ISSN 1000-9825, CODEN RUXUEW
Journal of Software, Vol.21, No.4, April 2010, pp.659−671
doi: 10.3724/SP.J.1001.2010.03595

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel/Fax: +86-10-62562563

快速统一挖掘超团模式和极大超团模式*

肖 波+, 张 亮, 徐前方, 蔺志青, 郭 军

(北京邮电大学 信息与通信工程学院,北京 100876)

Fast Unified Mining of Hyperclique Patterns and Maximal Hyperclique Patterns

XIAO Bo+, ZHANG Liang, XU Qian-Fang, LIN Zhi-Qing, GUO Jun

(School of Information and Communication, Beijing University of Posts and Telecommunications, Beijing 100876, China)

+ Corresponding author: E-mail: xiaobo@bupt.edu.cn

Xiao B, Zhang L, Xu QF, Lin ZQ, Guo J. Fast unified mining of hyperclique patterns and maximal hyperclique patterns. *Journal of Software*, 2010,21(4):659−671. http://www.jos.org.cn/1000-9825/3595.htm

Abstract: The hyperclique pattern is a new type of association pattern, in which items are highly affiliated with each other. The presence of an item in one transaction strongly implies the presence of every other item in the same hyperclique pattern. The maximal hyperclique pattern is a more compact representation of a group of hyperclique patterns, which is desirable for many applications. The standard algorithms mining the two kinds of patterns are different. This paper presents a fast algorithm called hybrid hyperclique pattern growth (HHCP-growth) based on FP-tree (frequent pattern tree), which unifies the mining processes of the two patterns. This algorithm adopts recursive mining method and exploits many efficient pruning strategies. Some propositions are also presented and proved to indicate the effectiveness of the strategies and the validity of the algorithm. The experimental results show that HHCP-growth is more effective than the standard hyperclique pattern and maximal hyperclique pattern mining algorithms, particularly for large-scale datasets or at low levels of support.

Key words: association rule; hyperclique pattern; maximal hyperclique pattern; data mining; FP-tree (frequent pattern tree)

摘 要: 超团模式是一种新型的关联模式,这种模式所包含的项目相互间具有很高的亲密度.超团模式中某个项目在事务中的出现很强地暗示了模式中其他项目也会相应地出现.极大超团模式是一组超团模式更加紧凑的表示,可被用于多种应用.挖掘这两种模式的标准算法是完全不同的.提出一种基于 FP-tree(frequent pattern tree)的快速挖掘算法——混合超团模式增长(hybrid hyperclique pattern growth,简称 HHCP-growth),统一了两种模式的挖掘.算法采用递归挖掘方法,并应用多种有效的剪枝策略.提出并证明几个相关命题来说明剪枝策略的有效性和算法的正确性.实验结果表明,HHCP-growth 算法相对于标准的超团模式挖掘算法和极大超团模式挖掘算法都具有更高的效率,尤其对于大数据集或在低支持度条件下更为显著.

关键词: 关联规则;超团模式;极大超团模式;数据挖掘;频繁模式树

中图法分类号: TP311 **文献标识码**: A

* Supported by the National High-Tech Research and Development Plan of China under Grant No.2007AA01Z417 (国家高技术研究发展计划(863)); the "111 Project" of China under Grant No.B08004 (高等学校学科创新引智计划)

Received 2008-08-08; Revised 2008-12-09; Accepted 2009-02-24

1 问题背景

关联模式发现旨在从大量数据中提取人们未知又潜在有用的模式,如在给定数据集中发现强相关的项目集合.关联模式发现有时也称为关联分析,目前已是数据挖掘研究中的重要内容之一,在很多领域具有重要应用,如购物篮分析[1]、告警关联分析[2,3]、气候研究、公共健康及生物信息学等.

Agrawal 等人在 1993 年首先提出从交易事务数据库发现项目间关联规则的相关问题[1].设 $I=\{i_1,i_2,\ldots,i_m\}$是 m 个不同项目的集合,给定一个事务数据库 $D=\{T_1,T_2,\ldots,T_n\}$,其中每个交易 T_i 是 I 中一组项目的集合,$T_i\subset I$.关联规则定义是一个形如 $X\rightarrow Y$ 的蕴涵式,其中 $X\subset I$,$Y\subset I$,并且 $X\cap Y=\varnothing$.如果 D 中有 t 个事务同时包含 X 和 Y,占总事务的 $s\%$,则定义关联规则 $X\rightarrow Y$ 的支持度为 $s\%$,记 $sup(X\rightarrow Y)=s\%$,t 定义为 $X\rightarrow Y$ 的支持度计数,记 $supcount(X\rightarrow Y)$.若 D 中 $c\%$的事务在包含 X 时也包含 Y,则定义关联规则 $X\rightarrow Y$ 的置信度为 $c\%$,记 $conf(X\rightarrow Y)=P(Y|X)=sup(X\cap Y)/sup(X)=c\%$.关联规则挖掘则是找到所有不小于给定最小支持度$\xi$和最小置信度$\theta$的规则.

Agrawal 等人同时也给出了基于频繁集的 Apriori 算法[1,4],该算法以递归统计为基础,以最小支持度为依据剪切生成频繁集,使用最小置信度参数评价规则的有效性.Apriori 算法及其各种改进算法大都基于频繁模式的反单调性质:若某个 k 项集在数据库中为非频繁项集,则包含该 k 项集的任意 k+1 项超集必为非频繁项集.这类方法挖掘较短的频繁模式比较有效,但对于大长度频繁项集效率变低.例如,设频繁项集长度为 50,则其频繁子集的数量约为 2^{50},即算法将产生约 2^{50} 个候选项集,显然,在目前的计算环境下,存储和查找如此多的项集是比较困难的.

为此,Han 等人提出了一种无需保存全部候选项集进行频繁模式挖掘的方法——FP-growth(frequent pattern growth)[5].该方法首先将事务数据库压缩到一棵被称为频繁模式树(frequent pattern tree,简称 FP-tree)的存储结构中,然后递归调用模式增长算法产生频繁项集.由于 FP-growth 采用深度优先遍历,其空间复杂度和时间复杂度一般均优于 Apriori 算法,从而提高了挖掘效率.

无论是 Apriori 算法还是 FP-growth 算法,都要根据给定的最小支持度产生频繁项集.然而当给定的最小支持度阈值较小时,这些方法将产生数量巨大的频繁模式,挖掘效率会降低,同时得到的频繁模式可能会包含一些相关性较弱的项目.Xiong 等人提出了 h-置信度(h-confidence)度量方法,并定义了一种新的关联模式——超团模式(hyperclique pattern)[6,7].超团模式是一类特殊的关联模式,可以认为是频繁模式的子集,模式中所包含的项目相互之间具有很强的关联性,即超团模式中某个项目在事务中的出现很强地暗示了模式的其他项目也会出现.同时文献[6]中也提出了挖掘所有超团模式的 Hyperclique Miner 算法,该算法依然基于 Apriori 算法,因此需要产生大量的候选项集,很难有效挖掘长度较大的超团模式.

如果一个超团模式的任意超集都不是超团模式,则称该超团模式为极大超团模式(maximal hyperclique pattern).由于极大超团模式的任意子模式都是超团模式,因此极大超团模式是这些子模式更加紧凑的表示.极大超团模式可被用于多种应用,如基于模式的聚类或分类.文献[8]提出了挖掘极大超团模式的 Hybrid 算法,算法首先进行广度优先遍历(breadth first search,简称 BFS),然后采用深度优先遍历(depth first search,简称 DFS),并引入等价项剪枝等策略,能够有效地挖掘极大超团模式.然而,对于存在大量低支持度项目的数据集,Hybrid 算法的挖掘效率较低.

