DOI: 10.13718/j.cnki.xdzk.2019.07.018

多最小支持度关联规则改进算法①

梁　杨，　钱晓东

兰州交通大学 电子与信息工程学院，兰州 730070

摘要：由于大数据具有多样性的特点，在数据挖掘过程中采用单一最小支持度会出现较多冗余规则，造成挖掘效率不高等问题，该文提出一种基于多最小支持度关联规则改进算法. 通过给每一项目设置单独的支持度阈值，构建多最小支持度模式树，利用最小频繁项目作为节点筛选标准，进行冗余节点删除；在挖掘频繁项集的过程中利用排序向下闭合的性质，删除冗余的候选项集，同时能够自动停止向下挖掘，从而快速直接地得到所有频繁项集，并且不需要多次扫描数据库. 实验结果表明，改进算法能够提高挖掘效率，节省计算时间.

关　键　词：大数据；频繁项集；关联规则；多最小支持度

中图分类号：TP274　　**文献标志码**：A　　**文章编号**：1673-9868(2019)07-0131-11

在这个大数据日益迅速发展的时代，人们对于数据的价值已经越来越重视. 根据 IDC(International Data Corporation)的研究报告预测，全世界数据总量将在 2020 年由 2013 年的 4.4 ZB 增长到 35 ZB[1]. 数据挖掘作为一个重要的技术手段已经被广泛应用于各个领域的研究当中，例如机器学习、模式识别、信息检索等[2-4]，而关联规则挖掘是数据挖掘中十分重要的手段之一，特别是在大数据时代下进行数据间的关联分析显得尤为重要. 由于大数据具有多样性的特点，使用单个支持阈值来评估数据库中所有项目的发生频率是不够的，因为每个项目不同，它们不应该被同等看待. 特别是在零售业，价格低廉的日常用品经常被购买，而奢侈品和高价位的产品却很少被购买；如果设置过高最小支持度阈值，只会挖掘到经常被购买的产品，而较低的最小支持度阈值又会产生较多无意义的规则，不利于产品的定位分析. 所以，传统的关联规则挖掘算法在处理海量数据时，容易挖掘出常规的或无效用规则.

为了处理数据间的差异性问题，Liu 等[5]首次提出一种多最小支持度框架下的关联规则算法 MSApriori，该算法通过给事务数据库中的每一个项目设定单独的最小支持度，将经典的 Apriori 算法扩展到关联规则的挖掘中，避免了设置单一支持度所产生的局限性. 但是，由于 Apriori 算法结构的缺陷，需要多次扫描数据库容易受到组合爆炸的影响，Hu 等[6]提出一种基于多最小支持度模式树的关联规则算法 CFP-growth，该算法基于 FP-tree 结构构造多最小模式树 MIS-tree 进行频繁项集的挖掘，由于 FP-tree 模型的优点，在挖掘过程中仅需 2 次扫描数据库来创建一系列条件模式树并生成完整的频繁项集，在一定程度上节省了挖掘时间. Tseng 等[7]提出了 MMS_Cumulate 和 MMS_Stratify 两种算法，在分类的情况下允许多种形式的最小支持度定义，并挖掘发现了广义的关联规则. Lee 等[8]提出了一种基于最大约束条件下的多最小支持度关联规则算法，在约束条件下进行关联规则挖掘，并证明利用最大约束得到的关联规则数目小于使用最小约束得到的数目，能够较为有效地减少无用规则产生.

随着研究深入，更多的基于多最小支持度挖掘关联规则算法被提出，例如基于属性关系多最小支持度的 REMMAR 算法[9]；模糊定量序列模式关联规则算法 FQDN-MMS[10]；一种基于多最小置信度的关联规

① 收稿日期：2018-09-11
基金项目：国家自然科学基金项目(71461017).
作者简介：梁　杨(1991-)，男，硕士研究生，主要从事数据挖掘研究.
通信作者：钱晓东，博士，教授.

则算法[11]等. 这些算法都从不同角度进行关联规则提取，但是随着数据不断增大，在挖掘过程中冗余造成时间和空间上的消耗，并且会出现较多的冗余规则，造成挖掘效率不高的问题. 所以，本文基于多最小支持度的定义，通过改进CFP-growth算法在频繁项集的挖掘过程中进行优化：将频繁项目最小支持度值作为初始项目的支持度阈值进行数据预处理，并在生成候选项集过程中利用排序向下闭合的属性进行冗余项删除，以达到解决挖掘时间和减少冗余规则产生的目的.

1 相关论述

1.1 多最小支持度定义

设项目集合为 $I=\{i_1, i_2, \cdots, i_n\}$，事务数据库为 $D=\{d_1, d_2, \cdots, d_m\}$，其中每一条事务为 $d_i(1\leqslant i\leqslant m)$，由一个事务编号 Tid 表示，d_i 为项目集合 I 的子集，即 $d_i\subseteq I$(其中包含 k 个项的项集为 k-项集)；若项集 $X\subseteq I$，其在事务数据库中出现的频率，即支持度计数记作 Support(X).

定义1 最小项支持度

对于上述项目集合 $I=\{i_1, i_2, \cdots, i_n\}$，对任意 $i\in I$，设定该项 i 一个最小支持度阈值，称之为最小项支持度(minimum item support)，记作 $MIS(i)$. 表1为某个数据库中各个项目所对应的 MIS 值.

