一种基于邻接表的最大频繁项集挖掘算法

殷 茗[①] 王文杰*[①] 张煊宇[①] 姜继娇[②]

[①](西北工业大学软件与微电子学院 西安 710072)

[②](西北工业大学管理学院 西安 710072)

摘 要：针对Apriori算法与FP-Growth算法在最大频繁项集挖掘过程中存在的运行低效、内存消耗大、难以适应稠密数据集的处理、影响大数据价值挖掘时效等问题，该文提出一种基于邻接表的最大频繁项集挖掘算法。该算法只需遍历数据库一次，同时用哈希表对邻接表进行辅助存储，减小了遍历的空间规模。理论分析与实验结果表明，该算法时间与空间复杂度较低，提高了最大频繁项集挖掘速率，尤其在处理稠密数据集时具有较好的优越性。

关键词：数据挖掘；频繁项集；Apriori；FP-Growth；FP-Tree

中图分类号：TP311.5 **文献标识码**：A **文章编号**：1009-5896(2019)08-2009-08

DOI: 10.11999/JEIT180692

A Maximal Frequent Itemsets Mining Algorithm Based on Adjacency Table

YIN Ming[①] WANG Wenjie[①] ZHANG Xuanyu[①] JIANG Jijiao[②]

[①](*Institute of Software and Microelectronics*, *Northwestern Polytechnical University*, *Xi'an* 710072, *China*)

[②](*Management School*, *Northwestern Polytechnical University*, *Xi'an* 710072, *China*)

Abstract: To solve the problems of Apriori algorithm and FP-Growth algorithm in the process of mining the maximal frequent itemsets, which refer to inefficient operation, high memory consumption, difficulty in adapting to the process of dense datasets, and affecting the time-effectiveness of large data value mining, this paper proposes a maximal frequent itemsets mining algorithm based on adjacency table. The algorithm only needs to traverse the database once and adopts the hash table to store the adjacency table, which reduces the memory consumption. Theoretical analysis and experimental results show that the algorithm has lower time and space complexity and improves the mining rate of maximal frequent itemsets, especially when dealing with dense datasets.

Key words: Data mining; Frequent itemsets; Apriori; FP-Growth; FP-Tree

1 引言

数据挖掘是从大量的数据中提取潜在且可以表示为可理解的知识的过程[1,2]。关联规则挖掘就是数据挖掘研究的重要内容之一，旨在发掘数集中不同项之间的内在关联[3]。它由两部分组成：(1)挖掘所有项的最大频繁项集，(2)用最大频繁项集产生所有满足置信度阈值的关联规则，其中最大频繁项的挖掘较为重要，它决定规则挖掘的整体性能[3,4]。

最大频繁项集挖掘算法中，最著名的是Agrawal等人[4–6]提出的布尔型关联规则问题和相应的Apriori算法。该算法虽然思路和结构简单，没有复杂的推导，在频繁项集挖掘的过程中，以递归统计为基础不断修剪生成频繁项集，另外利用本身性质而产生候选集的方法在许多情况下缩小了检查的候选规模，使算法效率有一定的提高[3,6]。但在产生最大频繁项集的过程中，需要产生、处理和保存大量的候选集和多次扫描数据库，占用大量的内存空间和系统时间，难以适应海量和稠密数据的挖掘[7–9]。

针对Apriori算法的缺陷，Han等人[9]提出了基于频繁模式树(FP-Tree)产生最大频繁项集的FP-Growth

收稿日期：2018-07-08；改回日期：2019-05-17；网络出版：2019-05-29

*通信作者： 王文杰 wenjie@mail.nwpu.edu.cn

基金项目：教育部人文与社会科学基金(16YJA630068,18YJA630043)，航空科学基金(2016ZG53071)，陕西省自然科学基础研究计划项目(2018JM7008)，陕西省社会科学基金(2018S28)，西北工业大学研究生种子基金(ZZ2018222)

Foundation Items: Ministry of Education Humanities and Social Science Foundation (16YJA630068, 18YJA630043), Aeronautical Science Fund of China (2016ZG53071), Shaanxi Natural Science Basic Research Project (2018JM7008), Shaanxi Social Science Foundation Project (2018S28), Graduate Student Seed Fund Project of Northwestern Polytechnical University (ZZ2018222)

算法[10]。该算法将数据集压缩到FP-Tree中，用FP-Tree映射存储关联信息，最后对该树递归遍历产生最大频繁项集[11–13]。该算法不需要产生候选项，仅需要遍历数据库两次生成FP-Tree，对FP-Tree进行递归挖掘便可产生最大频繁项集，减少了数据库的遍历次数[3,11]。但它需要创建包含所有数据项的FP-Tree，需要占用大量内存，内存消耗与FP-Tree宽度和深度成比例[13]。深度一般是单个事务所有项目数量的最大值，如果数据库中的频繁1-项集数量很大，且内存不能装载所有项目在FP-Tree的映射信息，该算法将不能有效地工作[14]。并且两次扫描数据库，生成和多次递归FP-Tree也使得该算法的空间和时间性能不高。