本文以提高超团模式挖掘的效率为目标,提出一种基于 FP-tree 的快速挖掘算法——混合超团模式增长(hybrid hyperclique pattern growth,简称 HHCP-growth).该算法不仅提高了挖掘效率,而且还统一了两种模式的挖掘方法.算法采用递归挖掘,无需保存全部候选项集,除了应用传统的最小支持度剪枝以外,还引入了最大支持度剪枝、项目自剪枝及剩余项目剪枝等多种有效的剪枝策略.同时,文中还提出并证明了几个相关命题来说明剪枝策略的有效性和算法的正确性.实验结果表明,HHCP-growth 算法相对于标准的超团模式挖掘算法 Hyperclique Miner 和极大超团模式挖掘算法 Hybrid 都具有更高的效率,尤其是在大数据集或低支持度条件下更为显著.

2 相关定义

在本节中,我们将讨论频繁模式树及超团模式的相关定义.

2.1 频繁模式树(FP-tree)

在进行关联模式挖掘时,由于数据库中的事务很多,有时很难直接存放到机器内存.为此,Han 等人提出FP-tree 结构,将事务数据库进行有效的压缩[5].

定义 1(频繁模式树). 频繁模式树 FP-tree 是一种按如下定义的树结构:

(1) 它包含一个标记为 *null* 的根结点(root),一个项目前缀子树(item prefix subtrees)的集合作为根结点的孩子以及一个频繁项目头表(frequent-item header table),有时简称为头表.

(2) 项目前缀子树中的每个结点至少包含 5 个域:*item-name*,*count*,*node-link*,*children-link* 以及 *parent-link*. *item-name* 表示结点代表的项目,*count* 表示包含从根结点到本结点的路径上所有结点的事务数目.*node-link* 指向树中下一个与本结点具有相同 *item-name* 的结点,这样,树中所有具有相同 *item-name* 的结点构成一个结点链表.*children-link* 代表本结点所有的孩子.*parent-link* 指向本结点的父结点.

(3) 频繁项目头表中的每行至少存储两个域:*item-name* 和 *node-link*.*item-name* 表示该行对应的项目,*node-link* 指向树中第 1 个具有相同 *item-name* 的结点.这样,所有项目的结点链表表头都存储在频繁项目头表中.为了便于遍历,一般将频繁项目头表中的各项目按支持度降序排序.

图 1 给出了一个频繁模式树的示例.该例中共有 7 个频繁项目,存储在频繁项目头表中,表中每个项目后的数字代表了该项目在数据库中的支持度计数.树中除根结点外其他结点用 *item-name*:*count* 的形式标记.

文献[5]描述了构造 FP-tree 的方法,如算法 1 所示.

算法 1. *CreateFPtree*().

输入:事务数据库 *DB*,最小支持度计数ξ.

输出:频繁模式树.

符号: t_i,数据库 *DB* 中的每个事务;*F*,项目集合;r_i,事务 t_i 中按支持度排序的频繁项目集合.

步骤:(1) 遍历 *DB* 中每个事务 t_i 包含的项目,得到项目集合 *F* 及每个项目的支持度计数;

(2) 通过ξ得到所有频繁项,按支持度降序插入到频繁模式树的频繁项目头表中;

(3) *T*=*createRoot*(); //建立频繁模式树的根结点 *T*,并标记为 *null*

(4) for *DB* 中的每个事务 t_i do {

(5) 对 t_i 中的所有频繁项按支持度降序排序,记为 r_i;

(6) if $r_i \neq \varnothing$ then call *insert_tree* (r_i,*T*);}

(7) return *T*; //输出以 *T* 为根结点的频繁模式树

算法 1 中,步骤 2 应用了传统的最小支持度剪枝策略进行剪枝.步骤 6 调用 *insert_tree* 函数,将频繁项集 r_i 添加到以 *T* 作为根结点的树中,算法 2 给出其递归处理过程,其中第 1 个参数 *r* 的格式定义为[*p*|*P*],*p* 是 *r* 的第 1 个项目,*P* 是 *r* 中的剩余项目.

算法 2. *insert_tree*().

输入:项集 *r*(定义为[*p*|*P*]),结点 *Q*.

符号:*n*,树中结点 *Q* 的某个孩子结点.

步骤:(1) if $\exists n$ and *n.item-name*=*p* then *n.count*=*n.count*+1;

(2) else {建立 *Q* 的新孩子结点 *n*,令 *n.item-name*=*p*, *n.parent-link*=*T*, *n.count*=1;

(3) 将 *n* 插入到 *item-name* 为 *p* 的结点链表中;}

(4) if $P \neq \varnothing$ then call *insert_tree* (*P*,*n*);}

例 1:给定包含 8 个事务的数据库,如表 1 中的前两列所示.设最小支持度计数ξ=2,则算法 1 中步骤 1 得到所有项集 *F* 如表 2 所示.步骤 2 去掉非频繁项 *g*,并将剩余项目排序,如图 1 中左侧的频繁项目头表所示.步骤 5 中,

对于每个事务 t_i,对其频繁项目排序得到 r_i,如表 1 第 3 列所示.最终算法产生的 FP-tree 如图 1 所示.

显然,对于挖掘频繁模式来说,FP-tree 包含了事务数据库的所有信息,其高度不大于数据库中各事务的最大长度.FP-tree 中从根结点到某个叶子结点的路径中,各节点的 *count* 值递减,且结点对应项目的支持度也递减.

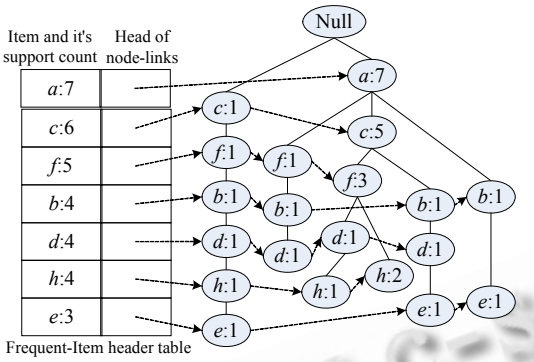


Fig.1 FP-tree of Example 1

图 1 例 1 生成的 FP-tree

Table 1 List of items in database

表 1 数据库中的项目

Tid	List of item id	Ordered frequent items
1	*b c d e f h*	*c f b d h e*
2	*a b d f*	*a f b d*
3	*a c d f h*	*a c f d h*
4	*a c f g h*	*a c f h*
5	*a b c d e*	*a c b d e*
6	*a b e*	*a b e*
7	*a c*	*a c*
8	*a c f h*	*a c f h*

Table 2 Support count of each item in database

表 2 数据库中的各项目支持度计数

Item	*a*	*c*	*f*	*b*	*d*	*h*	*e*	*g*
Support	7	6	5	4	4	4	3	1

2.2 超团模式

定义 2(*h*-置信度). 设项集 $P=\{a_1,a_2,\ldots,a_m\}(m\geq 2)$,定义其 *h*-置信度(*h*-confidence)为 $\min\{conf(a_1\rightarrow a_2,\ldots,a_m), conf(a_2\rightarrow a_1,a_3,\ldots,a_m),\ldots,conf(a_m\rightarrow a_1,\ldots,a_{m-1})\}$,记为 $hconf(P)$.其中 *conf* 为传统关联规则的置信度.

h-置信度显然具有反单调特性[6,7].$hconf(P)$可采用公式(1)简化计算,公式中只需 P 的支持度计数及 P 所包含项目中的最大支持度计数.

$$hconf(P)=\min\left\{\frac{sup(P)}{sup(a_1)},\frac{sup(P)}{sup(a_2)},\ldots,\frac{sup(P)}{sup(a_m)}\right\}=\frac{sup(P)}{\max\limits_{k=1\ldots m}\{sup(a_k)\}}=\frac{supcount(P)}{\max\limits_{k=1\ldots m}\{supcount(a_k)\}} \tag{1}$$

例 2:给定两个项目 *a* 和 *b*,其支持度计数分别为 *a*:5,*b*:4,若 *ab* 同时出现的次数为 3,则 $hconf(ab)=0.6$.