表1 项目的 *MIS* 值

Item	a	b	c	d	e	f	g	h
MIS	5	5	6	7	2	3	3	4

定义2 最小项集支持度

对于项集 $X=\{i_s, \cdots, i_t\}$，$1\leqslant s\leqslant t\leqslant n$，其最小支持度为项集中最小的最小项支持度，记作 $MIN(X)$，即 $MIN(X)=\min[MIS(i_s), \cdots, MIS(i_t)]$.

定义3 频繁项集

在多最小支持度下，若项 i 的支持度计数 $Support(i)>MIS(i)$，则项 i 是频繁项；若项集 X 的支持度计数 $Support(X)>MIN(X)$，则项集 X 是频繁项集.

通过以上定义，在频繁项集的挖掘过程中，赋予事务数据库中每一项单独的最小支持度阈值，能够更加确切地反应事务本身的特点，因为得到的任意候选项集都是由其本身所需满足的最小支持度决定，而不是统一的最小支持度，所以基于多最小支持度关联规则算法的核心步骤为设定单独的支持度阈值，通过统计得到项集的支持度计数，再得到频繁项集，并进一步得到关联规则.

1.2 改进策略

由于CFP-growth算法中，用于构造MIS-tree的标准仍然考虑了一些在高阶候选相机中不能产生任何频繁模式的项目，同时在进行频繁项集的挖掘时，CFP-growth会递归地构建MIS-tree中各个项目条件模式树，直到其各自的条件模式为空，但是会构建一些不频繁项的条件模式基，而这些条件模式基所构建的条件模式树则不会产生任何高阶频繁模式. 因此，本文基于上述问题会导致效率不高的现象进行如下改进策略.

1.2.1 最小频繁项支持度(LMS)

根据上述定义可知，在多最小支持度模型中，判断项集是否为频繁项集是根据该项集中值最小的最小项支持度，这与单支持度模型下进行频繁项集的判定不一样，所以在多最小支持度下项集的最小支持度是一个变动的值，并且取决于该项集中的项，但是在多最小支持度模式下会出现频繁项集的子集可能不是频繁项集的情况. 例如某项集 $X=\{b, e, h\}$ 为频繁项集，根据表1可知 $MIN(X)=MIS(e)=2$，但其子集 $Y=\{b, h\}$ 的 $MIN(Y)=MIS(h)=4$ 则由于最小支持度阈值的改变，其子集可能不是频繁项集. 因此，为了更好地进行频繁项集挖掘，本文将项集中最小频繁项所对应的最小项支持度作为该项集的最小支持度阈值.

定义4 在多最小支持度下，称 LMS 为所有频繁项集合中最小的 MIS 值.

例如项集 $X=\{a, b, c, d\}$，若 a, c, d 为频繁项，则

$$LMS(X)=\min[MIS(a), MIS(c), MIS(d)]=0.5$$

性质1 在事务数据库中，若存在项集 $X=\{i_s, \cdots i_t\}$ 且 $X\subseteq I$，那么当 $s\leqslant k\leqslant t$ 时，若 $Support(X)<$

LMS，则 $Support(X)<\min[MIS(i_s),\cdots MIS(i_t)]$，即项集 X 是非频繁项集.

证　由于项集 X 的支持度小于 LMS，并且 LMS 又是频繁项目集合中最小的项，所以根据定义 3 可以得到该项集 X 是非频繁项集.

性质 2　在事务数据库中，若存在 2 个项集 X 和 Y，$X\subset Y$ 且 $Support(X)<LMS$，则 $Support(Y)<LMS$，即项集 Y 是非频繁项集.

证　由于在事务数据库中，$X\subset Y$，则根据先验原理知 $Support(Y)\leqslant Support(X)$，又因为 $Support(X)<LMS$，则 $Support(Y)\leqslant Support(X)<LMS$，根据性质 1 可知项集 Y 是非频繁项集.

以上 2 个性质表明，LMS 能够保证在多最小支持模式下获得的频繁项集具有全局反单调性，并且在频繁项集的生成过程中使用 LMS 作为约束条件，可用于修剪无法生成任何高阶频繁项集的项，能够减少搜索空间和提高挖掘效率.

1.2.2　条件最小支持度

由于多最小支持度模式树的构建过程遵循了 FP-tree 思想，并且该结构是将事务数据库中的每条事物按照项目的 MIS 值大小重新排序后进行构建，所以在结构树中越靠下位置的项 MIS 值越小. 因此，考虑存在于树中的某一项 i 作为后缀项并构造其前缀子路径(即条件模式基)时，该项 i 的 MIS 是其所有条件模式基中包含的项中最小的 MIS 值，因此从该 i 的条件模式基中生成的任何频繁项集都大于 i 的 MIS 值. 因此，本文在挖掘频繁项集的过程中以该后缀项 i 的 MIS 作为该条件模树种的条件最小支持度，

定义 5　当多最小支持度模式树存在一个项 x，$Support(x)$是该项集 X 的支持度，则 $MIS(x)$表示为 x 必须满足的最小支持度阈值；Y 是以 x 为后缀的条件模式基，y 表示为该条件模式基 Y 中的一个项，项 y 在该模式基中的支持度为 $Support(y)$，其最小支持度为 $MIS(y)$，则模式〈x，y〉的最小支持度为 $MIS(x)$.

1.2.3　排序向下闭合

由上所述可知，在多最小支持度模式下，若项集 X 是频繁项集，那么它的子集可能是非频繁项集，这与传统 FP-growth 算法中频繁项集向下闭合的性质有所差别，这是由于其子集的最小支持度阈值发生了改变所导致的. 在频繁项集的挖掘过程中，为了将频繁项集向下闭合的性质能够重新适用于多最小支持度模式，本文提出一种排序向下闭合的概念，重新对该性质进行形式化定义，为了更好地区别传统的挖掘方法，一个按照序号排序好的 k-项集表示为 $X=\{i_1,i_2,\cdots,i_k\}$.

