2008(03):25-28.

[3] 王晓枫,任康. 轿车白车身刚度分析及轻量化设计研究[J].现代制造工程,2014(01):42-44,101.

[4] 王志亮,刘波,马莎莎,等. 基于弯曲刚度和扭转刚度的白车身优化分析[J].机械科学与技术,2008(8):1021-1024.

[5] 梅玉林,陈康阳,张健,等.车身弯曲刚度和扭转刚度试验研究[J].实验室科学,2016,19(2):20-24+27.

[6] 邵建旺,彭为,靳晓雄,等. SUV 白车身静态刚度试验研究[J].汽车技术,2009(04):41-44.

[7] 李海峰,吴冀川,刘建波,等. 有限元网格剖分与网格质量判定标准[J].中国机械工程,2012,23(3):368-377.

[8] 胡志远,曾必强,谢书港. 基于 LS-DYNA 和 Hyper Works 的汽车安全仿真与分析[M].北京:清华大学出版社,2011.

(上接第 48 页)

[6] Kabir M M J, Xu S, Kang B H, et al. A Hybrid GeneticMax Algorithm for Improving the Traditional Genetic Based Approach for Mining Maximal Frequent Item Sets[J]. International Journal of Computer Science & Network Security, 2014, 14(10):27-35.

[7] Niu X Z, She K. Mining Maximal Frequent Item Sets with Improved Algorithm of FPMAX[J]. Computer Science, 2013, 40(12):223-228.

[8] Ning H, Wang S, Cui L, et al. An algorithm for mining maximal frequent patterns based on improved FP-tree[J]. Applied Science & Technology, 2016,43(2):37-43.

[9] Pan Z, Liu P, Yi J. An Improved FP-Tree Algorithm for Mining Maximal Frequent Patterns[C].International Conference on Measuring Technology and Mechatronics Automation. IEEE Computer Society, 2018:309-312.

[10] Karim M R, Cochez M, Beyan O D, et al. Mining maximal frequent patterns in transactional databases and dynamic data streams: A spark-based approach[J]. Information Sciences, 2018, 432:278-300.

[11] Han J, Jian P, Yiwen Y. Mining frequent patterns without candidate generation[C]. Proceeding of the 2000 ACM SIGMOD International Conference Management of Data,Dallas,2000:1-12.

[12] 李少华,吕志旺,车德勇,等. 基于有序 FP-tree 的最大频繁项集挖掘算法[J].东北师大学报:自然科学版,2016, 48(2):65-69.