定义 3(超团模式). 对于含有 $m(m\geq 2)$个项目的项集 P,若 $sup(P)\geq\xi$且 $hconf(P)\geq\theta$,则称 P 是超团模式(hyperclique pattern).*m* 称为 P 的长度,记 $m=|P|$.其中ξ是给定的最小支持度阈值,θ是给定的最小 *h*-置信度阈值.当设置θ为 0 时,超团模式变为频繁模式.

超团模式是频繁模式的一个子集,由于利用 *h*-置信度度量参数进行有效剪枝,因此在挖掘超团模式时可以设置较低的支持度阈值.对于项目支持度分布不均匀的杂凑数据集(skewed data),挖掘超团模式更有意义[6].命题 1 分析了超团模式中各项目支持度的关系.

命题 1. 设超团模式 P 满足 *h*-置信度最小阈值θ,若 P 中含有项目 a_i 和 a_j,其支持度计数分别为 $supcount(a_i)$和 $supcount(a_j)$,并且 $supcount(a_j)\geq supcount(a_i)$,则 $supcount(a_j)$的上限为 $\left\lfloor\frac{supcount(a_i)}{\theta}\right\rfloor$.

证明:显然,$supcount(P)\leq supcount(a_i)$.又根据超团模式定义及公式(1),$\frac{supcount(P)}{\max\limits_{k=1\ldots m}\{supcount(a_k)\}}\geq\theta$,因此,

$$supcount(a_j)\leq\max_{k=1\ldots m}\{supcount(a_k)\}\leq\frac{supcount(P)}{\theta}\leq\frac{supcount(a_i)}{\theta},$$

又支持度计数必为正整数,因此,$supcount(a_j)$的上限为 $\left\lfloor\frac{supcount(a_i)}{\theta}\right\rfloor$. □

命题 1 可被用于搜索空间的剪枝,例如设θ=0.6,项目 *a* 的支持度计数为 3,则可与 *a* 构成超团模式的项目的最大支持度计数为 5.所有支持度计数大于 5 的项目不可能与 *a* 构成超团模式,因此可被剪枝.这种策略在搜索项目时不再考虑超过最大支持度的项目,我们称其为最大支持度剪枝策略.

定义 4(极大超团模式). 对于一个含有 m 个项目的超团模式 P,若其任意超集都不再是超团模式,则称 P 为极大超团模式(maximal hyperclique pattern).

定义 5(最大超团模式). 挖掘事务数据集得到所有超团模式,将长度最大的超团模式定义为最大超团模式(maximum hyperclique pattern).

例 3:设数据集包含 4 个项目 a,b,c 和 d,若挖掘得到 6 个超团模式,分别为 ab,ac,ad,bc,cd,abc,则极大超团模式分别为 ad,cd,abc,而最大超团模式为 abc.

显然,极大超团模式包含最大超团模式.一个 m 项的极大超团模式共包含 2^m-m-2 个长度在 2 到 $m-1$ 之间的子超团模式.极大超团模式是所有子超团模式更为紧凑的表示,因此相对超团模式和最大超团模式,挖掘极大超团模式更有意义,如基于模式的聚类使用极大超团模式很容易产生聚类结果.

3 基于 FP-tree 挖掘超团模式和极大超团模式

FP-tree 几乎包含了事务数据库的所有频繁项目信息,因此基于 FP-tree 可以挖掘超团模式及极大超团模式.为了便于分析挖掘过程的可行性和准确性,我们提出一些相关命题和概念.

3.1 挖掘超团模式

命题 2. 对于任意频繁项 a_i,所有包含 a_i 的超团模式都可以通过 a_i 在 FP-tree 中的结点链表获得.

证明:显然,通过 a_i 在 FP-tree 中的结点链表可以得到所有包含项目 a_i 的事务,而所有包含 a_i 的超团模式必包含在这些事务中,因此所有包含 a_i 的超团模式都可以通过 a_i 在 FP-tree 中的结点链表获得. □

命题 2 揭示了利用构造好的 FP-tree,通过依次遍历 a_i 的结点链表,就可以得到 a_i 的所有模式信息.例 4 给出了相关示例,详细说明挖掘超团模式的过程.

例 4:设定最小 h-置信度阈值 θ=0.6.对于例 1 构造的 FP-tree,根据命题 2,可以利用每个频繁项目在 FP-tree 中的结点链表获得相应的超团模式.我们将自下而上顺序处理头表中的每个项目.

如图 1 所示,头表中最后一个项目为 e,其支持度计数为 3.根据命题 1,可与 e 构成超团模式的项目支持度计数不超过 5,因此这些项目只可能是 b,d,h 和 f,而 c 和 a 通过最大支持度剪枝策略被剪枝.由此与 e 构成超团模式的项目只可能包含在 e 的 3 条路径上:(f,b,d,h,e),(b,d,e)及(b,e),如图 2 所示.

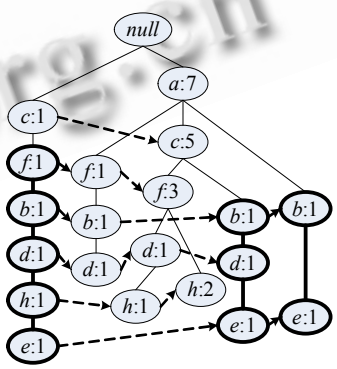


Fig.2 Prefix paths of e

图 2 项目 e 的前缀路径

第 1 条路径中 e 的计数为 1,说明(f,b,d,h,e)在数据库中同时出现 1 次,在搜索与 e 同时出现的项目时,可只使用 e 的前缀路径(f:1,b:1,d:1,h:1),路径中每个项目后面的数字代表与 e 同时出现的次数.同样方法可以得到后两条前缀路径.这样,e 的 3 条前缀路径"{(f:1,b:1,d:1,h:1),(b:1,d:1),(b:1)}"形成了 e 的子模式基,我们定义为 e 的条件模式基(即 e 存在条件下的子模式基),每条前缀路径是条件模式基的一个子模式.显然,该方法并没有像 FP-growth 算法一样遍历到每条路径的根结点,而是采用最大支持度剪枝策略,发现不符合最大支持度条件的结点就结束遍历,这样遍历的结点数量大为减少,从而提高了算法的处理速度.

为了区分项目在总事务数据库中的支持度,将条件模式基中的项目 a_i 的支持度计数称为 a_i 的相对支持度计数(relative support count),记为 $a_i.relative\text{-}supcount$.将项目(或模式)在总事务数据库(即 FP-tree)中的支持度称为绝对支持度计数(absolute support count).显然,在不同的条件模式基中,各项目的相对支持度可能有所不同.

统计 e 的条件模式基中各项目的相对支持度计数,分别为 b:3,d:2,f:1,h:1,意味着相应模式的绝对支持度计数分别为 be:3,de:2,fe:1,he:1.命题 3 给出相关证明.

命题 3. 设项目 a_i 的条件模式基中存在项目 a_j,且 a_i 与 a_j 构成模式 P,则有 $supcount(P)=a_j.relative\text{-}supcount$.

