频繁模式挖掘的约束算法

孟彩霞
(西安邮电学院 计算机系,陕西 西安 710065)

摘 要:在频繁模式挖掘过程中能够动态改变约束的算法比较少.提出了一种基于约束的频繁模式挖掘算法 MCFP. MCFP首先按照约束的性质来建立频繁模式树,并且只需扫描一遍数据库,然后建立每个项的条件树,挖掘以该项为前缀的最大频繁模式,并用最大模式树来存储,最后根据最大模式来找出所有支持度明确的频繁模式. MCFP算法允许用户在挖掘频繁模式过程中动态地改变约束.实验表明,该算法与 iCFP算法相比是很有效的.

关键词:频繁模式挖掘;动态约束;频繁项集;最大频繁模式
中图分类号: TP311 **文献标识码**: A **文章编号**: 1673-4785 (2009) 02-0142-06

A frequent pattern mining algorithm based on constraints

MENG Cai-xia
(Department of Computer Science, Xi'an University of Posts & Telecommunications, Xi'an 710065, China)

Abstract: Most algorithms don't allow users to dynamically change constraints in the process of mining frequent patterns. A new algorithm, constrain-based frequent patterns mining, was developed to provide frequent pattern mining with constraints. First, the algorithm constructs the FP-tree (frequent pattern tree) according to the descending or ascending order of constraints, and in this process the database only needs to be scanned once. Secondly, the conditional tree of each item was established to mine maximal frequent pattern with this term as a prefix, and the maximal frequent patterns were stored. Finally, all frequent patterns with precise support degrees were discovered according to the maximal frequent patterns. The significance of this method is that this algorithm allows users to dynamically change constraints during the process. Experimental outcomes showed that the proposed algorithm is more efficient than the algorithm of iCFP.

Keywords: frequent patterns mining; dynamic constraints; frequent item set; maximal frequent pattern

关联规则挖掘是数据挖掘领域研究的重要课题.关联规则挖掘问题的解决分为 2步: 1)找出所有的频繁项集 (频繁模式); 2)由频繁项集产生强关联规则.这 2步中,频繁项集挖掘是基本步骤,是关联规则挖掘问题的核心[1-3]. Agrawal等人提出了经典的 Apriori算法[4],但是该算法要产生大量的候选项集并需多次扫描数据库,大大降低了算法的性能.后来一些类 Apriori算法虽然对 Apriori做了很多改进,但是同样存在着这样的问题[1,5]. Hen等人提出了基于 FP-tree结构的不用产生候选项集的 FP-growth算法[6],该算法只需扫描 2次数据库来建立 FP-tree,大大提高了挖掘效率.目前,出现了有关挖掘最大频繁模式算法[7]、基于约束的频繁模式挖掘算法[8]等.对于交互式频繁模式挖掘, Zhu等人提出了 IUFP-Tree算法[9],它只是针对支持度不断变化的交互式挖掘算法. Leng把约束和交互式相结合提出了 iCFP算法[10],该算法允许用户在挖掘过程中动态改变约束,且不必重新扫描数据库,但是在找频繁项集时,需要不断建立条件树,占用了大量内存.

收稿日期: 2008-12-16.
基金项目:陕西省自然科学基金资助项目 (2004f283); 西安市科技创新支撑 —应用发展研究计划资助项目 (YF07024).
通信作者:孟彩霞. E-mail: mcxmcx@xiyou.edu.cn

提出了一个在频繁模式挖掘过程中可以改变约束的算法,该算法不用反复建立条件树,且在建树时是按照约束的大小顺序来建立的,并且每次只有一个条件树存在于内存中,挖掘完该条件树后立即释放,和现有的 iCFP相比节省了大量内存. 为了验证算法的性能,进行了一系列的实验来评价算法,实验表明该算法的优越性是明显的.

1　问题定义

设 $I = \{i_1, i_2, \cdots, i_n\}$ 是 n个不同项目的集合,其中 $i_j (j = 1, 2, \cdots, n)$ 是数据单项. 事务数据库 $DB = <T_1, T_2, \cdots, T_m>$,其中每个事务 $T_j (j = 1, 2, \cdots, m)$ 是 I中一组项目的集合,即 $T_j \subseteq I$, T_j有一个惟一的标识符 TID.