性质 3　(排序向下闭合)若存在一个频繁 k-项集 $X=\{i_1,i_2,\cdots,i_k\}$(其中 $k\geqslant 2$，$X\subset I$)，且 $MIS(i_1)\leqslant MIS(i_2)\leqslant\cdots\leqslant MIS(i_k)$，那么任何包含项 i_1 的 $k-1$ 项集是频繁项集.

证　由于项集 X 是频繁项集，$MIS(i_1)\leqslant MIS(i_2)\leqslant\cdots\leqslant MIS(i_k)$，则根据定义 2 可知 $Support(X)>MIN(X)=MIS(i_1)$，假设项集 $Y\subset X$ 且 $i_1\in Y$，则 $MIN(Y)=MIS(i_1)$，又因为项集 Y 是项集 X 的子集，则 $Support(Y)\geqslant Support(X)$，所以 $Support(Y)\geqslant Support(X)\geqslant MIS(i_1)=MIN(Y)$，因此项集 Y 是频繁项集.

根据性质 3，可以推导其反单调性.

性质 4　若 $k-1$ 项集 $X=\{i_1,i_2,\cdots,i_{k-1}\}$($X\subset I$)是非频繁项集且 $MIS(i_1)\leqslant MIS(i_2)\leqslant\cdots\leqslant MIS(i_{k-1})$，若存在一个 $i_k\in I$，使 $MIS(i_1)\leqslant MIS(i_k)$，则 k-项集也是非频繁项集.

根据这 2 个性质就可以解决在多最小支持度模式下频繁项集的子集是否为频繁项的问题，同时由于本文采用基于 FP-tree 结构的模式，其构建条件模式树是按照各项的最小支持度进行路径排序的，所以若后缀项不频繁，那么它的所有超后缀项集也将不频繁.

2　算法介绍

2.1　算法思想

基于设定单最小支持度容易造成效率不高的问题，本文利用多最小支持度思想，在挖掘过程中通过改进多最小支持度模式树和提高挖掘频繁项集效率进行关联规则提取. 首先为项目集合 $I=\{i_1,i_2,\cdots,i_n\}$ 中每一项设置独立的支持度 MIS，将事务数据库 $D=\{d_1,d_2,\cdots,d_m\}$中的每一条事务按照 MIS 大小顺

序重新进行降序排列，扫描排序后事务数据库，按照FP-tree建立多最小支持度模式树NMIS-tree，并构建项目头表Head-table，根据定义4中LMS，利用其性质将LMS值作为频繁项的筛选标准，删除部分不能产生任何频繁项集的项，并对初始NMIS-tree进行剪枝和合并，得到完整的NMIS-tree.

在得到完整NMIS-tree后进行频繁项集挖掘，利用Apriori中候选项集生成的方法得到每一项的候选项集，并根据重新定义后排序向下闭合的性质，对无法产生任何高阶频繁的项集停止挖掘，减少冗余候选项集的产生.

2.2 算法流程

2.2.1 NMIS-tree构建

多最小支持度模式树主要包含2个部分：MIS-List和前缀树节点. Head-table主要包含4个部分：项目名称(*item*)，支持度计数值(Sup)，各项最小支持度(MIS)，节点链接(*node-link*)；其中*node-link*表示为一个指针，指向多最小支持度模式树中具有相同项目名的项. Head-table中所有项目的顺序按照MIS值大小进行降序排列. 前缀树节点主要包含3个部分：项目名称(*item*)，支持度计数值(Sup)，节点链头(*link*). 该步骤如下：

1) 将排序后的每一条事务按照MIS值顺序插入到NMIS-tree中，得到初始NMIS-tree；

2) 删除初始NMIS-tree中的冗余项；

3) 合并NMIS-tree中可能包含相同路径的子节点，得到完整的NMIS-tree.

具体描述过程如表2所示.

表2 构建NMIS-tree算法

算法 构建NMIS-tree

输入：事务数据库D，包含n项的项集I，各个项的MIS

输出：NMIS-tree

步骤：

1：创建NMIS-tree的根节点Null；

2：for 每一条事务$t\in D$ do

3：将每一条事务中的项按照MIS值大小进行降序排列；

4：计算每个项的支持度，记作$Sup(i)$；

5：将排序后的每一条事务记作$[p|P]$，p代表第一个元素，P代表该事务剩余的集合，

6：调用$insert_tree([p|P], T)$

7：end for

8：for (；$j\geqslant 0$；$j=j-1$) do

9：if ($Sup(i_j)<MIS(i_j)$) then

10：删除头表中的项i_j；调用$NMIS_Pruning(NMIS\text{-}tree, \overline{f})$；

11：else

12：$LMS=MIS[i_j]$；break；

13：end if

14：end for

15：for (；$j\geqslant 0$；$j=j-1$) do

16：if ($Sup(i_j)<LMS$) then

17：删除头表中的项i_j；调用$NMIS_Pruning(NMIS\text{-}tree, \overline{f})$

18：end if

19：end for

20：调用MIN_Merge(NMIS-tree)

其中，过程 *insert_tree*([*p*|*P*], *T*)表示将每一条事务中的项插入多最小支持度树中，具体描述如表 3 所示.