证明:项目 a_j 在 a_i 的条件模式基所有子模式中的支持度计数为 $a_j.relative\text{-}supcount$,因此 a_j 和 a_i 共同出现的

次数为 $a_j.relative\text{-}supcount$.又 a_i 与 a_j 构成模式 P,$|P|=2$,可知 a_i 和 a_j 共同出现的次数即为 P 的绝对支持度计数 $supcount(P)$.因此有 $supcount(P)=a_j.relative\text{-}supcount$. □

通过命题3的结论,可以将项目的相对支持度转化为相应模式的绝对支持度.分析 e 的条件模式基中各个项目,f 和 h 的支持度计数为1,表示 fe 和 he 都只出现1次,通过最小支持度剪枝策略可被删除.对于剩余项目 b 和 d,也可能由于最小 h-置信度的约束而被删除,命题4给出这种剪枝策略的依据.

命题 4. 令最小 h-置信度阈值为 θ,通过项目 a_i 的条件模式基得到超团模式 $P(|P|\geq 2)$,设 P 包含项目 a_j,则有:

$$a_j.relative\text{-}supcount\geq supcount(a_j)\times\theta \tag{2}$$

证明:令 $m=|P|$.由于 P 通过项目 a_i 的条件模式基得到,$m\geq 2$,则可能有部分条件模式基包含项目 a_j 而不包含 P,即有 $a_j.relative\text{-}supcount\geq supcount(P)$.

又根据公式(1),$hconf(P)=\dfrac{supcount(P)}{\max\limits_{k=1\ldots m}\{supcount(a_k)\}}\geq\theta$,有 $supcount(P)\geq\max\limits_{k=1\ldots m}\{supcount(a_k)\}\times\theta\geq supcount(a_j)\times\theta$.

因此,$a_j.relative\text{-}supcount\geq supcount(a_j)\times\theta$.命题得证. □

命题4指出,项目 a_j 满足不等式(2)是构成超团模式的必要条件,因此可将不满足该条件的项目剪枝.这种利用项目的绝对支持度计数和相对支持度计数不满足不等式(2)而将其剪枝的策略,我们称其为项目自剪枝策略.

分析剩余项目 b 和 d,项目 b 的支持度计数为3,即 be 同时发生的次数为3,b 在整个事务数据库中的支持度计数为4,因此满足构成超团模式的必要条件.项目 d 的支持度计数为2,即 de 同时发生的次数为2,d 在整个事务数据库中的支持度计数为4,不满足不等式(2),通过项目自剪枝策略 d 被剪枝.

命题 5. 对于项目 a_i 的条件模式基中的所有项目,经过最小支持度剪枝和项目自剪枝后,剩余的每个项目都可与 a_i 构成超团模式.

证明:设 a_j 是经两种策略剪枝后剩余的任意项目,a_j 与 a_i 构成模式 P,则只需证明 P 为超团模式.

(1) 由于 a_j 没有因最小支持度剪枝,则 $a_j.relative\text{-}supcount\geq\xi$.又 a_j 与 a_i 构成模式 P,$|P|=2$,由命题3可得,

$$supcount(P)=a_j.relative\text{-}supcount\geq\xi,$$

即 P 满足最小支持度条件.

(2) 由FP-tree的性质可知,项目 a_i 所有前缀路径中各项目的绝对支持度都不小于 a_i 的绝对支持度.因此有

$$supcount(a_j)\geq supcount(a_i).$$

分析 P 的 h-置信度,有

$$hconf(P)=\frac{supcount(P)}{\max\limits_{k=i,j}\{supcount(a_k)\}}=\frac{supcount(P)}{supcount(a_j)}=\frac{a_j.relative\text{-}supcount}{supcount(a_j)}\geq\theta.$$

即 P 满足最小 h-置信度条件.

通过上面的(1)(2),可得模式 P 为超团模式. □

将项目 a_i 的条件模式基当作事务数据库,对其中的项目进行最小支持度剪枝和项目自剪枝,将每个子模式中的剩余项目看作一个事务,可根据算法1的思想构造一棵类似于FP-tree的新树,我们称其为项目的条件超团模式树(conditional hyperclique pattern tree,简称CHCP-tree).根据命题5,该树中的任意项目都可与 a_i 构成超团模式.这样,得到 e 的CHCP-tree非常简单:{(b:3)},最终由 e 得到的超团模式为(be:3),并且 $h\text{-}conf(be)$=0.75.

继续分析例4中的项目 h,其支持度为4,可根据命题1得到与其构成超团模式的项目最大支持度为6,由此可得 h 的条件模式基为{(c:1,f:1,b:1,d:1),(c:1,f:1,d:1),(c:2,f:2)},如图3所示.虽然 e 也曾和 h 同时出现,但包含 e 的所有超团模式已经挖掘得到,因此无需再考虑 e.项目 h 的条件模式基中各个项目的支持度计数分别为 c:4,f:4,d:2,b:1,其中 b 通过最小支持度剪枝策略被剪除,d 通过项目自剪枝策略被剪除,因此 h 的CHCP-tree为{(c:4,f:4)},该树头表中各项目的次序依然与原FP-tree头表的次序相同.接下来递归调用挖掘算法继续对 h 的CHCP-tree进行挖掘.

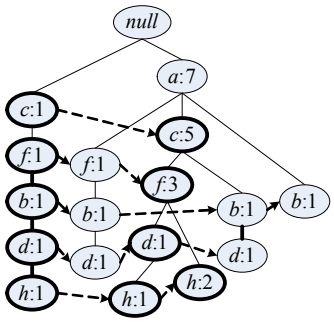


Fig.3 Prefix paths of *h*
图 3 项目 *h* 的前缀路径

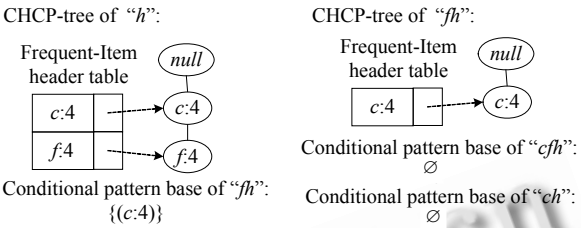


Fig.4 CHCP-tree built for *h*
图 4 项目 *h* 生成的条件超团模式树

图 4 给出挖掘项目 *h* 的条件超团模式树{(*c*:4,*f*:4)}的过程.该树包含 2 个项目 *c* 和 *f*,按照自下而上顺序处理.首先根据命题 5,由 *f* 得到超团模式(*fh*:4),然后递归计算得到 *fh* 的条件模式基为{(*c*:4)},产生的 CHCP-tree 为{(*c*:4)},最后可得超团模式(*cfh*:4),递归结束.处理 *c* 得到超团模式(*ch*:4),由于 *ch* 的条件模式基为∅,递归结束.这样最终得到包含 *h* 的超团模式集合为{(*fh*:4),(*cfh*:4),(*ch*:4)}.

依此类推,可得例 4 中每个项目的条件模式基和条件超团模式树,进而得到相应的超团模式,见表 3.

