ISSN 1000-9825, CODEN RUXUEW
Journal of Software, Vol.19, No.2, February 2008, pp.338−350
DOI: 10.3724/SP.J.1001.2008.00338
© 2008 by Journal of Software. All rights reserved.
E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel/Fax: +86-10-62562563

基于FP-Tree的反向频繁项集挖掘*

郭宇红[1+], 童云海[2], 唐世渭[1,2], 杨冬青[1]

[1](北京大学 计算机科学技术系,北京 100871)
[2](北京大学 视觉与听觉信息处理国家重点实验室,北京 100871)

Inverse Frequent Itemset Mining Based on FP-Tree

GUO Yu-Hong[1+], TONG Yun-Hai[2], TANG Shi-Wei[1,2], YANG Dong-Qing[1]

[1](Department of Computer Science and Technology, Peking University, Beijing 100871, China)
[2](State Key Laboratory of Machine Perception, Peking University, Beijing 100871, China)
+ Corresponding author: Phn: +86-10-62757756, E-mail: yhguo@pku.edu.cn, http://www.pku.edu.cn

Guo YH, Tong YH, Tang SW, Yang DQ. Inverse frequent itemset mining based on FP-tree. *Journal of Software*, 2008,19(2):338−350. http://www.jos.org.cn/1000-9825/19/338.htm

Abstract: After the current definition of the inverse frequent set mining problem is expanded and its three practical applications are explored, an FP-tree-based method is proposed for the inverse mining problem. First, the method divides target constraints into some sub constraints and each time it solves a sub linear constraint problem. After some iterations, it finds an FP-tree satisfying the whole given constraints. Then, based on the FP-tree it generates a temporary database *TempD* that only involves frequent items. The target datasets are obtained by scattering infrequent items into *TempD*. Theoretic analysis and experiments show that the method is right and efficient. Moreover, compared with the current methods, the method can output more than one target data set.
Key words: inverse mining; FP-tree; frequent itemset; privacy preserving; knowledge hiding

摘 要: 在拓展现有反向频繁挖掘问题定义,探索反向频繁项集的 3 个具体应用后,提出了一种基于 FP-tree 的反向频繁项集挖掘方法.该方法首先采用分治思想,将目标约束划分为若干子约束,每步求解一个子线性约束问题,经过若干步迭代后找到一个满足整个给定约束的目标 FP-tree;然后根据目标 FP-tree 生成一个仅含频繁项的临时事务数据库 *TempD*;最后通过向 *TempD* 中撒入非频繁项得到目标数据集.理论分析和实验表明该方法是正确的、高效的,且与现有方法仅能输出 1 个目标数据集相比,该方法能够输出较多的目标数据集.
关键词: 反向挖掘;FP 树;频繁项集;隐私保护;知识隐藏
中图法分类号: TP311 **文献标识码**: A

频繁项集挖掘是数据挖掘研究中最为突出的任务之一,从中可以派生出关联规则.由于频繁项集可以看作是原始数据集的一种汇总概括,从给定的频繁项集倒推原始数据集的反向频繁项集挖掘,近年来得到了较多的关注.这种关注一方面由于反向频繁项集挖掘会对原始数据集数据本身的安全和隐私造成威胁[1];另一方面由于反向频繁项集挖掘本身能够应用到敏感性关联规则的隐藏[2]、关联规则基准测试数据集的生成[3]等问题

* Supported by the National Natural Science Foundation of China under Grant No.60403041 (国家自然科学基金)
Received 2006-06-10; Accepted 2007-02-05

中去.

粗略地讲,反向频繁项集挖掘可以看作是频繁项集挖掘的反向过程.文献[4]对反向频繁项集挖掘问题的描述如下:给定一个频繁项集及其支持数的集合,找到一个事务数据集,该数据集能够精确地满足给定集合中的所有频繁项集及其对应的支持数,而其他项集的支持数小于预定义的支持数阈值.该问题与频繁项集刻画原始数据集特征的好坏程度相关,也与隐私在频繁项集中被保护的好坏程度相关,使得它在面向隐私保护的数据共享背景中有着非常实际的应用.

反向频繁项集挖掘问题由 Mielikainen 在文献[1]中首次提出,Mielikainen 在该文献中证明:找到一个数据集满足一个给定的频繁项集集合,或者确定是否存在一个满足给定频繁项集约束的数据集是 NP 完全的.Calder 在文献[5]中也提出了一个类似的问题,称作“频繁项集可满足性(frequency satisfiability,简称 FREQSAT)问题”,具体描述为:给定一些“项集-区间”对,找到一个数据库,使得每一个“项集-区间”对中的项集在该数据库中出现的频度落在该区间内.Calder 证明 FREQSAT 问题也是 NP 完全的.

目前,对于反向频繁挖掘这个 NP 完全问题,研究者提出了若干方法.文献[2,3]中提出了一种基于线性规划的方法,用以解决近似反向频繁项集挖掘问题,旨在构造一个事务数据库,该数据库近似满足给定的频繁项集约束.就精确反向频繁项集挖掘而言,Calder 在文献[5]中给出了一种朴素的“产生-测试”方法,以从给定的频繁项集“猜测”和水平地构造一个数据库.所谓“水平”,是指算法一个事务一个事务地构造数据库.与“产生-测试”框架下水平地构造数据库相反,文献[4]中提出了一种垂直的数据库生成算法,以垂直地“猜测”和构建一个数据库.所谓“垂直”,是指把事务数据库看成一个二维矩阵,一列一列地构造数据库.然而在这种“产生-测试”框架下,这两种算法都非常低效,因为它们从本质上都属于简单的穷举搜索方法,都是盲目地尝试所有可能值的设定,即便尝试的路径是缺乏解答的.这意味着一旦“测试”过程发现前面的尝试失败,算法必须回滚已经执行的昂贵的“产生-测试”操作和回溯到某个源点,从而导致巨大的计算开销.

针对这种状况,本文提出了一种基于 FP-tree 的、无回溯的反向频繁项集挖掘方法.我们的方法针对精确反向频繁项集挖掘,且输入中给定的频繁项集和支持数来自一个真实的数据库,这将保证至少这个原始的数据库是精确地满足给定频繁项集和支持数约束的目标数据库.我们把 FP-tree 看成是频繁项集和原始数据集之间的一个过渡桥梁,先通过分治策略将目标约束分解成若干子约束,针对每个子约束建立线性不等式方程组求解,经过若干步迭代后找到一个满足整个给定约束的目标 FP-tree;然后根据目标 FP-tree 生成一个满足给定约束的、仅包含频繁项的临时事务数据库 *TempD*;最后通过向 *TempD* 撒入非频繁项得到目标数据库.理论分析表明我们的方法是正确的、高效的,并且与现有方法仅能输出 1 个目标数据库相比,我们的方法能输出一系列(至少 1 个,不是所有,但通常有很多)的满足约束的目标数据库.