表 3　*insert_tree* 过程

过程 1　*insert_tree*([*p*|*P*], *T*)
1：if 存在一个 *T* 的子节点 *N*，使得 *p*.*item-name*＝*N*.*item-name then*
2：*N* 的 Count 增加 1；
3：else
4：创建一个新的子节点 *N*，使其节点的 Conut 为 1，并链接到父节点 *T*；
5：通过节点链接的结构将该节点链接到同名的节点上；
6：end if
7：if $P \notin \emptyset$ then
8：调用 *insert_tree*([*p*|*P*], *T*)
9：end if

过程 *NMIS_Pruning*(*NMIS-tree*, $\overline{f}$)表示进行多最小支持度树中冗余节点删除过程，具体描述如表 4 所示.

表 4　*NMIS_Pruning* 过程

过程 2　*NMIS_Pruning*(*NMIS-tree*, $\overline{f}$)
1：for 任意在 NMIS-tree 与 i_j 链接的节点
2：if 如果该节点是叶子节点 then 直接删除；
3：else 删除该节点，并将该节点的子节点链接到它的父节点上；
4：end if
5：end for

过程 *MIN_Merge*(*NMIS-tree*)表示进行多最小支持度树中相同节点的合并过程，具体描述如表 5 所示.

表 5　*MIN_Merge* 过程

过程 3　*MIN_Merge*(*NMIS-tree*)
1：for Frequent header table 中每一项 do
2：if 存在相同名称的子节点 then
3：合并这些节点，支持度计数为这些节点的计数之和；
4：end if
5：end for

2.2.2　NCFP-growth

根据上述过程构建多最小支持度模式树 NMIS-tree，在得到频繁 2-项集后，通过 Apriori 中获取候选项集的方法得到高阶条件候选项集，然后利用排序向下闭合的性质进行候选项集压缩，最后进行频繁项集挖掘. 具体描述过程如表 6 所示.

2.3　算法示例

2.3.1　*数据库预处理*

数据预处理是将原始事务数据库进行重新排序.

表 7 是一个包含 10 个事务的数据库，从编号 1 到编号 10 分别对应不同的事务数据集. 首先根据表 1 所示的各项 *MIS* 值由大到小进行重新排序，若存在相同 *MIS* 值得项则按照字母顺序进行排列，最终得到排序后的事务数据库，如表 8 所示.

表6 NCFP-growth

算法 NCFP-growth
输入：$NMIS\text{-}tree$，$MIS(i_j)$，k
输出：频繁项集
步骤：
1：for each i_j in the header of $NMIS\text{-}tree$ do
2：if $k=2$ then
3：generate pattern $\beta=i_j \cup \alpha$ with i_j. support；
4：else
5：generate pattern $\beta=Apriori\text{-}gen(k, Fs)$；
6：end if
7：end for
8：Construct set of β's conditional pattern do
9：for each β in the set of β's conditional pattern do
10：if β is frequent then
11：Fs. add(β)；
12：Call NCFP-growth($NMIS\text{-}tree$, $k+1$, β, $MIS(i_j)$)

表7 原始事务数据库 D

Tid	Items	Tid	Items
1	c, d	6	c, d
2	a, d	7	a, c, d, h
3	b, c, d, g	8	b, d, f
4	a, b, c, f, h	9	a, b, c, e, f
5	a, d, g	10	a, c

表8 排序后的事务数据库 D

Tid	Items	Tid	Items
1	c, d	6	c, d
2	d, a	7	c, d, a, h
3	c, d, b, g	8	d, b, f
4	c, a, b, f, h	9	c, a, b, f, e
5	d, a, g	10	c, a

2.3.2 构建 NMIS-tree

构建 NMIS-tree 是将排序后的事务数据库按照类似 FP-tree 的构建方法进行构建，具体过程如下：

创建项目头表 Header-Table，其中包含项目名、各项的 MIS 值以及该项的支持度计数 count，其中 item 一栏按照 MIS 值从大到小的顺序进行排列；创建根节点 Null，并将排序后事务数据库中的每一条事务插入到 NMIS-tree 中，并更新记录各项的支持度计数，最终得到初始 NMIS-tree 如图1所示.

2.3.3 冗余剪枝

冗余剪枝是将项目头表中某些项进行删除，通过前述中 LMS 的定义，将 LMS 作为冗余项删除的标准，并且删除与之对应的初始 NMIS-tree 中的节点，具体过程如下：

判断项目集合中的频繁项，根据各个项目的 Support 和 MIS，按照定义3可知 Header-Table 中频繁项为$\{c\},\{d\},\{d\},\{f\}$，所以 $LMS=min[MIS(c), MIS(d), MIS(a), MIS(f)]=3$，则删除项为$\{h\}$，$\{g\},\{e\}$，并将对应 NMIS-tree 中的节点进行删除，剪枝过程如图2所示.