Table 3 Mining of all hyperclique patterns by creating conditional pattern bases

表 3 建立条件模式基挖掘所有超团模式

Item	Conditional pattern base	CHCP-Tree	Hyperclique patterns
E	{(*f*:1,*b*:1,*d*:1,*h*:1),(*b*:1,*d*:1),(*b*:1)}	{(*b*:3)}	*be*:3
H	{(*c*:1,*f*:1,*b*:1,*d*:1),(*c*:1,*f*:1,*d*:1),(*c*:2,*f*:2)}	{(*c*:4,*f*:4)}	*fh*:4,*cfh*:4,*ch*:4
D	{(*c*:1,*f*:1,*b*:1),(*f*:1,*b*:1), (*c*:1,*f*:1),(*c*:1,*b*:1)}	{(*f*:3,*b*:2),(*b*:1)}	*bd*:3,*fd*:3
B	{(*c*:1,*f*:1),(*f*:1),(*c*:1)}	∅	∅
F	{(*c*:1),(*a*:1),(*a*:3,*c*:3)}	{(*c*:4)}	*cf*:4
C	{(*a*:5)}	{(*a*:5)}	*ac*:5
A	∅	∅	∅

根据以上分析,算法 3 首先给出通过产生条件模式基来生成条件超团模式树的过程.

算法 3. *CreateCHCPtree*().

输入:频繁模式树或条件超团模式树 *Tree*,项目β,最小支持度计数阈值ξ,最小 *h*-置信度阈值θ.

输出:项目β的条件超团模式树.

符号:P_i,β的第 *i* 条路径,形式为$[b_i|p_i]$,b_i为包含β的结点,p_i为前缀路径;cpb_i,β的条件模式基中第 *i* 个子模式;f_{ij},p_i的每个结点;$\beta.supcount$,β在 *Tree* 中的支持度计数;*F*,项目集合.

步骤:(1) for 项目β的每条路径 P_i(即$[b_i|p_i]$) do {

(2) cpb_i=∅;

(3) for p_i 自下而上的每个结点 f_{ij} do {

(4) if $\beta.supcount<supcount(f_{ij}.item\text{-}name)\times\theta$ then break; //应用最大支持度剪枝策略

(5) $cpb_i=\{f_{ij}.item\text{-}name\}\cup cpb_i$;} //沿前缀路径构造子模式

(6) 设置 cpb_i 中每个项目的支持度计数为 $b_i.count$;}

(7) 遍历所有 cpb_i 包含的项目,得到项目集合 *F* 及每个项目的支持度计数;

(8) 应用最小支持度剪枝策略和项目自剪枝策略,剪除 *F* 中不符合条件的项目;

(9) 将 *F* 中的剩余项目按支持度降序插入到频繁模式树的频繁项目头表中;

(10) *T*=*createRoot*();

(11) for 条件模式基中的每个子模式 cpb_i do {

(12) 剪除 cpb_i 中在频繁项目头表不存在的项目,剩余项目记作 r_i;

(13) if $r_i \neq \varnothing$ then call *insert_tree* (r_i,T);}

(14) return T;

算法 3 中的步骤 1~步骤 6 描述了如何通过前缀路径产生条件模式基,步骤 7~步骤 13 描述了生成条件超团模式树的过程,不难发现该过程与算法 1 基本相同,只是在步骤 8 剪除项目时引入项目自剪枝策略,在步骤 12 中,由于各个子模式中的项目支持度已经有序,因此不需要对 r_i 中的项目排序.算法最终得到的 CHCP-tree 中,每条路径自上而下各结点对应项目的支持度计数同样由大到小排序.

根据命题 5,项目 a_i 的条件超团模式树中的所有项目都可以与 a_i 构成超团模式,进而可以对每个新生成的超团模式构造相应的条件模式基和条件超团模式树,递归挖掘直至得到所有的超团模式.在每次递归挖掘过程中,新产生的超团模式在原超团模式上增长了一个项目,因此称这种挖掘算法为超团模式增长(hyperclique pattern growth,简称 HCP-growth).算法 4 描述了其挖掘过程.在初始调用该算法时,参数 *Tree* 为事务数据库对应的 FP-tree,参数 $\alpha=\varnothing$.在递归过程中,参数 *Tree* 为模式 α 的条件超团模式树.

算法 4. *HCP-growth*().

输入:频繁模式树或条件超团模式树 *Tree*,当前已得到的超团模式 α.

输出:所有超团模式构成的集合 S_{hcp}.

符号:λ,*Tree* 中的项目 β_i 与 α 构成的超团模式;$Tree_\lambda$,λ 的条件超团模式树.

步骤:(1) for *Tree* 的头表中自下而上的每个项目 β_i do{

(2) $\lambda=\beta_i \cup \alpha$; //产生超团模式 λ,其支持度为 β_i.*relative-supcount*

(3) if $|\lambda|\geq 2$ then $S_{hcp}=\{\lambda\}\cup S_{hcp}$;

(4) $Tree_\lambda$=*CreateCHCPtree*(*Tree*,β_i); //调用算法 3 产生条件超团模式树

(5) if $Tree_\lambda \neq \varnothing$ then call *HCP-growth*($Tree_\lambda$,λ);}

分析挖掘结果的正确性和完整性.首先,在初始调用算法 4 时,步骤 2 得到的模式只有一个频繁项目,步骤 4 得到该模式的 CHCP-tree.若树存在,则递归调用 HCP-growth 算法,此时的参数 *Tree* 为新产生的 CHCP-tree,参数 α 为新生成的模式.根据命题 5,项目与其 CHCP-tree 中各个项目均能构成超团模式,因此步骤 2 在每次循环中得到的模式 λ 必为超团模式,在步骤 4 中得到该模式的 CHCP-tree.若树存在,则继续递归,得到不断增长的超团模式,直到相应的 CHCP-tree 为 $\varnothing$.因此得到的所有超团模式是正确的.另一方面,命题 2 指出,包含项目 a_i 的超团模式都可以通过 a_i 在 FP-tree 中的结点链表获得,分析例 4 的挖掘过程,当挖掘较大支持度项目的超团模式时可忽略小支持度项目.由此,在挖掘 a_i 的超团模式时,只需遍历 a_i 各个结点的前缀路径,得到其条件模式基.根据命题 5,经最大支持度剪枝策略和项目自剪枝策略后去除冗余项目,最终与 a_i 构成超团模式的项目会全部包含在 a_i 的 CHCP-tree 中.这种处理在递归过程中同样有效,因此最终挖掘得到的超团模式是完整的.由以上讨论可知,通过算法 4 能够正确得到事务数据库所包含的所有超团模式.

3.2 统一挖掘超团模式和极大超团模式

分析算法 4,每次新产生的超团模式都会包含递归前的超团模式,而只有当递归终止(即产生的条件超团模式树为 $\varnothing$)时,产生的超团模式 P 才有可能是极大超团模式,在这里我们定义 P 为局部极大超团模式(local maximal hyperclique pattern).若先前产生的各个极大超团模式中存在 P 的超集,则 P 不是极大超团模式;反之,P 一定是极大超团模式.

例如表 3 中处理项目 h 时,得到包含 h 的极大超团模式 cfh:4,而处理项目 f 时,又得到局部极大超团模式 cf:4,由于 cf 的超集 cfh 已经得到,显然 cf 不是极大超团模式.

在挖掘某项目的 CHCP-tree 时,如果得到的超团模式中已经包含了 CHCP-tree 头表中所有未处理的项目,则无需再对未处理的项目进行处理,我们称这种策略为剩余项目剪枝策略.命题 6 给出相关证明.

命题 6. 设频繁模式树或条件超团模式树 *tree* 的头表中共包含 n 个项目,按支持度升序依次为 $a_1,a_2,\ldots,a_n$,

顺序处理每个项目,若处理 a_k 时得到的超团模式 P 包含了所有未处理的项目 $a_{k+1},\ldots,a_n$,则 P 必为局部极大超团模式,且 P 包含了处理 $a_{k+1},\ldots,a_n$ 时得到的所有超团模式.