定义 1　对于一个集合 X,如果 $X \subseteq I$,且 X包含 k个项目,则称 X为 k - 项集,简称为项集.

定义 2　如果项集 $X \subseteq T_j$,则称事务 T_j包含项集 X; 项集 X在事务数据库 DB中的支持度,记为 $\mathrm{Sup}_{DB}(X)$,即事务数据库 DB中包含项集 X的事务个数.

定义 3　如果项集 X在事务数据库 DB中的支持度 $\mathrm{Sup}_{DB}(X)$不小于用户或专家给定的最小支持度阈值 δ,则称项集 X为频繁项集(频繁模式); 反之称之为非频繁项集.

定义 4　给定最小支持度 δ,对于项集 $X \subseteq I$,若 $\mathrm{Sup}_{DB}(X) \geq \delta$,且对于 $\forall Y (Y \subseteq I \wedge X \subset Y)$,均有 $\mathrm{Sup}_{DB}(Y) < \delta$,则称 X为 DB中的最大频繁项集(或称最大频繁模式). 也就是说,如果频繁项集 X的所有超集都是非频繁项集,那么 X为最大频繁项集.

性质 1　频繁项集的任何真子集都不是最大频繁项集.

性质 2　频繁项集的任何子集都是频繁项集.

显然,任何频繁项集都是某一个最大频繁项集的子集,所以可以把挖掘所有频繁项集的问题转化为挖掘最大频繁项集的问题.

对于项目集合 I中的每个单项可能有预先定义的属性,例如有属性 A、B. 定义在项集上的约束 $\theta \in \{\mathrm{true}, \mathrm{false}\}$,如果 $\theta(X) = \mathrm{true}$,则项集 X满足该约束. 目前常用的约束分为 3大类:反单调性约束、单调性约束和简洁性约束.

定义 5　反单调性(anti-monotone)约束. 如果项集 X不满足约束 θ,它的超集也不满足,则 θ是反单调性约束.

定义 6　单调性(monotone)约束. 如果项集 X满足约束 θ,它的所有超集也都满足,则 θ是单调性约束.

定义 7　简洁性(succinct)约束. 如果可以直接精确地产生满足约束的所有项集,而不用产生不满足约束的项集,则该约束 θ为简洁性约束.

例如,$\min(X.A) \geq v$是简洁反单调性约束,$\max(X.A) \geq v$是简洁单调性约束.

2　算法 MCFP

算法 MCFP是针对 iCFP存在的问题(即不断建立条件树)而提出的一种改进方法.

2.1　数据结构

表 1和表 2分别是一个事务数据库和一个数据单项的属性表. 这里假设数据单项有属性 A和 B.

表 1　事务数据库

Table 1　Transaction database

TID	Items
1	abcd
2	bdf
3	abce
4	abde

表 2　数据单项的属性表

Table 2　Attributes of data item

item	A	B
A	60	300
B	30	400
C	35	500
D	40	450
E	55	600
F	65	380

这里按照属性 A上值的大小顺序来建立 FP-tree,这样仅需扫描一遍数据库,然后建立满足简洁性约束的项的条件树. 假设简洁性约束为 $C_s = \min(s.A) \geq 30$,最小支持度 $\delta = 2$

在 FP-tree中设计了 2个数据结构: List和 FP-tree. List用于存储可能的频繁单项, FP-tree用于存储可能的频繁项集,具体说明如下:

List一个三元组表. 每个三元组 (A, item-name, δ),其中 A是数据单项属性表中有约束的属性上的分量值, item-name是单项的名称, δ是该项的支持度. List表中三元组按照 A上的值从大到小

(或从小到大)顺序排列.

FP-tree:一个字典树,每个节点是一个对偶(item-name, F),其中 item-name是项名, F是由 FP-tree的根节点至当前节点的路径上所有项组成的项集的支持度.

2.2 建立 FP-tree

建立 FP-tree的步骤如下:

1)把项按照 A的值从大到小顺序排序,放进 List表中.