本文第 1 节给出反向频繁项集问题的定义和描述.第 2 节探讨反向频繁项集挖掘在一些场景中的典型应用.第 3 节详细陈述所提出的基于 FP-tree 的反向频繁项集挖掘方法.第 4 节对全文作出总结.

1 反向频繁项集挖掘问题定义

设 $I=\{I_1,I_2,...,I_m\}$ 为 m 个项的集合,$D=\{T_1,T_2,...,T_n\}$ 为事务数据库,其中事务 $T_i(i\in[1,...,n])$ 为 I 的子集.项集 $A\subseteq I$ 在 D 中的支持数是指 D 中包含 A 的事务数,记为 $support(A)$.如果 A 的支持数不小于用户预定义的最小支持数阈值σ,则称 A 为 D 中的频繁项集,D 中所有频繁项集构成的集合记为 FI.如果 A 是频繁的,并且 A 的超集都是不频繁的,即对于任意 $B\supset A$,$support(B)<\sigma$,则称 A 为 D 中的最大频繁项集,D 中所有最大频繁项集构成的集合记为 MFI.如果 A 是频繁的,并且不存在 A 的超集使得每一个包含 A 的事务也包含 A 的这个超集,则称 A 为 D 中的频繁闭合项集, D 中所有频繁闭合项集构成的集合记为 FCI.根据定义,不难得出 $MFI\subseteq FCI\subseteq FI$.

定义 1(**反向频繁项集挖掘 inverse frequent itemset mining,简称 IFIM**). 给定项集 $I=\{I_1,I_2,...,I_m\}$、最小支持数阈值σ、频繁项集集合 $FI=\{f_1,f_2,...,f_n\}$及其对应的支持数集合 $S=\{support(f_1),support(f_2),...,support(f_n)\}$,找到一个数据库集合 DBs,使得 DBs 中的每一个数据库 D'是基于 I 上的数据集,并且在同样的最小支持数阈值σ下,从 D'中能够挖掘出带有同样支持数集合 S 的同样的频繁项集集合 FI.

定义 2(**反向最大频繁项集挖掘 inverse maximal frequent itemset mining**,**简称 IMFIM**). 给定项集 $I=\{I_1, I_2,...,I_m\}$、最小支持数阈值σ、最大频繁项集集合 $MFI=\{f_1,f_2,...,f_n\}$及其对应的支持数集合 $S=\{support(f_1), support(f_2),...,support(f_n)\}$,找到一个数据库集合 DBs,使得 DBs 中的每一个数据库 D'是基于 I 上的数据集,并且在同样的最小支持数阈值σ下,从 D'中能够挖掘出带有同样支持数集合 S 的同样的最大频繁项集集合 MFI.

定义 3(**反向频繁闭合项集挖掘 inverse frequent closed itemset mining**,**简称 IFCIM**). 给定项集 $I=\{I_1,I_2,..., I_m\}$、最小支持数阈值σ、频繁闭合项集集合 $FCI=\{f_1,f_2,...,f_n\}$及其对应的支持数集合 $S=\{support(f_1), support(f_2),...,support(f_n)\}$,找到一个数据库集合 DBs,使得 DBs 中的每一个数据库 D'是基于 I 上的数据集,并且在同样的最小支持数阈值σ下,从 D'中能够挖掘出带有同样支持数集合 S 的同样的频繁闭合项集集合 FCI.

显然,由于频繁闭合项集及其对应的支持数即$\langle FCI,S\rangle$唯一地决定了所有的频繁项集及其准确支持数$\langle FI,S\rangle$,所以,上述定义中的“反向频繁项集挖掘”问题和“反向频繁闭合项集挖掘”问题是完全等价的,可视为同一个问题,而“反向最大频繁项集挖掘”问题和这两个问题本质上是不等价的.从输入规模上看,“反向最大频繁项集挖掘”问题要远小于“反向频繁闭合项集挖掘”问题,而“反向频繁闭合项集挖掘”问题又要远小于“反向频繁项集挖掘”问题,因为通常 MFI 中的元素数要比 FCI 中的元素数少几个数量级,FCI 中的元素数又要比 FI 中的元素数少几个数量级.关于这 3 个问题的比较见表 1.

Table 1 Comparison of IFIM, IMFIM and IFCIM

表 1 IFIM,IMFIM 和 IFCIM 的比较

Problem	Input size	Target constraint	Solution space	Computational complexity
IFIM	Large	As IFCIM	As IFCIM	NP-Complete
IMFIM	Small	Loosest	Largest	Open problem
IFCIM	Middle	As IFIM	As IFIM	NP-Complete

定义 2 和定义 3 是对现有反向频繁项集挖掘问题定义的拓展,其现实意义在于:(1) 相对于定义 1、定义 2 和定义 3 的输入规模要减小很多;(2) 当数据中存在长的频繁模式时,产生所有的频繁项集是不切实际的,并且对一些数据挖掘应用来讲,比如生物领域中的组合模式发现,找到最大模式或闭合模式已经足够,这样一来,就有一种驱动使得我们想从最大频繁项集或频繁闭合项集出发,重构原始数据集,或找到与原始生物数据集具有相同最大频繁项集特征的基准样例数据集.

需要说明的是,本文中定义的反向频繁项集挖掘问题,是找到能够精确满足频繁项集和支持数约束的数据集,而不同于文献[2]中的近似反向频繁项集挖掘.文献[2]试图寻找的数据集只能近似满足给定的支持数约束,也就是说,从目标数据集中挖掘出来的频繁项集的支持数和给定的支持数是有一个区间差的.另外,本文中定义的反向频繁项集问题,输出是由满足给定约束的数据库组成的集合(通常输出不止 1 个数据库),而不同于现有文献对于反向频繁项集挖掘方法的研究,只输出 1 个满足给定约束的数据库.