本文提出了一种基于邻接表最大频繁项集挖掘算法。同时借助了邻接表和哈希表结合的存储方式，以减少数据库的扫描次数和遍历的空间规模，并可以尽早修剪掉小于支持度阈值的项集，避免生成较长最大频繁项集的所有非空子集。该算法充分利用建立的邻接表，只需对原数据库扫描一次，具有时间复杂度低，消耗内存等优点。

2 最大频繁项集挖掘模型

2.1 最大频繁项集

设一个数据项集$I=\{T_1, T_2, T_3, \cdots, T_n\}$, D是一个事务集，每条事务T对应一个数据项的子集，即$T\subseteq I$，对项集X当$X\subseteq T_i$时，称事务T_i包含X项集中项目的个数称为项集的维数或长度，若项集长度为L，称为L-项集[3,7]。项集X出现次数定义为

$$\sigma(X)=|\{T_i|X\subseteq T_i, T_i\in T\}| \tag{1}$$

定义 1 项目集$X\subseteq I$, $Y\subseteq I$，其中$X\cap Y=\varnothing$，则X, Y的关联规则可以表示为$X\to Y$，关联规则的强度可以它的支持度和置信度来度量[3,7]。设非空事务集D的总数量为N, $\sigma(X)$表示D中X项集出现的频度，则 $X\to Y$的支持度(support)：

$$s(X\to Y)=\frac{\sigma(X\cup Y)}{|T|} \tag{2}$$

支持度刻画了项集$X\cup Y$的出现频次。则置信度(confidence):

$$c(X\to Y)=\frac{\sigma(X\cup Y)}{\sigma(X)} \tag{3}$$

置信度可以理解为条件概率$P(Y|X)$，度量在已知事务中包含X同时包含Y的概率[8,10]。对于可靠的最大频繁项集，其支持度与置信度应大于设定的阈值[14,15]。则最大频繁项集挖掘问题，可等价于给定最小支持度阈值min_sup，置信度阈值min_conf，找出所有满足支持度大于等于min_sup，置信度大于等于min_conf条件的项集，该项集即为最大频繁项集。

定义2 为更好地区分数据集的稠密程度，将数据库中的事务数与项目数的比值作为数据集稠密程度的体现，当比值大于50%时，称为稠密数据集，反之为稀疏数据集。

若采用暴力穷举的方式，对数据库中事务数据集的所有项d，不断重复遍历挖掘最大频繁项集数量：

$$\begin{aligned}\sum_{i}^{d} C_d^i \sum_{j}^{d-i} C_{d-i}^j &= \sum_{i}^{d} C_d^i 2^{d-i}-2^d+1\\ &=3^d-2^{d+1}+1\end{aligned} \tag{4}$$

可以看出穷举所有的最大频繁项集，其时间复杂度会达到指数级别，尤其对海量数据缺乏实时性。因此需要时间复杂度更低的算法，实现最大频繁项集的挖掘。

2.2 传统最大频繁项集挖掘算法

Apriori算法产生频繁项集有两个特点：(1)它是逐层算法，即从频繁项1-项集到最长的频繁项集；(2)使用产生及测试策略来挖掘频繁项集，在每次迭代之后，新的候选项集都由前一次迭代发现的频繁项集产生，然后对每个候选项的支持度进行计数，并与最小支持度阈值进行比较[15]。为产生大量的候选频繁k-项集，需要自连接合并频繁$(k-1)$-项集F_{k-1}，确定是否至少有k–2个项相同，假设事务的平均宽度为w，则合并频繁项集的总时间为

$$\sum_{k=2}^{w}(k-2)|C_k|<t<\sum_{k=2}^{w}(k-2)|F_{k-1}|^2 \tag{5}$$

构造Hash树存放候选集，树的最大深度为k，将候选集散列到Hash树的时间复杂度为$O\left(\sum_{k=2}^{w} k|C_k|\right)$在候选集剪枝的过程中，需要检查判断每个$k$-项集的$k$–2个子集是否频繁，候选集剪枝的时间消耗为$O\left(\sum_{k=2}^{w} k(k-2)|C_k|\right)$。可见该算法需要产生、处理、保存大量的候选集，占用大量的内存空间和时间消耗[6,14]。

FP-Growth算法针对Apriori算法的缺点，引入一些数据结构减少I/O次数，实现只需扫描数据库两次，很大程度上降低了时间复杂度[15]。这个数据结构包括3个部分：第1部分是项头表，用于记录所有项集出现的次数，并按照次数的进行降序排列；第2部分是频繁模式树(FP-Tree)，将原始数据集映射到内存中的FP-Tree中，并且仍然保持项目集之间关联的信息；第3部分是节点链表，所有项

头表中的频繁项集都是一个节点链表的头，它依次指向FP-Tree中频繁项集出现的位置[15]。

如表1事务数据集，该算法在第1次扫描数据库后，得到所有的频繁项集的频繁度计数。第2次扫描数据库时，将读到的数据集项删除非频繁项后并按支持度计数降序排列，目的是为构建FP-Tree时，可以尽可能地共用祖先节点，如图1所示。