2)扫描事务数据库,根据 List中项的顺序建立 FP-tree,把每次涉及到的项的支持度(即 List表中项的 δ值)加 1,直到最后一个事务,同时建立相同项之间的连接.此时包含非频繁项的树建立完毕,如图 1所示.

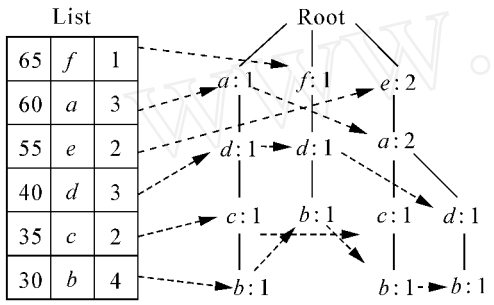

图 1 包含非频繁项的树的结构

Fig 1 Tree included infrequent item set

3)查 List表去掉表中非频繁项.当去掉非频繁项时,根据连接重新调整树的结构.由图 1中 List可知 f的支持度小于 2,去掉项 f后的 FP-tree如图 2所示.

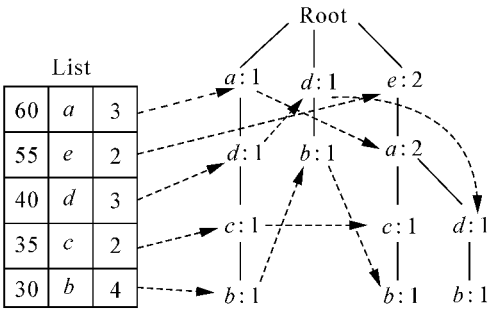

图 2 FP-tree结构

Fig 2 FP-tree structure

算法 1 建立 FP-tree的算法.

输入:DB,最小支持度 δ,简洁性约束;

输出:FP-tree

方法:

1)按照每个单项的属性 A值由大到小(简洁单调性约束)或由小到大(简洁反单调性约束)创建 List表;

2)初始化 FP-tree,根指针 T为"NULL";

3) For DB中的每个事务 t

① If事务 t的项集已经是 T的一个分支(子树){

② 该分支上的节点 N的支持度增加 1;

③ List中该项集的每个单项的支持度增加 1;}

④ Else {

⑤ 创建新的节点 N,存放事务 T的项集;

⑥ 节点 N和 T连接起来;

⑦ 具有相同项的节点连接起来;}

⑧ End If

⑨End For

4) If N的支持度小于最小支持度

5)删除节点 N以及和 N有关的连接;

6) End If

2.3 建立满足约束的条件树找出最大频繁模式

首先从 List表中最后一项开始,判断各项是否满足给定的条件.如果不满足,则不用建立该项的条件树,这样就把约束放到了挖掘过程中,根据约束的性质来降低搜索空间.如果找到满足约束的项,则建立该项的条件树,并把该树中的非频繁项即局部非频繁项去掉.由图 2可知,最后一项是 b, b满足假设的简洁性约束 $C_s = \min(s\ A) \geq 30$, b的条件树如图 3所示.由于 b的条件树中没有局部非频繁项,所以不用再重新调整该树的结构.

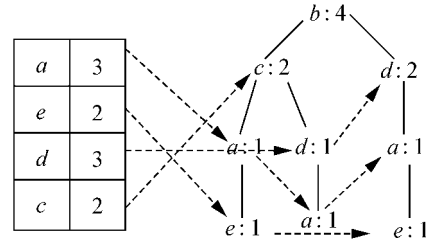

图 3 b的条件树

Fig 3 Condition-tree of b

然后用该树的分支两两相交来得到局部最大频繁模式.例如 b的条件树前 2个分支 $bcae$:1和 $bcda$:1相交得到 bca:2.然后按照支持度由大到小的顺序插入到最大模式树中.最大模式树初始值为空,如果某项集已存在于最大模式树中或是已存在的最大模式树的子集,则不再插入该项集.挖掘完 b

的条件树后得到的最大模式树如图 4所示.