关于反向频繁项集挖掘问题,Mielikainen 已经证明仅仅确定是否存在一个数据集满足给定频繁项集约束就是 NP 完全的.然而,如果给定的频繁项集和对应的支持数正好是从一个已有的数据库中挖掘出来的,那,么显然这个原有的数据库就是一个满足约束的目标解,即便是从计算代价上很难找到它.问题和挑战在于,它是唯一的吗?我们能够通过一个较好的搜索策略快速地找到一个目标数据集吗?本文的第 3 节将针对频繁项集和支持数来自一个已有数据库的反向频繁项集挖掘问题,给出一种求解方法.

2 反向频繁项集挖掘应用

2.1 面向隐私数据保护的频繁项集挖掘

数据挖掘能够基于对数据的分析生成有意义的模式和知识.而出于人们对于个人隐私数据的保护,面向隐私数据保护的数据挖掘成为近几年来的研究热点之一,其任务是在不精确访问个体隐私数据的条件下仍能得到准确的挖掘模型.具体到面向隐私数据保护的频繁项集挖掘[6−8],其目标一是保护原始数据集,二是保证挖掘出来的频繁项集的准确性.下面我们根据这两个方面的目标,分析和阐述反向频繁项集挖掘在面向隐私数据保护的频繁项集挖掘中的作用.

其一,面向隐私保护的数据挖掘的目标之一,就是要保护原始数据集不被恶意或者好奇的用户通过某种手段探知,从而达到保护隐私数据的目的.然而,由于频繁项集可以看作是原始数据集的一种较低层次的汇总信息,恶意用户就很可能通过反向频繁项集挖掘来猜测推导出原始数据集,从而对原始数据中的隐私造成严重威胁.虽然从这一点来讲,反向频繁项集挖掘似乎是起反面作用,然而为了阻止恶意用户的威胁和更好地保护隐私,面向隐私保护的数据挖掘研究者非常有必要研究反向频繁项集挖掘,正如为了让自己的加密算法更安全,研究加密算法的人也要研究解密算法一样.

其二,已有的面向隐私保护的数据挖掘研究中一直面临着一个不可调和的矛盾,就是隐私保护度和数据挖掘结果准确度的对立.经常出现的现象是,数据挖掘结果准确度提高了,隐私保护度就降低了;反之,隐私保护度提高了,挖掘结果准确度却降低了.造成这种现象的很重要的一个原因就是已有的研究缺乏对数据集和频繁项集映射关系的研究和探索,而反向频繁项集挖掘则为研究两者关系进而调和上述矛盾提供了契机,因为通过反向挖掘,能够从频繁项集约束推导出满足约束的数据集,从而得到两者的映射关系.假如频繁项集约束与数据集不是一对一映射关系(如果是,则说明要保证挖掘结果绝对准确,就必须贡献原始数据),而是一对多映射关系,就意味着存在多个原始数据集的替代版本,其中的任意一个版本都能保证挖掘结果的准确性.这时,只需考虑隐私保护度再从中选取一个即可.也就是说,反向频繁项集挖掘能够为先保证挖掘结果准确再考虑保护隐私提供支持.

2.2 敏感性关联规则的隐藏

反向频繁项集挖掘的第 2 个应用就是用于敏感性关联规则的隐藏[9−11].这一问题的提出源于人们对数据中所蕴涵的知识的保护.在一些场景中,商家虽然愿意共享数据,但出于商业机密,而不愿意向公众暴露数据中隐藏的某些知识,比如某家超市愿意共享其客户交易数据,但由于商业竞争因素,而不愿意公开其数据中蕴藏的“大多数客户在买牛奶的同时也买了面包”这条规则.该问题的目标是在保证敏感规则不被挖掘出的条件下最大程度地保持原始数据集的其他特征.解决此问题的一种方法是基于对原始数据集中事务的修改,通过修改原始数据集中支持敏感性规则的事务达到保护敏感规则的目的,这种基于事务修改的方法会产生大量的 I/O 操作,且造成新数据集中一些规则的丢失和产生一些额外规则,同时数据拥有者的工作量很大.文献[12]提出了一种基于反向项集格挖掘的敏感性关联规则隐藏方法,该方法可以看作是反向频繁项集挖掘在敏感性关联规则隐藏中的一个应用雏形,对其抽象后可以概括出反向频繁项集挖掘在敏感性关联规则隐藏中的应用场景如下:

(1) 数据拥有者自己利用挖掘工具从数据集 D 中挖掘出频繁项集 F;

(2) 数据拥有者从 F 中删去支持敏感性规则的频繁项集 F_h;

(3) 数据拥有者根据 $F-F_h$,利用反向频繁项集挖掘工具生成一个人造数据集 D',将 D'发布给公众.

2.3 基准数据集的生成

反向频繁项集挖掘的第 3 个应用是生成基准测试数据集(benchmark).文献[3]对这一应用的动机和背景进行了说明.文献[3]指出,已有的频繁项集挖掘算法只在使用人造数据集(比如 IBM Almaden 提供的购物篮数据生成器生成的基准数据)时显示出各自的性能优势,然而,文献[3]的作者在分别使用人造数据集和真实数据集对 5 种著名的频繁项集挖掘算法进行测试后发现,人造数据集与真实数据集的特征相差很远,造成基于人造数据集得到的性能测试结果和基于真实数据集得到的性能测试结果差别很大.因此文献[3]指出,非常有必要使用真实数据集作为关联规则挖掘社区的基准测试数据集.然而,出于对隐私数据的保护,数据拥有者可能不愿意提供其真实数据集.在这样的背景下,反向频繁项集挖掘就有了新的用武之地,它能够在不泄露真实数据的情况下生成具有真实数据集特征的数据集供公众使用.下面给出反向频繁项集挖掘应用在基准数据集生成中的两个场景.

场景 1:

(1) 数据拥有者自己利用挖掘工具从 D 中挖掘出频繁项集 F;

(2) 数据拥有者根据频繁项集 F 利用反向频繁项集挖掘工具生成一个数据集 D';

(3) 将 D'发布给公众作为基准数据集.