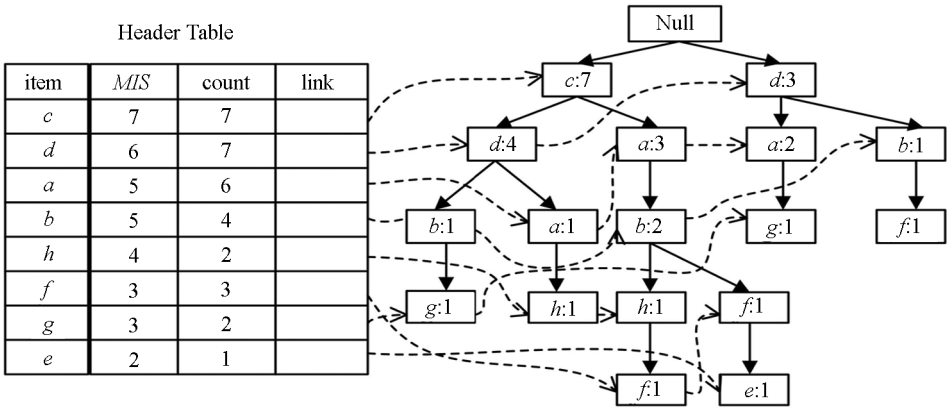

图 1　初始 NMIS-tree

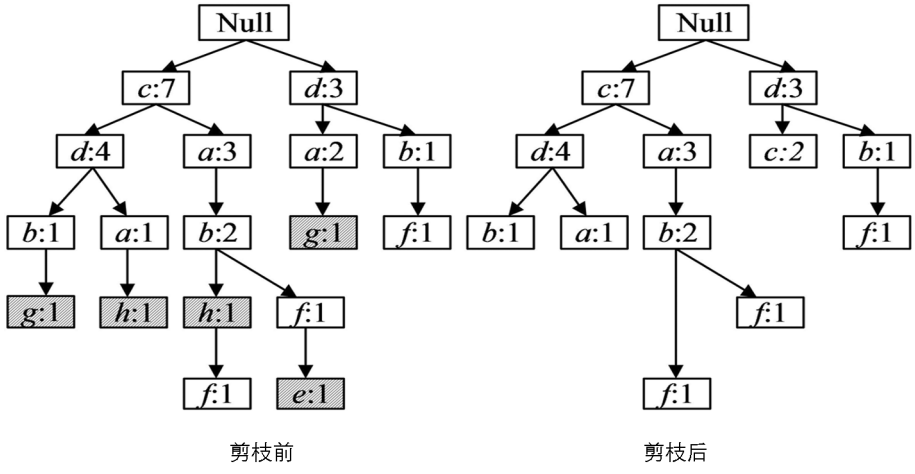

图 2　剪枝过程

2.3.4　合并重复路径

合并重复路径是将剪枝后 NMIS-tree 中链接在同一父节点的同名子节点进行路径合并，支持度计数为该同名节点支持度计数之和.

遍历剪枝后的 NMIS-tree，节点〈b：2〉含有同名但不同路径的子节点〈f：1〉，合并这 2 个子节点，并重新计算支持度计数得到完整 NMIS-tree，如图 3 所示.

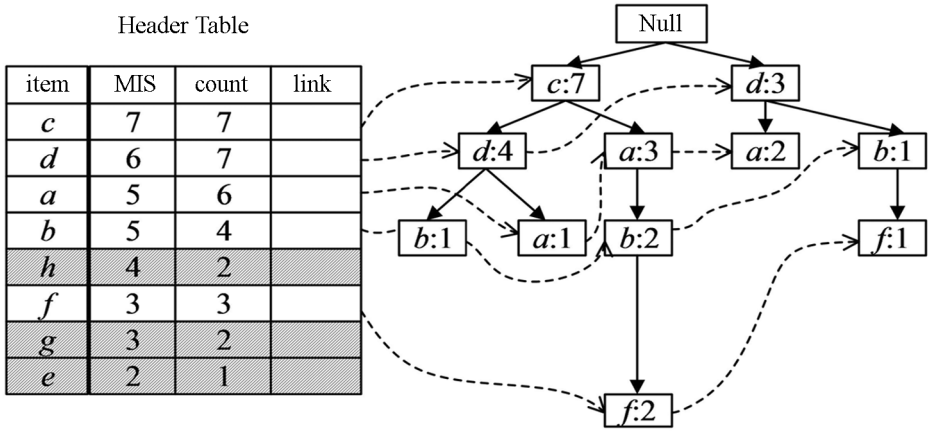

图 3　完整 NMIS-tree

2.3.5 频繁项集挖掘

在得到完整NMIS-tree之后，按照项目头表中各项的*MIS*值从小到大进行频繁项集挖掘，但是根据排序向下闭合的特性可知，由于项a是非频繁项目，以其为后缀建立的条件候选项集都是非频繁项集，所以挖掘顺序从$f\rightarrow b\rightarrow a\rightarrow d\rightarrow c$变为$f\rightarrow b\rightarrow d\rightarrow c$.

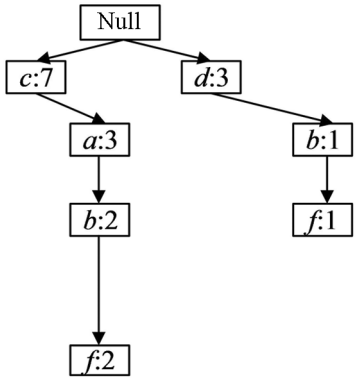

图4 项f的前缀路径

频繁项集挖掘过程从Header-Table最底端的项进行挖掘，对于频繁项$\{f\}$，以f为后缀项建立其前缀路径的条件模式如图4所示. 由于项f为频繁项，从2-条件候选项集($\{b,f\},\{a,f\},\{c,f\},\{d,f\}$)进行挖掘. 本文利用*LMS*性质进行频繁项集筛选，由于只有项集$\{b,f\}$的支持度计数不小于*LMS*值，所以得到$\{b,f\}$是频繁项集.