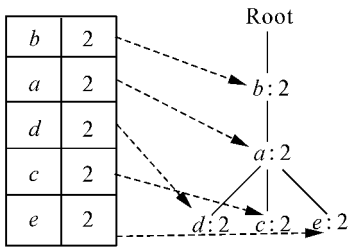

图 4　挖掘完 *b*条件树后的最大模式树

Fig 4　Maximal patterns tree after mining *b*

同理,分别建立 *c*、*d*、*a*、*e*的条件树,依次得到局部最大模式插入到最大模式中,最后得到的最大模式树即为图 4所示.

由图 4可知,最后的最大频繁模式为 *bac*: 2、*bae*: 2、*bad*: 2

如果还需要其他的频繁模式,只需从最大频繁模式中得到所需要的模式,因为这些模式肯定是最大频繁模式的子集. 所以得到所有的频繁模式很容易,但得到它们的支持度却不容易,因为最大模式中不包含它们的支持度信息. 目前许多算法都是在找到最大模式后再重新扫描一遍数据库来得到所有频繁模式的支持度信息,这样在重新扫描数据库时算法的性能就降低了. 针对这一问题,MCFP算法把支持度的信息都保存在 List表中,所以不用重新扫描数据库就很容易得到所有频繁模式的信息. 例如,由最大频繁模式 *bac*可以知道它的子模式有 *ba*、*bc*、*ac*,并且根据简洁性约束的性质,直接就能判断这些子模式都满足约束条件. 现在最主要的就是得到它们的支持度来判断它们是否频繁. 以计算 *bc*的支持度为例,由 List表中包含的频繁项的信息可得 *b*: 4、*c*: 2,所以 *bc*的支持度以这 2项中支持度小的一项为准,即 *bc*: 2 按照这种方法,最后所有的满足约束的频繁模式为 *ba*: 3、*bd*: 3、*ad*: 2、*bc*: 2、*be*: 2、*ae*: 2、*bac*: 2、*bae*: 2、*bad*: 2

算法 2　建立 MFT (maximal frequent pattern tree).

输入: FP-tree、最小支持度 δ、约束;

输出:最大频繁模式 MFT.

1) *M* =List表中的最后一项;

2) *M* 的条件树 M-conditional tree = *φ*;

3) For(*i* = 1; *i* < =项的个数; *i* + +) { ;

4)　if(min(*M*. *A*) ≥30) {

5)　找出含有项 *M* 的所有路径;

6)　把这些路径插入 *M* 的条件树中;

7)　把条件树中的相同项连接起来;

8)　调用 mining(*M* -conditional tree);

9)　释放 *M* 的条件树; } }

10)最大频繁模式即为 MFT的每个分支.

**算法 3　**mining(*M* -conditional tree)算法.

输入:条件树 *M* -conditional tree,最小支持度 δ;

输出:局部最大频繁模式 (LMFT)

1) LMFT = *φ*;

2) *M* 的条件树的每个分支两两相交得到局部频繁模式 *P*;

3) 按照支持度递减的顺序把 *P*插入到 MFT中;

4) 如果该局部频繁模式存在于 MFT中或是 MFT的子集;

5) 不需要插入该局部频繁模式.

3　MCFP处理动态约束

当约束变化时约束范围可能是以前范围的超集也可能是子集. 约束范围由小变大时称为松约束性变化 (relaxing change),约束范围由大变小时称为紧约束性变化 (tighting change).

3.1　处理紧约束性的动态变化

设约束由 C_{old} = min(*s A*) ≥30 变为 C_{new} = min(*s A*) ≥40,显然有 $SS_{new} \subseteq SS_{old}$,其中 SS_{new}是指新约束的范围, SS_{old}是指以前的约束范围. 所以只需在最大模式树中去掉不满足约束的项,即 *b*、*c* 删除 *b*、*c*后的最大模式树如图 5所示.