场景 2:

(1) 数据拥有者自己利用挖掘工具从 D 中挖掘出频繁项集 F;

(2) 将 F 发布给公众;

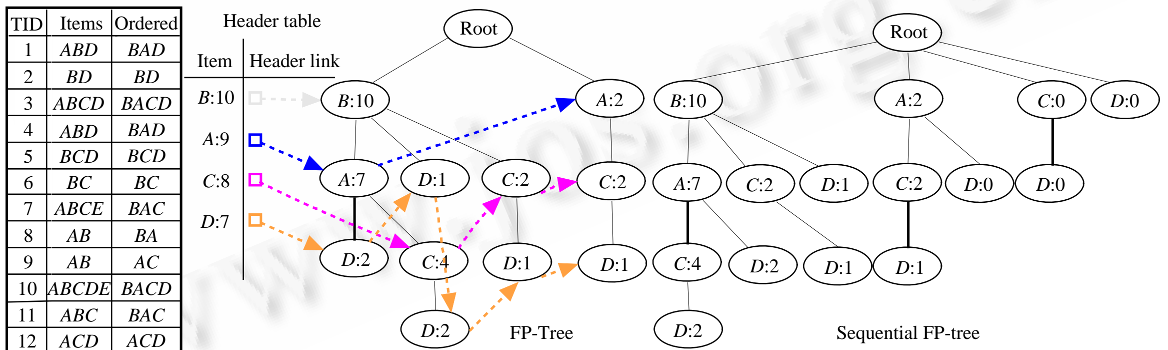
(3) 想使用人造数据集的用户根据 F,利用反向频繁项集挖掘工具生成数据集 D'来使用.

3 基于FP-tree的反向频繁项集挖掘方法

3.1 FP-tree和有序FP-tree

频繁模式树(frequent pattern tree,简称 FP-tree)是韩家炜在文献[13]中提出的一种扩展前缀树结构,用于在频繁项集挖掘[14,15]中存储关于频繁模式的压缩信息.它由一个根节点和作为其孩子的项前缀子树集合组成.项前缀子树的每个节点由 3 个域组成:节点名称 item-name、节点记数 count 和节点链 node-link,其中,节点记数表示到达该节点的路径上的事务数,节点链指向树中具有同一名称的下一节点.另外,为方便树遍历,创建了一个频繁项头表 Head Table,它由两个域组成:item 和 HeadLink,其中,HeadLink 指向树中与之项名相同的第 1 个节点.

图 1 给出了一个事务数据库和当σ=4 时所对应的 FP-tree 的例子.该数据库由 12 个事务组成,事务中所包含的项 item 取自项集{A,B,C,D,E},其 FP-tree 通过两次扫描数据库形成.第 1 次扫描数据库后得到按支持数递减顺序排列的频繁项序列 L=[B:10,A:9,C:8,D:7].然后,将数据库每个事务中的项也按此顺序排列,得到按 1-项集支持数大小递减排列的数据库事务(如图 1 表格中最右列所示).其 FP-tree 在对表格最右列的 12 个数据库事务扫描过程中构建,大致可看成是事务一个个地依次爬上树的过程,具体过程详见文献[13].若树中节点的孩子节点也按序列 L 中的顺序从左到右排列,就形成了一棵 4 阶有序 FP-tree(如图 1 右所示),该有序 FP 树中节点的排列非常严格,其结构仅取决于 L 中的项和项的排列次序.假设 $L=[I_1,I_2,...,I_p]$,则对应的 p 阶有序 FP-tree 的结构完全确定,包括根节点在内共 2^p 个节点,其中,$I_i(1\le i\le p)$只分布在树的前 i 层,共包含 2^{i-1} 个节点.FP 树中所有的节点计数均不小于 0,且子节点的计数之和不小于父节点计数.不同的事务数据库若有相同的 L(各项的支持数可不同,但排列次序相同),则它们对应的有序 FP-tree 结构完全一样,不一样的是树中各节点的节点计数.



TID	Items	Ordered
1	*ABD*	*BAD*
2	*BD*	*BD*
3	*ABCD*	*BACD*
4	*ABD*	*BAD*
5	*BCD*	*BCD*
6	*BC*	*BC*
7	*ABCE*	*BAC*
8	*AB*	*BA*
9	*AB*	*AC*
10	*ABCDE*	*BACD*
11	*ABC*	*BAC*
12	*ACD*	*ACD*

Fig.1 A transaction database, its frequent pattern tree and sequential FP-tree

图 1 一个事务数据库及其频繁模式树 FP-tree 和有序 FP-tree

3.2 基于FP-tree的反向频繁闭合项集挖掘算法

(1) 基本思想

我们的方法利用 FP-tree 作为从频繁项集到数据集的过渡桥梁,可以看作是基于 FP-tree 的频繁项集挖掘的反向过程.我们的思想来源于这样一个事实,即 FP-tree 作为一种非常紧凑的数据结构,存储了事务数据库与频繁项集挖掘相关的全部信息,可以看成是原始数据库和对应的频繁项集的中间产物.直观地讲,FP-tree 减少了数据库和频繁项集之间的差距,使得从给定的频繁项集到数据库的转换过程变得平滑、自然且容易.

我们的方法工作步骤大体如下:首先,我们找到一个满足给定频繁项集和支持数约束的 FP-tree;然后,通过该 FP-tree 生成一个满足给定约束的、仅由频繁项构成的临时数据库 *TempD*;最后,在该临时数据库 *TempD* 的基础上,通过在最小支持数阈值的限制下,撒入非频繁项生成一系列满足约束的目标数据库.

(2) 算法描述

算法. 基于 FP-tree 的反向频繁闭合项集挖掘算法 FP-IFCIM.

输入:项集 I,最小支持数阈值 σ,由带有支持数的频繁闭合项集构成的集合 F.

输出:一系列满足 F 的目标数据库所构成的集合 *DBs*. //从集合 *DBs* 中的每个元素中都能挖掘出同样的 F

Gen_DB(F,I,σ) //算法主过程

1. $DBs \leftarrow \varnothing$, $I\overline{F} \leftarrow I-\{$出现在$F$中的所有item$\}$; // $I\overline{F}$ 为非频繁项组成的集合
2. $FP \leftarrow Gen_FPtree(F,\sigma)$; //调用子过程生成FP-tree
3. $TempD \leftarrow Outspread(FP)$; //调用子过程生成*TempD*
4. $DBs \leftarrow DBs \cup \{TempD\}$;
5. **For** i=1 to $|I\overline{F}|$,
   (a) $NewTempDSet_i \leftarrow$将 $I\overline{F}$ 中第i个非频繁项撒入到*DBs*中的所有数据库后新形成的数据库集合,注意每个非频繁项被撒入的事务数要小于σ;
   (b) $DBs \leftarrow DBs \cup NewTempDSet_i$;
6. **Return** *DBs*;