对于3-条件候选项集进行挖掘时，本文通过使用Apriori算法中生成候选项集的方法得到3-条件候选项集，为了更好地表明该步骤中利用了排序向下闭合中的反单调性，将所有候选项集标出如图5所示. 例如通过项集$\{d,f\}$产生的3-候选项集$\{d,b,f\},\{d,a,f\},\{c,d,f\}$，可以看到这些候选项集的最小支持度并没有发生改变，并且最小支持度为$MIS(f)$. 所以，根据性质3可以得到这些候选项集都是非频繁项集. 因此，当挖掘到2-项集时，只需要将满足*LMS*值的项进行高阶项集挖掘，其余项集已经自动停止挖掘，所以能够减少大量候选项集的生成，极大地提高了挖掘效率. 而对于满足*LMS*的项集$\{b,f\}$，由于其无法产生3-项集，所以挖掘过程结束，得到以f为后缀的频繁项集为$\{b,f\}$.

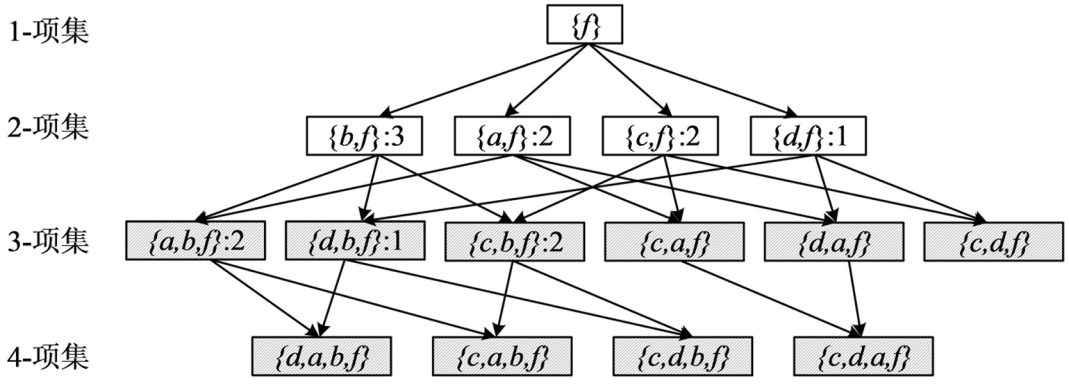

图5 候选项集生成过程

3 实证分析

3.1 实验环境及数据

本实验环境：CPU主频2.40 GHz，内存8.00 GB，操作系统为Windows 10，JAVA 语言环境. 本文采用表9所示Transaction数据集和模拟合成的大规模数据集进行测试，其中Transaction数据集包含40 000+记录、20种属性；模拟合成数据集基于文献[12]中的方法，生成不同数量的大规模数据集T10I4D100K.

表9 数据集

数据集	项的个数	事务数	平均长度	最大长度
T10I4D100K	1 000	100 000	10	29
Transaction	20	46 243	5.5	68

注：平均长度和最大长度的单位，都为项的个数.

对于每项*MIS*值设置如下：

$$MIS(i)=\begin{cases}M(i) & M(i)>MS\\ MS & \text{otherwise}\end{cases}$$

$$M(i)=\sigma\times f(i)$$

其中，$f(i)$是项目 i 的实际频率，MS 代表所有项目中最小的 MIS 值，σ 用来控制项目的 MIS 值和发生频率之间的关系；当 $\sigma=0$ 时，等同于在单一最小支持度下挖掘频繁项集.

3.2　实验结果分析

3.2.1　时间消耗

1）图 6 为 Transaction 和 T10I4D100K 两个数据集下不同 σ 值进行数据挖掘所耗费的时间分析，本实验设置 $MS=0.001$.

通过图 6 可知，在不同数据集中随着值的改变，改进算法相比于 CFP-growth 和 CFP-growth＋＋算法表现得更加良好. 随着 σ 值增大，3 种算法的挖掘时间都随之缩短，改进算法在大数据集下的趋势表现更为明显，这是由于 σ 值增大，项目实际的支持度阈值也会被设置为一个较大的值，较多的项目会被过滤，所以构建的条件模式树较为稀疏，挖掘频繁项集的数量会减少，从而时间缩短幅度较大.

在 Transaction 数据集下，随着 σ 值增加改进算法比 CFP 算法节省将近一个数量级的挖掘时间，并且性能更加优越于 CFP＋＋算法，这是由于改进算法在频繁项集挖掘过程中不需要每一项都向下进行挖掘，因此提高了挖掘效率. 在 T10I4D100K 数据集下，由于设置的 MS 值非常小，CFP-growth 必须构造大量的条件模式树进行频繁项集挖掘，从而耗时过多. 改进算法在构建条件模式树的过程中，能够进行冗余节点删除和重复路径合并使结构树更加简洁，避免了计算资源的浪费，从而大幅度提高挖掘过程中的性能表现.

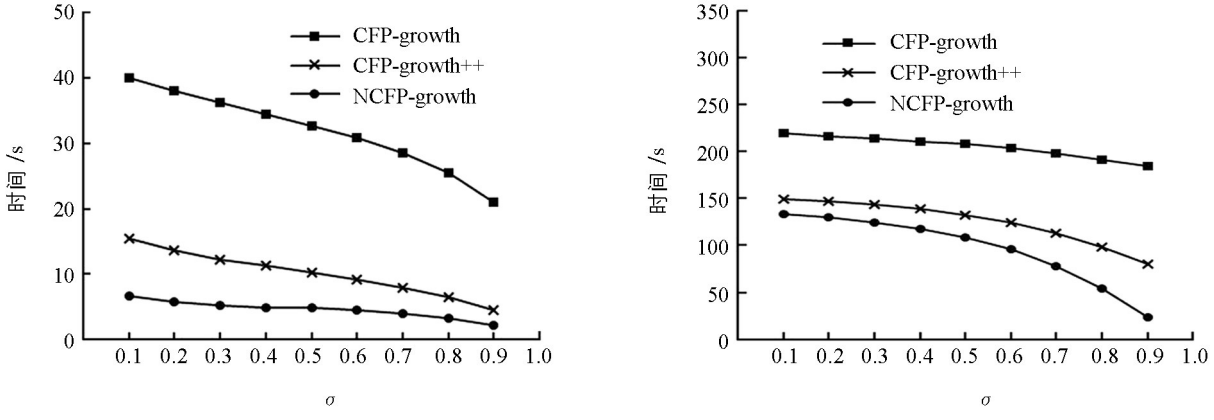

图 6　不同 σ 值下挖掘时间对比

2）图 7 为 Transaction 和 T10I4D100K 两个数据集在不同 MIN 值下的挖掘时间对比($\sigma=0.2$ 和 $\sigma=0.1$).