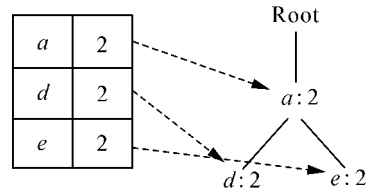

图 5　约束化后的最大模式树

Fig 5　Maximal patterns tree after changed constraint

由图 5可得满足约束的最大模式为 *ad*: 2、*ae*: 2 由最大模式可得满足约束的所有频繁模式有 *a*: 3、*d*: 3、*e*: 3

3.2　处理松约束性的动态变化

设约束由 C_{old} = min(*s A*) ≥40 变为 C_{new} = min(*s A*) ≥30,则 $SS_{new} \supseteq SS_{old}$. 这样 *b*和 *c*也满足约束条件了,只需再挖掘 *b*、*c*的条件树,最后把局部频繁模式插入到 C_{old} = min(*s A*) ≥40约束时的最大

模式树中即可.

4 实验结果和性能分析

本节通过一系列实验来比较算法 MCFP和 iCFP. 由于在算法 MCFP中每次只有一个条件树存在内存中,和 iCFP相比显然节省了大量的内存,所以对 2种算法不进行所消耗内存的比较,只从算法的运行时间上进行比较.

算法均用 C++语言编写. 运行环境: Intel Pentium Ⅳ 2.4GHz CPU, 1.0GB内存, 40GB硬盘. 实验数据都来自于文献[7],所以很具有可比性. 实验中所用的数据库包括 10万个记录,每个事务大约包含 10个项. 数据库由 IBM Almadem Research Center产生. 设最小支持度 δ为 0.01%,这 2种算法的运行时间如图 6、图 7和图 8所示.

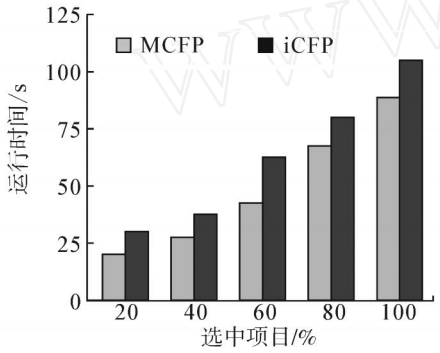

图 6 运行时间比较

Fig. 6 Comparison of running times

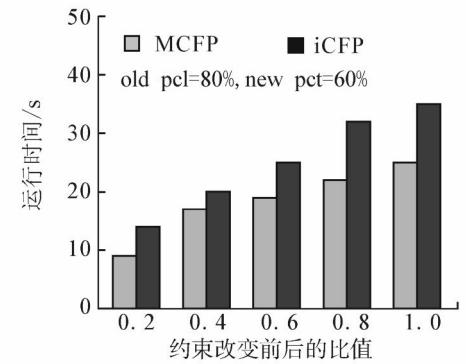

图 7 处理紧约束性变化

Fig. 7 Tighting constraint change

图 6表明 MCFP比 iCFP算法更有效. 第一, MCFP在建立 FP-tree时只需扫描遍数据库,而 iCFP需扫描 2遍, I/O代价付出大. 第二,在挖掘满足约束的频繁项时, iCFP需要不断地建立条件树,并且不释放,这样占用了大量的内存,然而, MCFP只需建立一次每个项的条件树,并且在挖掘完成后释放该树,和 iCFP相比,节省了大量的内存.

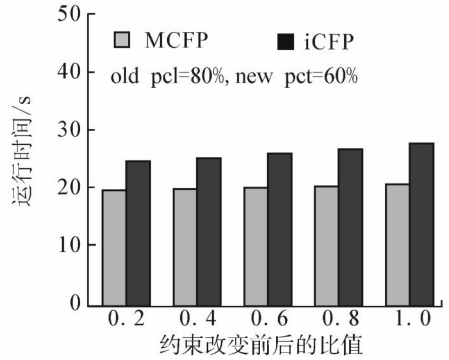

图 8 处理松约束性变化

Fig. 8 Relaxing change constraint

图 7和图 8给出了处理动态约束变化时的执行时间, MCFP的效率同样高于 iCFP. 当约束变化时, MCFP只需从最大模式中来找满足新的约束的频繁模式. 另外, MCFP能挖掘到最大模式,当只需要挖掘最大模式时, MCFP算法就节省了更多的时间. 因此, MCFP算法的性能高于 iCFP.