Gen_FPtree(F,σ) //算法子过程

1. 产生FP-tree的根节点*FP*,标记为null; //生成0阶FP-tree
2. 提取F中所有的item,并按其支持数从高到低排列,形成有序列表$L=[I_1,I_2,...I_p]$;
3. 将F中的项集按L中项的顺序排列,然后划分到p个集合$F(I_1),F(I_2),...,F(I_p)$中去,其中,$F(I_k)$表示所有以项I_k结尾的项集所构成的集合;
4. **For** i=1 to p,生成i阶有序FP-tree;
5. **Return** *FP*;

Outspread(FP) //算法子过程

1. $TempD \leftarrow \varnothing$;
2. if (FP=null) then return *TempD*
   else
   (c) 深度优先搜索找到一个叶子节点$ln{:}s$,其中,ln为item名,s为节点记数*count*;
   (d) $t \leftarrow$从根*FP*到叶子节点的路径上的所有item组成的项集;
   (e) **For** i=1 to s, $TempD \leftarrow TempD \cup \{t\}$; //*TempD*可包含重复事务
   (f) Update *FP*: //更新从根到叶子所在路径上的节点记数*count*
   For 根 *FP* 到叶子节点 *ln* 路径上的每一个节点 n,
   i. $n.count \leftarrow n.count-s$;
   ii. **if** ($n.count$=0) **then** 将节点 n 从树中删除;
   (g) *Outspread*(*FP*); //递归调用

整个算法由 3 个过程组成.在主过程"*Gen_DB*()"中,*FP* 表示通过调用子过程"*Gen_FPtree*()"从频繁闭合项集 F 得到的 FP 树,*TempD* 表示通过调用子过程"*Outspread*()"而得到的临时数据库事务集.注意,*FP* 和 *TempD* 都仅包含出现在 F 中的项.语句 5 在 *TempD* 的基础上以及在最小支持数阈值的限制下,通过撒入一些非频繁项形成一系列的数据库,并构成集合 *DBs*.假定非频繁项的个数为 $q(|I\overline{F}|=q)$, $I\overline{F}=\{item_1,...,item_q\}$,则 $DBs=\{TempD\}\cup NewTempDSet_1\cup...\cup NewTempDSet_i\cup...\cup NewTempDSet_q$,其中,$NewTempDSet_1$ 是指把 $item_1$ 撒入到 *TempD* 后新生成的所有包含 $item_1$ 的数据库集合,$NewTempDSet_i$ 是指把 $item_i$ 撒入到前面已经生成的集合$\{TempD\}\cup...\cup NewTempDSet_{i-1}$中的所有数据库后新生成的所有包含 $item_i$ 的数据库集合.

www.jos.org.cn
www.jos.org.cn

在子过程“*Gen_FPtree*()”中,我们采用分治策略和增量迭代的思想来生成一个满足 F 的 FP 树.我们将需要被满足的频繁项集集合 F 划分成 p 个子集(p 为频繁项的个数),每次循环面向一个子集中的频繁项集,每一次循环结束后得到的 FP-tree 满足所有先前循环所涉及的所有频繁项集和支持数约束以及当前子集中的所有频繁项集和支持数约束.这样,经过 p 次循环迭代后,就得到了一个满足 F 的 FP-tree(即在最小支持数阈值σ限制下,从该 FP-tree 可以挖掘出同样的 F).其中,语句 4 生成 i 阶有序 FP-tree 的过程如下:

(1) 让 $i-1$ 阶有序 FP-tree 所有节点均长出一个节点名称为 I_i 的新叶子节点,此时,新叶子节点数等于 $i-1$ 阶有序 FP-tree 的总节点数,即为 2^{i-1}.假定这些新叶子节点的计数从左到右依次为未知变量 $x1,x2,...,x2^{i-1}$.

(2) 针对上述未知变量,根据下面 3 个约束条件列出由 2^i-1 个约束方程组成的 2^{i-1} 元线性方程组:① 对于树中包含以 I_i 结尾的某个频繁项集的所有分支,其新叶子节点计数之和等于该项集的支持数,其中,以 I_i 结尾的所有频繁项集和其支持数可由频繁闭合项集 $F(I_i)$推导出来;② 对于树中包含以 I_i 结尾的某一个非频繁项集的所有分支,其新叶子节点计数之和小于支持数阈值;③ 新节点计数与其他兄弟节点计数之和小于等于其父节点的节点计数.由于所有包含 I_i 的项集个数为 2^{i-1}(前 $i-1$ 个项或者出现,或者不出现),所以,根据①,②可得到 2^{i-1} 个方程;而由于新节点个数为 2^{i-1},除了与根节点的一次比较(根节点的节点计数为无穷大)以外,根据③可得到 $2^{i-1}-1$ 个不等式,因此,共得到 $2^{i-1}+2^{i-1}-1=2^i-1$ 个方程组成的线性不等式方程组.

(3) 找到上述不等式方程组的一个解,将求得的解依次填入到新叶子节点的节点计数中,若求得的某个节点计数为 0,则将该节点从树中移除.

在子过程“*Outspread*()”中,FP-tree 的每一条路径上的项集从树上逐个“爬”下来,依次形成 *TempD* 的一个个事务.此过程是文献[13]中所描述的构造 FP-tree 的逆过程,最终形成的临时事务数据库 *TempD* 可以看成是按频繁 1-项集频繁度排序后的事务数据库状态(如图 1 中表格的最右列数据库所示).

3.3 算法举例

如图 2 所示,我们通过一个例子来说明算法的思想和执行过程.给定 $I=\{A,B,C,D,E\}$;最小支持数阈值$\sigma=4$;频繁闭合项集集合 $F=\{A_9,B_{10},C_8,D_7,AB_7,AC_6,AD_5,BC_6,BD_6,CD_4,ABC_4,ABD_4\}$(下脚标表示支持数),其中,$F$ 是在$\sigma=4$ 时通过频繁闭合项集挖掘算法从图 1 所示的事务数据库挖掘出来的.我们的算法将输出由一系列的事务数据库构成的集合 *DBs*,满足在$\sigma=4$ 时从 *DBs* 的任意一个数据库都能挖掘出同样的频繁闭合项集 F.