证明:设在挖掘 *tree* 之前已得到模式 Q,在处理 a_k 时得到的超团模式 $P=Q\cup\{a_k,a_{k+1},\ldots,a_n\}$.由于 P 是经过递归处理最后一个项目 a_n 时得到的,因此 P 为局部极大超团模式.在处理 $a_{k+i}(1\le i\le n-k)$时,递归过程中只可能遍历 a_{k+i} 后面的项目,因此最后得到的超团模式除了包含 Q 以外,还可能包含 $\{a_{k+i},a_{k+i+1},\ldots,a_n\}$的若干子集,即这些超团模式必为 $Q\cup\{a_k,a_{k+1},\ldots,a_n\}$的子集,因此超团模式 P 必将包含处理 $a_{k+1},\ldots,a_n$ 时得到的所有超团模式. □

显然当新产生的局部极大超团模式只有一个时才有可能应用剩余项目剪枝策略.例如,例 4 中当挖掘 h 的条件模式树时,需要处理头表中的 f 和 c,处理 f 时得到模式 *cfh*,已包含未处理项目 c,因此无需再处理 c.

对算法 4 进行修改,我们提出挖掘超团模式和极大超团模式的统一方法——混合超团模式增长(HHCP-growth)算法,如算法 5 所示.算法中引入了目标参数 *target*,其值为 0 时表示挖掘所有超团模式,值为 1 时表示挖掘所有极大超团模式.

算法 5. *HHCP-growth*().

输入:频繁模式树或条件超团模式树 *Tree*,挖掘目标 *target*,当前已得到的超团模式 α.

输出:所有超团模式 S_{hcp} 或所有极大超团模式 S_{mhcp}.

符号:λ,β_i 与 α 构成的超团模式;$Tree_\lambda$,λ 的条件超团模式树;N_λ,本次递归完成时新产生的所有局部极大超团模式构成的集合;λ',N_λ 中的某个超团模式;φ,*Tree* 的头表中剩余未处理项目集合;S_λ,λ 的任意超集.

步骤:(1) for *Tree* 的头表中自下而上的每个项目 β_i do{

(2) $\lambda=\beta_i\cup\alpha$;

(3) if *target*=0 and $|\lambda|\ge 2$ then $S_{hcp}=\{\lambda\}\cup S_{hcp}$;

(4) $Tree_\lambda=CreateCHCPtree(Tree,\beta_i)$;

(5) if $Tree_\lambda\ne\varnothing$ then {

(6) call *HHCP-growth*($Tree_\lambda,\lambda$); //递归处理,完成后新产生的所有局部极大超团模式构成集合 N_λ

(7) if *target*=1 and $|N_\lambda|$=1 and $\lambda'\supseteq\varphi$ then break;} //应用剩余项目剪枝策略

(8) else if *target*=1 then { //此时 λ 为局部极大超团模式

(9) if $S_\lambda\notin S_{mhcp}$ then $S_{mhcp}=\{\lambda\}\cup S_{mhcp}$;}}

例 5:设定最小 h-置信度阈值 θ=0.6.对于例 1 构造的 FP-tree 挖掘所有极大超团模式,挖掘结果见表 4,其中第 4 列给出算法中步骤 8 产生的局部极大超团模式,第 5 列给出最终的挖掘结果.

Table 4 Mining of all maximal hyperclique patterns by creating conditional pattern bases

表 4 建立条件模式基挖掘所有极大超团模式

Item	Conditional pattern base	CHCP-Tree	Local maximal hyperclique patterns	Maximal hyperclique patterns
e	{(*f*:1,*b*:1,*d*:1,*h*:1),(*b*:1,*d*:1),(*b*:1)}	{(*b*:3)}	*be*:3	*be*:3
h	{(*c*:1,*f*:1,*b*:1,*d*:1),(*c*:1,*f*:1,*d*:1),(*c*:2,*f*:2)}	{(*c*:4,*f*:4)}	*cfh*:4	*cfh*:4
d	{(*c*:1,*f*:1,*b*:1),(*f*:1,*b*:1),(*c*:1,*f*:1),(*c*:1,*b*:1)}	{(*f*:3,*b*:2),(*b*:1)}	*bd*:3,*fd*:3	*bd*:3,*fd*:3
b	{(*c*:1,*f*:1),(*f*:1),(*c*:1)}	∅	∅	∅
f	{(*c*:1),(*a*:1),(*a*:3,*c*:3)}	{(*c*:4)}	*cf*:4	∅
c	{(*a*:5)}	{(*a*:5)}	*ac*:5	*ac*:5
a	∅	∅	∅	∅

3.3 HHCP-Growth算法分析

首先分析 HHCP-growth 算法挖掘结果的正确性和完整性.挖掘所有超团模式时,HHCP-growth 算法退化为算法 4,第 3.1 节已对其进行分析.当挖掘极大超团模式时,步骤 2 产生的 λ 已经是超团模式,根据命题 6,剩余项目剪枝策略去除的是某个超团模式的所有真子集,而极大超团模式不会被剪枝,因此挖掘结果是完整的.步骤 9 中,只有在不存在 λ 的超集时才会保留 λ,此时 λ 必为极大超团模式,因此挖掘结果是正确的.由此,算法 5 可得事务数据库所包含的所有超团模式和极大超团模式.

接下来对 HHCP-growth 算法的性能进行分析.文献[5]指出,算法 1 通过最小支持度剪枝策略得到所有频繁

项,使数据库每个事务中的频繁项可以有效地压缩到 FP-tree 中,从而节省存储空间,提高遍历效率.HHCP-growth 算法只需遍历一次 FP-tree,即可得到各个项目的 CHCP-tree.由于 CHCP-tree 是在项目的条件模式基上建立的,并且采用了 3 种有效的剪枝策略:最小支持度剪枝策略、最大支持度剪枝策略和项目自剪枝策略,因此 CHCP-tree 的大小远小于项目的条件模式基.而条件模式基只是 FP-tree 中项目前缀路径的一部分,因此 CHCP-tree 更是远小于 FP-tree.每进入下一层递归,新产生的 CHCP-tree 会更小,并且递归的最大深度不会超过 FP-tree 的高度,因此算法具有较高的挖掘效率.

在挖掘过程中,虽然 HHCP-growth 算法需要多次产生 CHCP-tree,但其处理过程类似于深度优先搜索(DFS)算法,相对于其他需要保存大量候选项集的标准算法来说,其时间花费大为减少.在挖掘极大超团模式时,算法又引入剩余项目剪枝策略,相对于挖掘超团模式,此过程减少了遍历和递归的次数.在步骤 9 中,对每个新产生的局部极大超团模式λ,需要判别在极大超团模式集合中是否存在其超集,这个过程经过如下简单处理可以大大提升判别效率:将先前得到的极大超团模式集合映射为一棵类似于 FP-tree 的树,树的每个叶子结点到根结点的路径便是一个极大超团模式.当需要判别λ是否存在超集时,只需判别树中是否存在相应的路径包含λ.若不存在其超集,则λ必为极大超团模式,并采用类似于算法 1 的方法,将λ加入到这棵树中.

4 实验结果分析

4.1 数据集及实验环境描述

实验中使用了两个来自真实世界的数据集,这些数据集也用来评估模式挖掘算法的基准数据集.其中 Alarm 数据集[2](http://www.pris.net.cn/alarm/alarm.htm)是一个通信网中的告警数据集,其采集过程是每隔 1 分钟开始一次告警数据的采集,每次采集持续 2 分钟.该数据集共含有 8 261 个告警项目及 22 255 个事务,每个事务是每次采集得到的 2 分钟内发生的告警种类集合.该数据集最小事务长度为 1,最大事务长度为 320,平均事务长度为 29.5.Retail 数据集(http://fimi.cs.helsinki.fi/data/)是比利时某个零售超市的销售记录,该数据集包含了 16 470 个商品项目及 88 162 个交易事务,其中最小事务长度为 1,最大事务长度为 76,平均事务长度为 10.3.