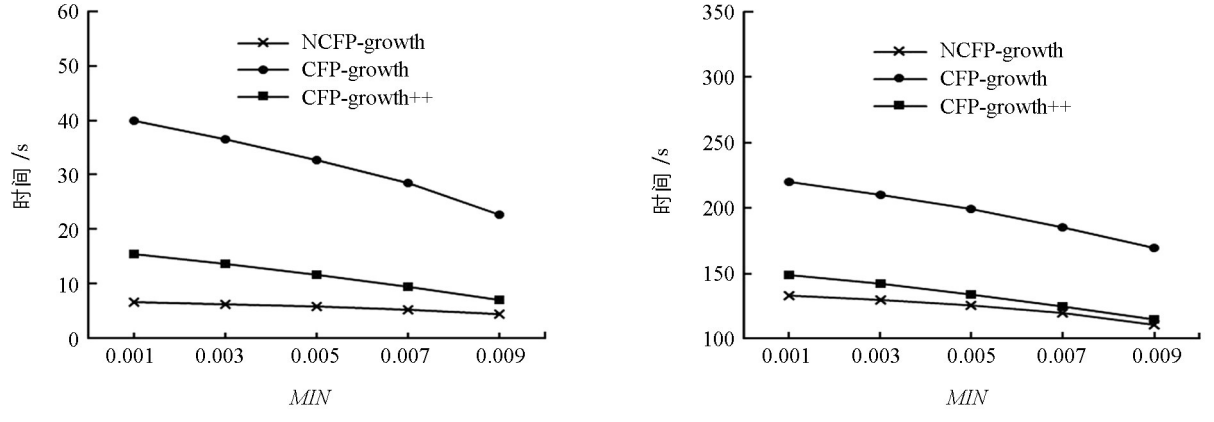

图 7　不同 *MIN* 值下挖掘时间对比

以上 2 组实验分组在固定的 σ 值下，为了更好地观察 MIN 值改变对于各个项目最小支持度的影响，实验分别设定 $\sigma=0.2$ 和 $\sigma=0.1$，将结果进行记录得到时间对比图如图 7 所示.

通过图7可知，在不同数据集下通过固定σ值，本文所提算法挖掘时间都少于CFP-growth和CFP-growth++，并且明显优异于CFP-growth算法，这说明了改进算法在挖掘频繁项集的过程中能够减少大量无用项集的产生，提高挖掘效率；同时随着MIN值增大，改进算法和其余2种算法的挖掘时间都有所下降，改进算法改变幅度较小，而CFP-growth和CFP-growth++算法改变幅度较大，这是由于较大MIN值会影响各个项目的最小支持度阈值，从而改变多最小支持度模式树的结构，使得候选项集的生成数目有所减少，从而缩短挖掘频繁项集的时间.

3.2.2 效率性能

为了更好地分析算法的挖掘效率，观察其可延展性，通过设置不同数据量的同一数据集(T10I4D100K，T10I4D200K，T10I4D300K，T10I4D400K)进行对比分析如图8. 本实验设置$MS=0.001$，$\sigma=0.5$.

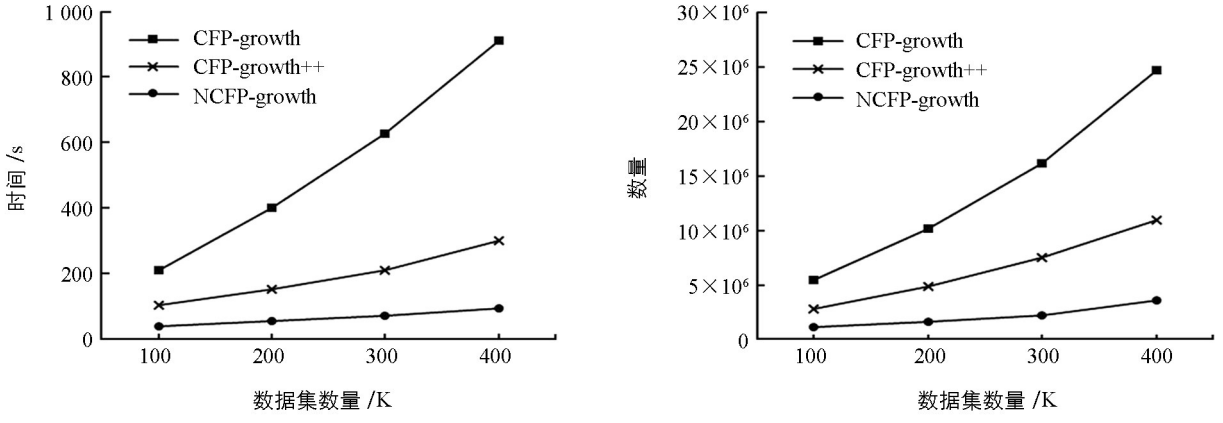

图8 挖掘时间和候选项集数量对比

通过图8可知，3种算法随着数据集数量增加，其挖掘时间和候选项集数量都呈一个近似线性增长，但改进算法比CFP-growth和CFP-growth++算法表现出更高的效率. 随着数据集数量增加，CFP-growth和CFP-growth++算法耗费时间明显增多，改进算法呈现出较小幅度的变化，并且在同一数据集下，改进算法比CFP-growth节省将近一个数量级的时间，表现出较高的挖掘效率，这是由于数据集的数量增加，前2种算法需要更多次数的递归调用进行挖掘，而改进算法在挖掘频繁项集的过程中能够优化条件模式和减少递归次数，从而较大幅度地降低挖掘时间.