FP-tree 的生成.首先提取 F 中所有的 item,并按其支持数从高到低排列,得到有序列表 $L=[BACD]$,将 F 中的所有项集按 L 中项的顺序排列,并划分到 4 个集合中,得到 $F(B)=\{B_{10}\}$,$F(A)=\{A_9,BA_7\}$,$F(C)=\{C_8,AC_6,BC_6BAC_4\}$,$F(D)=\{D_7,AD_5,BD_6,CD_4,BAD_4\}$.我们的算法要生成的是一个 4 阶有序 FP 树,其完整结构如图 2 左上角所示,我们需要求出 $b1;a1,a2;c1,c2,c3,c4;d1,d2,d3,d4,d5,d6,d7,d8$ 共 4 组 15 个未知非负整数变量.首选,根据 $F(B)=\{B_{10}\}$,可得到 $b1=10$,从而生成一个满足 $F(B)$的 FP 树 $FP(B)$,如图 2(a)所示;其次根据 $F(A)=\{A_9,BA_7\}$可得到方程组($a1+a2=9,a1=7,a1\le 10$),其中,$a1\le 10$ 是根据子节点计数不大于父节点计数得来的,求此方程组得到 $a1=7,a2=2$,从而生成了一个满足 $F(B)$和 $F(A)$的 FP 树 $FP(BA)$,如图 2(b)所示;然后,根据 $F(C)=\{C_8,AC_6,BC_6,BAC_4\}$可得到方程组($c1+c2+c3+c4=8,c1+c3=6,c1+c2=6,c1=4,c1\le 7,c2\le 10-7,c3\le 2$),解此方程组得到 $c1=4,c2=2,c3=2,c4=0$,从而得到一个满足 $F(B),F(A)$和 $F(C)$的 FP 树 $FP(BAC)$,如图 2(c)所示;最后,根据 $F(D)=\{D_7,AD_5,BD_6,CD_4,BAD_4\}$可得到方程组($d1+d2+d3+d4+d5+d6+d7+d8=7,d1+d2+d5+d6=5,d1+d2+d3+d4=6,d1+d3+d5+d7=4,d1+d2=4,d1<4,d1+d3<4,d1+d5<4,d1\le 4,d2\le 7-4,d3\le 2,d4\le 10-7-2,d5\le 2,d6\le 2-2,d7\le 0$),其中,$d1<4,d1+d3<4,d1+d5<4$ 这 3 个不等式是根据 3 个非频繁项集 *BACD*,*BCD*,*ACD* 的支持数应小于阈值σ得到的,后面 7 个是根据子节点计数之和不大于父节点计数得到的.通过求解此方程组,得到了一个解 $d1=1,d2=3,d3=2,d4=0,d5=1,d6=0,d7=0,d8=0$,从而得到了一棵满足 $F(B),F(A),F(C)$和 $F(D)$的 FP 树 $FP(BACD)$,如图 2(d)所示.由于 $F(B),F(A),F(C)$和 $F(D)$构成了 F 的划分,故 $FP(BACD)$是满足给定约束 F 的 FP 树.

从 FP-tree 到 *TempD*.首先,图 2(d)所示的 FP-tree 中最左边的分支“$BACD_1$”从树上“爬”下来,形成 *TempD* 的第 1 个事务:$TempD(1)=BACD$.我们从树中删除该“$BACD_1$”分支,通过递归调用 $Outspread(FP)$,“BAC_3”从树上下

来,形成 3 个事务:*TempD*(2)=*BAC*;*TempD*(3)=*BAC*;*TempD*(4)=*BAC*.通过执行类似的操作,“BAD_3”,“BCD_2”,“*B*”,“ACD_1”,“AC_1”依次从树上“爬”下来,并相继形成 *TempD* 的剩余 8 个事务,如图 2 左下角的表格所示.