文献[6]提出了 Hyperclique Miner 算法,该算法直接将 h-置信度剪枝加入 Apriori 算法,不断迭代产生候选项集,挖掘得到所有超团模式.实验中算法的实现按照同样方法将 h-置信度剪枝加入 Apriori 源代码(http://www.borgelt.net/apriori.html).文献[8]提出了 Hybrid 算法,该算法采用 bitmap 结构存储数据库[9],挖掘时首先进行广度优先遍历(BFS),然后采用深度优先遍历(DFS),并引入等价项剪枝等策略,可以挖掘得到所有极大超团模式.实验中将 HHCP-growth 算法与以上两种算法进行性能比较.

测试机器 CPU 为 PIV 3.20GHz,内存为 1GB,使用 Windows XP 操作系统.所有的实验程序使用 VC 6.0 编译.

4.2 HHCP-Growth算法挖掘结果分析

图 5 给出 HHCP-growth 算法挖掘 Alarm 数据集得到的超团模式数量.不难发现,最小支持度 minsup 或最小 h-置信度 minhconf 设置越小,得到的超团模式越多,这是因为一个 n 项超团模式存在 2^n-n-2 个长度大于 1 的子超团模式,当 n 较大时得到的超团模式非常多.在 minsup=0.001 时可得到几千个超团模式.

图 6 给出挖掘 Alarm 数据集得到的极大超团模式数量.对比图 5,数量明显比超团模式要少,这是因为每得到一个极大超团模式都会去掉所有子集构成的超团模式.从图中可以发现,minsup 和 minhconf 设置越小,一般来说得到的极大超团模式越多,但也有例外.例如,minhconf 设置为 0.6 时,minsup=0.1 得到极大超团模式反而比 minsup=0.5 时要少,这是因为 minsup=0.1 时得到长度较大的极大超团模式,而 minsup=0.5 时,这样的模式不满足条件,因此得到了其部分子集构成的极大超团模式,导致此时的模式数量更多一些.

图 7 给出挖掘的超团模式的最大长度(即最大超团模式的长度).可以看到,在 minsup 为 0.0001 时,超团模式最大长度超过 100,表明很多发生次数较低的告警总是一起发生.随着 minsup 和 minhconf 的增大,这些告警被剪除,从而使得到的模式最大长度大为减小.如当 minsup≥0.001 时,最大长度在 10 以下.

图 8 给出了所有极大超团模式的平均长度.在相同 minsup 下,基本上 minhconf 越小,得到的超团模式平均

长度越大.这是由于较小的 minhconf 对超团模式的 *h*-置信度限制较少,容易得到较长模式,使得所有极大超团模式的平均长度较大.而对于不同的 minhconf,平均长度随 minsup 增大的变化趋势不同.例如,在 minhconf≥0.8 时,平均长度基本上随着 minsup 的增大而减少,而当 minhconf≤0.7 时,平均长度基本上先增大后变小.这是因为在 minhconf 和 minsup 都较小时,得到的极大超团模式比较多,而这些模式中较短的极大超团模式又占多数,随着 minsup 的增大,较短的极大超团模式在所有极大超团模式中的比例减少,因此模式的平均长度增大,当 minsup 进一步增大时,较长的极大超团模式越来越少,从而使平均长度变小.当 minconf 较大时,得到的极大超团模式不多.随着 minsup 的增大,极大超团模式的总数变少,同时较长模式的相对比例也变少,模式的平均长度越来越小.

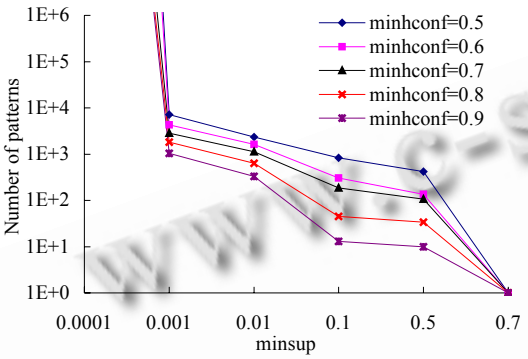


Fig.5 Result of mining hyperclique patterns patterns
图 5 挖掘超团模式结果

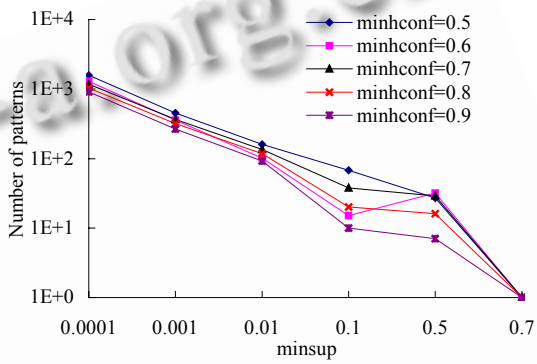


Fig.6 Result of mining maximal hyperclique
图 6 挖掘极大超团模式结果

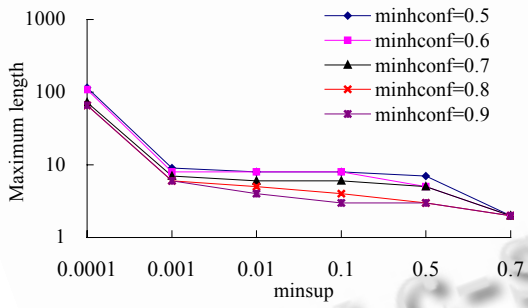


Fig.7 Maximum length of hyperclique patterns in mining result
图 7 挖掘结果中超团模式的最大长度

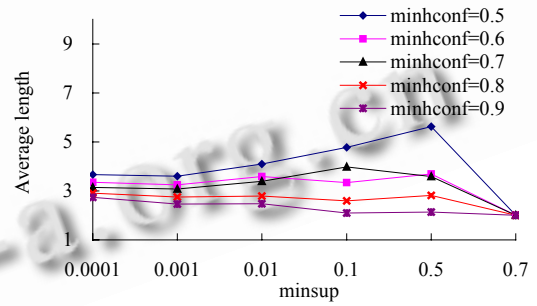
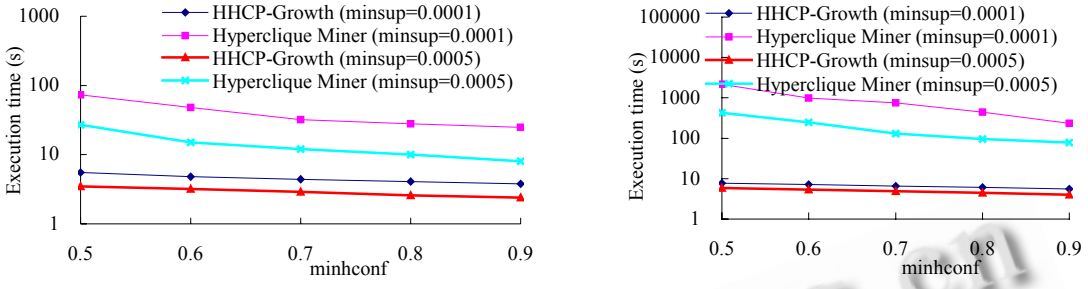


Fig.8 Average length of maximal hyperclique patterns in mining result
图 8 挖掘结果中极大超团模式的平均长度

4.3 HHCP-Growth与Hyperclique Miner性能比较

针对 Alarm 数据集和 Retail 数据集,图 9 分别给出 HHCP-growth 算法和 Hyperclique Miner 算法的性能比较,这里,两种算法的用时都没有包含输出结果的时间.分别设置 minsup 为 0.000 1 和 0.000 5 进行测试,从图中可以发现,由于 Hyperclique Miner 基于 Apriori 算法,需要保存大量候选项集,因此用时较大.而 HHCP-growth 算法采用递归处理,类似于 DFS 算法,无需保存候选项集,因此在各种最小 *h* 置信度下,HHCP-growth 算法都具有更高的时间效率.另外,由于 Retail 数据集中包含的项目比 Alarm 数据集要多,比较图 9(a)和图 9(b)两图可以发现,Hyperclique Miner 算法挖掘大数据集时用时更多,而 HHCP-growth 算法的用时相差不大,基本上都在 10s 以内.