5 结束语

提出了一种新的交互式约束的频繁模式挖掘算法,该算法不仅能挖掘最大频繁模式,而且在此基础上能找到所有的满足约束的支持度精确的频繁模式. 该算法克服了以前的最大模式挖掘算法中不能精确知道由最大模式产生的所有频繁模式的支持度问题. 另外,该算法采用新的构建 FP-tree的方法,避免了在约束变化时重新构建 FP-tree.

参考文献:

[1] AGRAWAL R, SRIKANT R. Fast algorithms for mining association rules[C]//Proceedings 20th International Conference on VLDB. Morgan Kaufmann, 1994: 487-499.

[2] 颜跃进, 李舟军, 陈火旺. 频繁项集挖掘算法[J]. 计算机科学, 2004, 31(3): 112-114.
YAN Yuejin, LI Zhoujun, CHEN Huowang. Frequent item sets mining algorithms[J]. Computer Science, 2004, 31(3): 112-114.

[3] 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): 215-222.
YAN Yuejin, LI Zhoujun, CHEN Huowang. Efficiently mining of maximal frequent item sets based on FP-Tree[J]. Journal of Software, 2005, 16(2): 215-22.

[4] AGRAWAL R, MIELNSKI T, SWAMI A. Mining associ-

ation rules between sets of items in large database[C]// Proc of the ACM SIGMOD Conf on Management of Data. Washington, DC, 1993: 207-216.

[5] SRIKANT R, AGRAWAL R. Mining sequential patterns: generalizations and performance improvements[C]// Proc of the 5th Int'l Conf on Extending Database Technology. Berlin: Springer-Verlag, 1996: 3-17.

[6] PEI J, HAN J, PINTO H, et al. Mining sequential patterns efficiently by prefix-projected pattern growth[C]// The 17th Int'l Conf on Data Engineering. Heidelberg, Germany, 2001: 195-201.

[7] BURDICK D, CALIMLIM M, GEHRKE J. A maximal frequent itemset algorithm for transactional database[C]//Piscataway, NJ: IEEE Press, 2001: 443-452.

[8] 陆介平, 刘月波, 倪巍伟. 基于 PrefixSpan的快速交互序列模式挖掘算法[J]. 东南大学学报:自然科学版, 2005, 35(5): 692-696.
LU Jieping, LIU Yuebo, NI Weiwei. Fast interactive sequential pattern mining algorithm based on PrefixSpan[J]. Journal of Southeast University: Natural Science Edition, 2005, 35(5): 692-696.

[9] ZHU Qunxiong, LIN Xiaoyong. Mining frequent patterns with incremental updating frequent pattern tree[C]//The 6th World Congress on Intelligent Control and Automation. Dalian, 2006: 125-128.

[10] LENG Kaisong. Interactive constrained frequent-pattern mining system[C]//Proc of the Int'l Database Engineering and Applications Symposium (IDEAS'04). Los Alamitos, CA: IEEE Computer Society Press, 2004: 48-58.

作者简介:

孟彩霞,女,1966年生,副教授,主要研究方向为数据库、数据挖掘等.

第 2届智能计算技术与自动化国际会议
The Second International Conference on Intelligent Computation Technology and Automation

ICICTA 2009 aims to provide a high-level international forum for scientists, engineers, and educators to present the state of the art of intelligent computation and automation research and applications in diverse fields. The conference will feature plenary speeches given by renowned scholars and regular sessions with broad coverage. All accepted papers will appear in conference proceedings published by the IEEE Computer Society and will be indexed both EI (Compendex) and ISTP. Relevant topics include, but are not limited to:

- Advanced Computation Theory, Application and Simulation
- Control Theory, Application and Test-bed
- Automation and Simulation
- Mechanic Manufacturing System and Engineering Optimization
- Decision and Management

Prospective authors are encouraged to submit a full paper for review in PDF format. Only original papers that have not been published or submitted for publication elsewhere will be considered. The submission process is carried through ICICTA09 conference management system. The manuscripts must follow the IEEE CS format.

Web site: http://www.icicta.org.