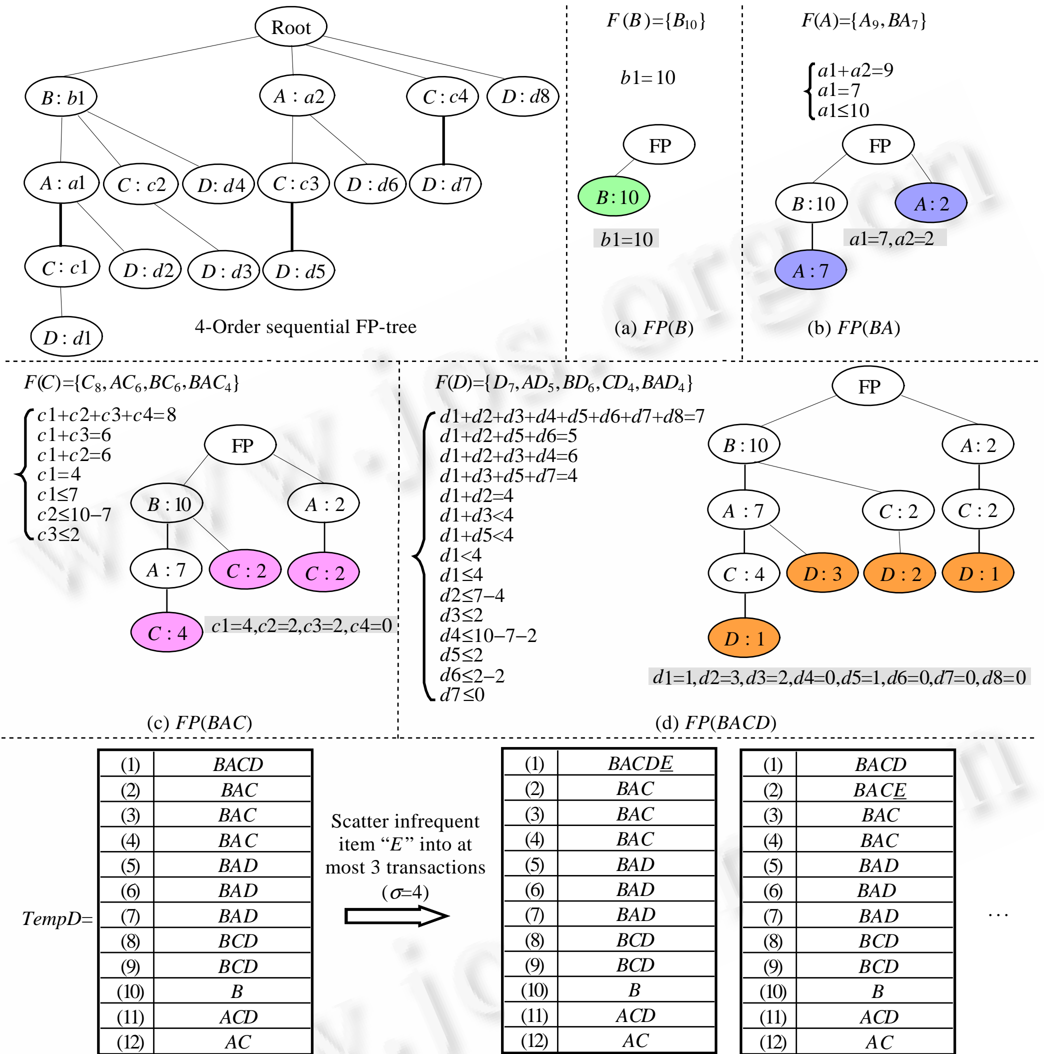


Fig.2 An example of FP-IFCIM algorithm execution process

图 2 FP-IFCIM 算法执行过程举例

从 *TempD* 到 *DBs*.上一过程形成的只包含频繁项的 *TempD* 保存了有关频繁项集的全部信息,构成了要找的目标数据库的“骨架”.通过在最小支持数阈值的限制下,向 *TempD* 撒入非频繁项,就能得到一个满足给定约束的目标数据库集合.在本例中,非频繁项集合 $\overline{IF}=I-\{A,B,C,D\}=\{E\}$,$\sigma=4$,将 *E* 撒入到 *TempD* 不超过 3 个事务中,就得到一系列既包含频繁项又包含非频繁项的目标数据库,如图 2 右下角的一系列表格所示.本例中最终形成的 *DBs* 包含的数据库个数等于 $C_{12}^0+C_{12}^1+C_{12}^2+C_{12}^3$.实际中可根据需要选择撒入策略,限制所生成的数据库数量.

3.4 算法分析

(1) 正确性

我们来检查一下算法的正确性,即验证 *DBs* 中的任一数据集都满足给定的频繁项集约束.首先,在 FP-tree 生成过程中,每一阶 FP-tree,都是基于目标约束的一个子约束列方程组求解得到的,这些子约束构成了整个目标

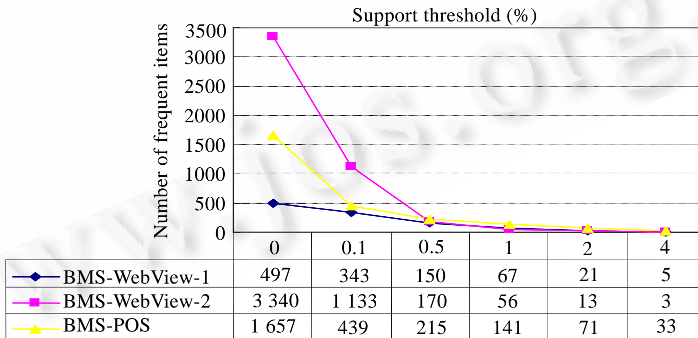
约束上的一个划分,而对整个目标约束由于已知至少有 1 个解,因此,根据子约束所列的方程组也至少有 1 个解,即每次迭代后一定能找到一个满足子约束的 FP-tree.这样,算法进行若干次迭代后,最终生成的 FP-tree 一定满足整个目标约束;其次,从 FP-tree 到 *TempD* 的生成过程中,所有的频繁项集都从树上“爬”下来,*TempD* 保留了 FP-tree 中的所有频繁项集及其支持数信息,因此,*TempD* 满足给定的频繁项集约束;最后,在从 *TempD* 到 *DBs* 的生成过程中,由于只是在支持数阈值的限制下,向 *TempD* 撒入非频繁项,保证了生成的任一数据集都能像 *TempD* 一样保留给定的频繁项集而不引入新的频繁项集,从而使 *DBs* 中的任一数据集都满足给定的频繁项集约束.

(2) 高效性

基于 FP-tree 的反向频繁项集挖掘算法的高效性源于以下 3 个方面:① 算法的无回溯性;② 问题转换过程规模的减小;③ 求解过程分治策略的应用.

首先,相对于文献[4,5]提出的“产生-测试”方法,我们的方法是无回溯的.在第 1 步生成 FP-tree 的过程中,FP-tree 增量迭代逐阶推进产生,每一次迭代求解一个线性不等式方程组,根据求得的解,FP-tree 扩展一层新频繁项节点.由于每次迭代所列的不等式方程组都至少存在 1 个解,因此,通过若干次迭代就可直接求出一个满足给定约束的目标 FP-tree.第 2 步从 FP-tree 到 *TempD* 的转换及第 3 步向 *TempD* 中撒入非频繁项的过程也是直接无回溯的,从而避免了回溯操作所造成的巨大计算开销.

其次,我们的方法首先生成一个满足给定约束的仅含频繁项的数据集 *TempD*,此过程暂时没有考虑非频繁项.这一思想使得在利用 FP-tree 将问题转换为线性约束问题时,未知变量和约束方程个数都大为减少,从而使转换后问题的规模大幅度减小,求解过程变得相对容易.我们与文献[3]中利用相依表(contingency table)将反向频繁项集挖掘问题转换为线性约束问题作一个比较.在文献[3]中,转换后的线性约束问题的变量数和约束方程个数为 $2^{|I|}-1$,其中,$|I|$为所有可能出现的项的个数;在我们的方法中,线性约束问题的变量数为 $2^{|IF|}-1$,约束方程个数为$(2^1-1)+(2^2-1)+...+(2^{|IF|}-1)=2^{|IF|+1}-|IF|-2$,其中,$|IF|$为频繁项的个数.$|IF|$与支持数阈值$\sigma$相关,通常,$|IF|$要比项的总个数$|I|$小得多,并且随着$\sigma$的增大,$|IF|$急剧减小,从而使问题规模也急剧缩小.为了验证这一点,我们对 3 个频繁项集挖掘中常用的 3 个真实数据集进行了测试,测试结果如图 3 所示.从图中可以看出,3 个数据集中项的总个数$|I|$分别为 497,3 340,1 657,当支持度阈值增大到 1%时,频繁项的个数$|IF|$分别减少到 67,56 和 141,因此与文献[3]相比,我们的方法在支持度阈值较大时,可以将问题的规模迅速减小,使问题的求解变得相对高效和容易.



	0	0.1	0.5	1	2	4
BMS-WebView-1	497	343	150	67	21	5
BMS-WebView-2	3 340	1 133	170	56	13	3
BMS-POS	1 657	439	215	141	71	33

Fig.3 The number of frequent items $|IF|$ decreases rapidly as support threshold grows

图 3 频繁项个数$|IF|$随着支持度阈值的增大而迅速减小

最后,在问题求解过程中,我们采用分治策略将目标约束分解为若干独立的子约束,通过对子约束问题的迭代求解求出目标解,这在一定程度上也降低了计算开销.具体地讲,假定$|IF|=p$,若不采用分步迭代算法,则求解含有 2^p-1 个未知变量的线性约束问题的时间复杂度为 $O((2^p-1)^3)\approx O(8^p)$,将该问题划分为子问题分为 p 步迭代计算时的复杂度为 $O(1^3+2^3+4^3+...+(2^{p-1})^3)=O((8^p-1)/7)$.可以看出,采用分步迭代后计算代价降低了约 1 个数量级.