(a) Mining Alarm dataset
(a) 挖掘 Alarm 数据集

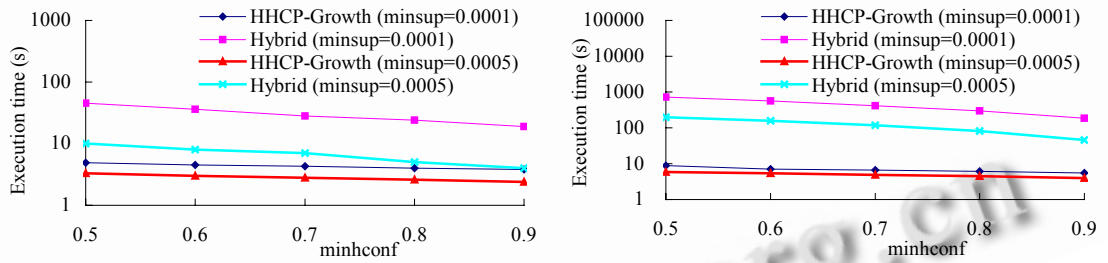
(b) Mining Retail dataset
(b) 挖掘 Retail 数据集

Fig.9 Execution time comparison for mining hyperclique patterns

图 9 挖掘超团模式时间开销比较

4.4 HHCP-Growth与Hybrid性能比较

在两数据集上挖掘极大超团模式,图 10 给出了两种算法的用时比较.实验中由于产生较多的一项频繁项集,Hybrid 算法在进行 BFS 时耗时较大,尤其是挖掘 Retail 数据集下,基本都需要几十秒到几百秒的时间,而HHCP-growth算法在递归时利用有效的项目自剪枝和剩余项目剪枝,使挖掘用时得到较好的控制,基本都在10s以下,因此具有较好的时间效率.另外,分析不同 minsup 下的用时,可以发现在低支持度条件下 HHCP-growth 算法的挖掘效率较 Hybrid 算法更高.



(a) Mining alarm dataset
(a) 挖掘 Alarm 数据集

(b) Mining Retail dataset
(b) 挖掘 Retail 数据集

Fig.10 Execution time comparison for mining maximum hyperclique patterns

图 10 挖掘极大超团模式时间开销比较

5 结 论

超团模式是一种新型的关联模式,这种关联模式所包含的项目相互间具有很高的亲密度.超团模式中某个项目在事务中的出现很强地暗示了模式的其他项目也会出现.极大超团模式是一组超团模式更加紧凑的表示,可被用于多种应用,如基于模式的聚类和分类.针对两种模式的挖掘,本文提出一种基于FP-tree的统一挖掘算法HHCP-growth.算法采用了递归挖掘的思想,无需保存全部候选项集,除了应用传统的最小支持度剪枝策略以外,还引入最大支持度剪枝,项目自剪枝以及剩余项目剪枝等策略,可以快速去除冗余项目,减少遍历和递归的次数,使挖掘效率大为提升.针对剪枝策略的有效性和算法的正确性,文中提出并证明几个相关命题来加以分析.同时,本文对算法的挖掘效率也进行了定性分析.实验结果表明,HHCP-growth 算法相对于标准的超团模式挖掘算法和极大超团模式挖掘算法而言,都具有更高的效率,尤其是在大数据集或低支持度的条件下更为显著.

References:

[1] Agrawal R, Imielinski T, Swami A. Mining association rules between sets of items in large databases. In: Buneman P, Jajodia S, eds. Proc. of the ACM SIGMOD Conf. on Management of Data (SIGMOD'93). New York: ACM Press, 1993. 207−216.

[2] Xiao B, Xu QF, Lin ZQ, Guo J, Li CG. Credible association rule and its mining algorithm based on maximum clique. Journal of Software, 2008,19(10):2597−2610 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/19/2597.htm [doi: 10.3724/SP.J.1001.2008.02597]

[3] Xu QF, Xiao B, Guo J. A mining algorithm with alarm association rules based on statistical correlation. Journal of Beijing University of Posts and Telecommunications, 2007,30(1):66−70 (in Chinese with English abstract).

[4] Agrawal R, Srikant R. Fast algorithms for mining association rules. In: Bocca JB, Jarke M, Zaniolo C, eds. Proc. of the 20th Int'l Conf. on Very Large Data Bases. Santigo: Morgan Kaufman Publishers, 1994. 478−499.

[5] Han J, Pei J, Yin Y. Mining frequent patterns without candidate generation. In: Chen WD, Naughton J, Bernstein PA, eds. Proc. of the 2000 ACM SIGMOD Int'l Conf. on Management of Data (SIGMOD 2000). New York: ACM Press, 2000. 1−12.

[6] Xiong H, Tan PN, Kumar V. Mining strong affinity association patterns in data sets with skewed support distribution. In: Wu XD, Tuzhilin A, Shavlik J, eds. Proc. of the ICDM 2003. Melbourne: IEEE Computer Society, 2003. 387−394.

[7] Xiong H, Tan PN, Kumar V. Hyperclique pattern discovery. Data Mining and Knowledge Discovery Journal, 2006,13(2):219−242. [doi: 10.1007/s10618-006-0043-9]

[8] Huang YC, Xiong H, Wu WL, Deng P, Zhang ZN. Mining maximal hyperclique pattern: A hybrid search strategy. Information Sciences, 2007,177(3):703−721. [doi: 10.1016/j.ins.2006.07.029]

[9] Burdick D, Calimlim M, Gehrke J. MAFIA: A maximal frequent itemset algorithm for transactional databases. In: Georgakopoulos D, Buchmann A, eds. Proc. of the 17th IEEE Int'l Conf. on Data Engineering. Heidelberg: IEEE Computer Society Press, 2001. 443−452.

附中文参考文献:

[2] 肖波,徐前方,蔺志青,郭军,李春光.可信关联规则及其基于极大团的挖掘算法.软件学报,2008,19(10):2597−2610. http://www.jos.org.cn/1000-9825/19/2597.htm [doi: 10.3724/SP.J.1001.2008.02597]

[3] 徐前方,肖波,郭军.一种基于相关度统计的告警关联规则挖掘算法.北京邮电大学学报,2007,30(1):66−70.

肖波(1975－),男,山东高密人,博士生,讲师,主要研究领域为数据挖掘.

蔺志青(1959－),女,教授,主要研究领域为智能信息处理,计算机网络及应用.

张亮(1979－),男,博士生,主要研究领域为数据挖掘.

郭军(1959－),男,博士,教授,博士生导师,主要研究领域为模式识别,网络搜索,数据挖掘.

徐前方(1975－),女,博士,讲师,主要研究领域为数据挖掘,网络管理.