通过图8(b)中候选项集挖掘数量的对比可知，随着数据集数量增加，改进算法挖掘得到的候选项集数量增幅很小，而CFP-growth和CFP-growth++算法增长幅度明显，这说明了改进算法在大数据集下具有较高的稳定性和适应性. 同时，在同一数据集下，改进算法得到的候选项集数量少于CFP-growth算法一个数量级之多，低于CFP-growth++算法所耗费时间的一半，这是由于改进算法在构造条件模式树挖掘频繁项集的过程中能够删除多余候选项集，这也说明了改进算法在大数据集下能够较大程度地节省时间和提高效率.

4 结 论

本文在挖掘关联规则的过程中，针对单一支持度算法的局限性以及挖掘效率不足的问题，提出一种基于多最小支持度的挖掘算法. 在挖掘关联规则的主要步骤即挖掘频繁模式的过程中，以LMS值作为筛选标准，在生成频繁项集的步骤中将无法产生频繁项集的候选项集直接删除，从而快速得到频繁项集；通过实验结果对比，证明了本文所提改进算法能够大幅地提高挖掘效率，并且在大规模数据集下的性能表现良好. 由于数据规模的不断增大，对于算法运行存储空间的要求越来越高. 随着分布式储存和计算的迅猛发展，如何利用有限的资源去获取数据价值的探讨和研究十分必要.

参考文献：

[1] GANTZ J, REINSEL D. 2011 Digital Universe Study: Extracting Value from Chaos[M]. Hopkinton: IDC Go-to-Market Services, 2011.

[2] WU F, WANG Z, ZHANG Z, et al. Weakly Semi-Supervised Deep Learning for Multi-Label Image Annotation [J]. IEEE Transactions on Big Data, 2015, 1(3): 109-122.

[3] CORMACK G V , CLARKE C L A , BUTTCHER S. Information Retrieval: Implementing and Evaluating Search Engines [J]. The Electronic Library, 2011, 29(6): 853-854.

[4] NAUN C C. Book Review: Introduction to Modern Information Retrieval [J]. Library Resources & Technical Services, 2011, 55(4): 239-240.

[5] LIU B, HSU W, MA Y M. Mining Association Rules with Multiple Minimum Supports [C]//San Diego: Proceedings of the Fifth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining-KDD'99, 1999.

[6] HU Y H, CHEN Y L. Mining Association Rules with Multiple Minimum Supports: A New Mining Algorithm and a Support Tuning Mechanism [J]. Decision Support Systems, 2006, 42(1): 1-24.

[7] TSENG M C, LIN W Y. Efficient Mining of Generalized Association Rules with Non-Uniform Minimum Support [J]. Data & Knowledge Engineering, 2007, 62(1): 41-64.

[8] LEE Y C, HONG T P, LIN W Y. Mining Association Rules with Multiple Minimum Supports Using Maximum Constraints [J]. International Journal of Approximate Reasoning, 2005, 40(1-2): 44-54.

[9] LIU Y C, CHENG C P, TSENG V S. Discovering Relational-Based Association Rules with Multiple Minimum Supports on Microarray Datasets [J]. Bioinformatics, 2011, 27(22): 3142-3148.

[10] HUANG T C K. Discovery of Fuzzy Quantitative Sequential Patterns with Multiple Minimum Supports and Adjustable Membership Functions [J]. Information Sciences, 2013, 222: 126-146.

[11] RAGE U K, KITSUREGAWA M. Efficient Discovery of Correlated Patterns Using Multiple Minimum All-Confidence Thresholds [J]. Journal of Intelligent Information Systems, 2015, 45(3): 357-377.

[12] TANG K, CHEN Y L, HU H W. Context-Based Market Basket Analysis in a Multiple-Store Environment [J]. Decision Support Systems, 2008, 45(1): 150-163.

An Improved Algorithm for Association Rules with Multiple Minimum Supports

LIANG　Yang,　QIAN Xiao-dong

School of Electronic and Information Engineering, Lanzhou Jiaotong University, Lanzhou 730070, China

Abstract: Due to the diversity of big data, using a single minimum support in the data mining process will result in inefficient mining and redundancy rules. This paper proposes an improved algorithm based on multi-minimum support association rules. By setting a separate support threshold for each project, a multi-minimum support pattern tree is constructed, and the minimum frequent items are used as node screening criteria to perform redundant node deletion. In the process of mining frequent itemsets, the nature of sorting down-close is utilized to delete redundant candidate sets, and at the same time, it can automatically stop down mining, so that all frequent itemsets can be quickly and directly obtained, and the database does not need to be scanned multiple times. Experimental results show that the improved algorithm can improve mining efficiency and save computing time.

Key words: big data; frequent itemset; association rule; multiple minimum support

责任编辑　夏　娟