(3) 输出的数据集个数

算法所输出的数据集个数$|DBs|$与下面 3 个参数有关:① *TempD* 中的事务数$|TempD|$;② $I\overline{F}$ 中非频繁项的个数$|I\overline{F}|$;③ 最小支持数阈值σ.假定$|TempD|=u$,$|I\overline{F}|=q$,$f(q)$表示当 $I\overline{F}$ 中第 q 个非频繁项撒入到前面已生成的数

据库后所形成的所有数据库的数量,则有递推方程: $f(q+1)=f(q)+f(q)(C_u^1+C_u^2+...+C_u^{\sigma-1})$ (n≥0,2≤σ≤u,u≥1,$\sigma\in N$)(C_u^1 代表从一个数据库中的 u 个事务中选择一个事务作为目标撒入的数据库的数量);初始状态 $f(0)=1$,意味着当 $I\bar{F}$ 为空,即没有非频繁项时,我们的算法找到唯一一个满足约束的数据库 *TempD*.解此递推方程,得到 $|DBs|=f(q)=(1+C_u^1+C_u^2+...+C_u^{\sigma-1})^q$ (q≥0,2≤σ≤u,u≥1,$\sigma\in N$).当σ=1 时,|*DBs*|=1;当σ<<u 时,|*DBs*|正比于 $(\sigma C_u^\sigma)^q$.在实际中,当|*DBs*|是一个天文数字时,可根据需要限制所输出的数据集个数.

3.5 实验评价

为了验证方法的正确性、可行性和高效性,我们选取真实数据集 BMS-WebView-1 作为原始数据集.该数据集的每个事务包含了一个 Web 访问进程中所浏览的全部页面.它被作为基准数据集用于各关联规则挖掘算法的性能评测中,可以在 KDD-CUP2000[16]上找到.其基本特性:事务总数|*Trans*|=59602;项种类|*Items*|=497;所包含的总项数|*ItemsCount*|=149639.我们的目标是在同样的支持数阈值下,找到若干与 BMS-WebView-1 具有完全相同挖掘结果的数据集.所设计的实验方法如下:(a) 设定支持数阈值σ,从 BMS-WebView-1 正向挖掘出带有支持数的频繁项集集合 *FS*;(b) 根据 *FS*,利用本文提出的算法 FP-IFCIM,从 *FS* 反向构造出新的数据集 *D′*;(c) 在最小支持数阈值σ下,对 *D′*实施正向挖掘,挖掘出结果集 *FS′*,判断 *FS′*是否等于 *FS*,验证 FP-IFCIM 的正确性和可行性,同时记录 FP-IFCIM 不同阶段的运行时间及总时间,验证和分析算法的高效性.

实验的硬件环境为 P4 1.5Ghz 的 CPU 和 256MB 的内存,软件环境为 Windows XP Professional 操作系统.正向挖掘采用文献[13]提出的 FP 树算法,代码用 Visual C++(6.0)实现,在反向挖掘 FP-IFCIM 算法的实现中,线性约束方程组的求解使用线性规划软件包 LINDO6.1,其余代码均用 Visual C++(6.0)实现.

(1) 正确性与可行性

表 2 给出了根据前述实验方法所得到的一组实验数据,每一行代表一次反向频繁项集挖掘求解过程.随着参数支持度阈值的增大,从 BMS-WebView-1 挖掘出的频繁项集个数减少,FP-IFCIM 算法的输入规模随之减小,进而使问题求解过程所涉及的未知变量和约束方程个数减小,LINDO(linear interactive and discrete optimizer)求解过程变得容易,生成的 FP 树和输出的临时事务数据库 *TempD* 规模也相应减小.由于 FP-IFCIM 算法中线性约束方程对于目标解是精确约束,所以只要 LINDO 在有序 FP 树构建的每一步能找到方程组的解,求出的 FP 树和 *TempD* 就一定满足目标约束.为了验证这一点,我们对仅包含频繁项的 *TempD* 重新实施正向挖掘,多次实验结果表明,在相同支持数阈值下,从 *TempD* 中能够挖掘出与 BMS-WebView-1 完全一样的频繁项集结果,从而验证了我们的方法是正确和可行的.

Table 2 Experimental data of the FP-IFCIM based on BMS-WebView-1

表 2 基于 BMS-WebView-1 的反向频繁项集挖掘实验数据

Parameter	Input		LINDO solving process			Generated FP-tree characters			Output *TempD*	
Support threshold (%)	|Frequent item sets|	|Frequent items|	|Unknown variables|	|Equations|	|Iterations|	|Orders|	|Layers|	|Nodes|	|Trans|	Total items
0.6	162	130	886	78	266	23	14	128	96 540	109 992
0.7	133	109	476	58	173	18	13	99	90 467	101 675
0.8	105	88	241	36	80	15	9	47	82 759	92 099
0.9	90	76	183	29	57	14	9	39	77 374	86 059
1	77	67	89	20	11	10	4	22	73 945	80 917

(2) 高效性

为了验证本文提出的 FP-IFCIM 算法的高效性,我们对求解过程不同阶段的运行时间进行了记录,如图 4 所示.结果表明:(1) 随着参数支持度阈值的增大,算法 FP-IFCIM 的运行时间迅速减小.其原因在于,随着支持度阈值的增大,目标约束减少,问题规模减小.(2) FP-IFCIM 算法最耗时的阶段是构建 FP-tree,对应于算法中的 *Gen_FPtree*(F,σ)子过程;(3) 运行时间随着支持度阈值变化幅度最大的阶段是构建 FP-tree,其他两个阶段相对平缓.这说明,当支持度阈值较大时,寻找目标 FP-tree 的过程很快;而当支持度阈值较小时,寻找目标 FP-tree 的过程将变得较慢.尽管如此,在我们的实验中,在支持度 0.6~1 的范围内,反向寻找与 BMS-WebView-1 具有相同挖掘结果的目标数据集的总耗时都在 1s 以内,表明我们的算法在支持度阈值较大时是高效的.其原因在于,实验中发