表 1　事务数据库

TID	项
T100	B, C, E
T200	F, B
T300	C, A, D
T400	D, B, C, A, E
T500	C, E, D
T600	E, F

根据频繁模式树(FP-Tree)便可以得到频繁项集。首先从项头表的底部项依次向上挖掘，对于项头表对应于TP-Tree的每一项，要找到它的条件模式基。条件模式基是以要挖掘的节点作为叶子节点所对应的FP-Tree子树[13,15]。得到这个子树，然后将子树中每个节点的计数设置为叶子节点的计数，并删除计数低于支持度的节点。最后从这个条件模式基，可以递归挖掘得到频繁项集。FP-Growth算法需要扫描两次数据库，将数据库中频繁信息压缩在FP-Tree中，将对事务数据库的挖掘转化为对FP-Tree的挖掘[13]。但对于大规模或者稠密数据集，该算法存在内存和计算的瓶颈，使得算法效率不高。

3　基于邻接表的最大频繁项集挖掘算法

在进行最大频繁项集挖掘的过程中，Apriori算法需要多次遍历数据库，并产生、处理、保存大量的候选集；FP-Growth算法需要两次遍历数据库，对各事务项按频繁度数降序排序，并生成和多次递归FP-Tree。这些都使得该两类算法需要占用大量的内存空间和时间消耗，尤其在处理稠密数据集时，更显得效率低下。

针对以上两类算法的缺陷，本文提出了基于邻接表的最大频繁项集挖掘算法，将每项事务集视为完全图，各项出现的频繁次数视为图中相邻节点的权重。该算法主要包括两个核心步骤：首先遍历数据库，将每项事务集中的各项依次存入带权的邻接表中，该邻接表已保存了所有事务项的关联信息，然后对邻接表进行挖掘。首先按照设置的最小支持度，取邻接表中的顶点项与相邻项形成2-项频繁项集，然后遍历邻接表，以2-项频繁项集为基础不断进行置换对应的顶点项，再进行合并形成更高维的频繁项集，直至没有可以合并的顶点项该算法结束。算法整体过程如图2所示。

3.1　生成邻接表

数据库中每一条事务集中的项认为是相互关联的，组成一个完全图，即每对不同的项之间有一条边相连，该边即表示关联关系，边的权重为频繁度。相互关联的两项，每关联一次其边的权值加1，所有的事务集合并后，便形成一个带权无向图。

本文采用邻接表实现对图的存储，因为构造n个顶点和e条边的无向图，邻接矩阵的时间复杂度为 $O(n\cdot n+e\cdot n)$，其中邻接矩阵初始化消耗的时间为$O(n\cdot n)$。而构造邻接表的时间复杂度为

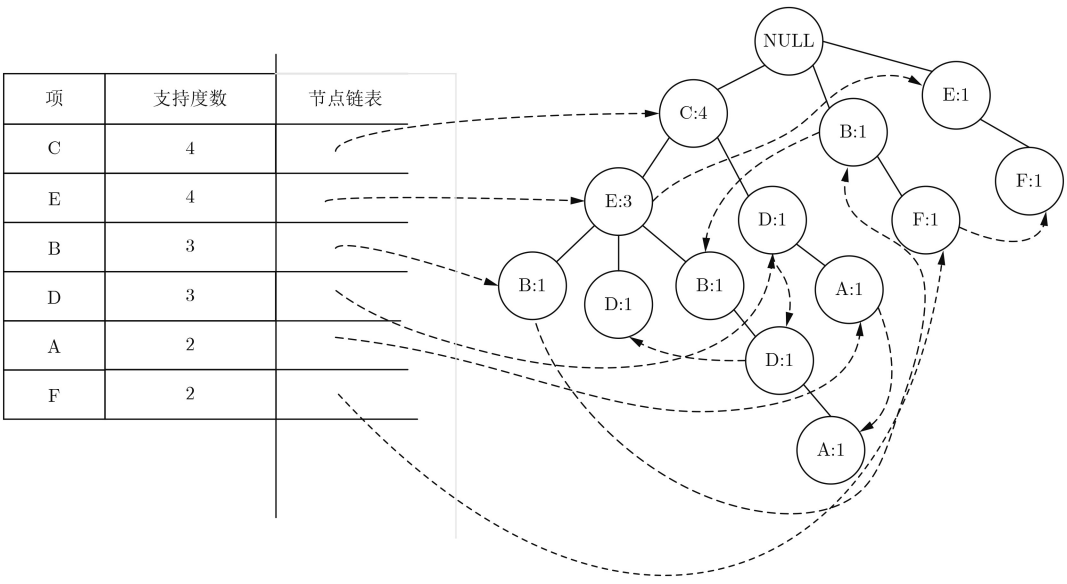

图 1 顶头表与FP-Tree

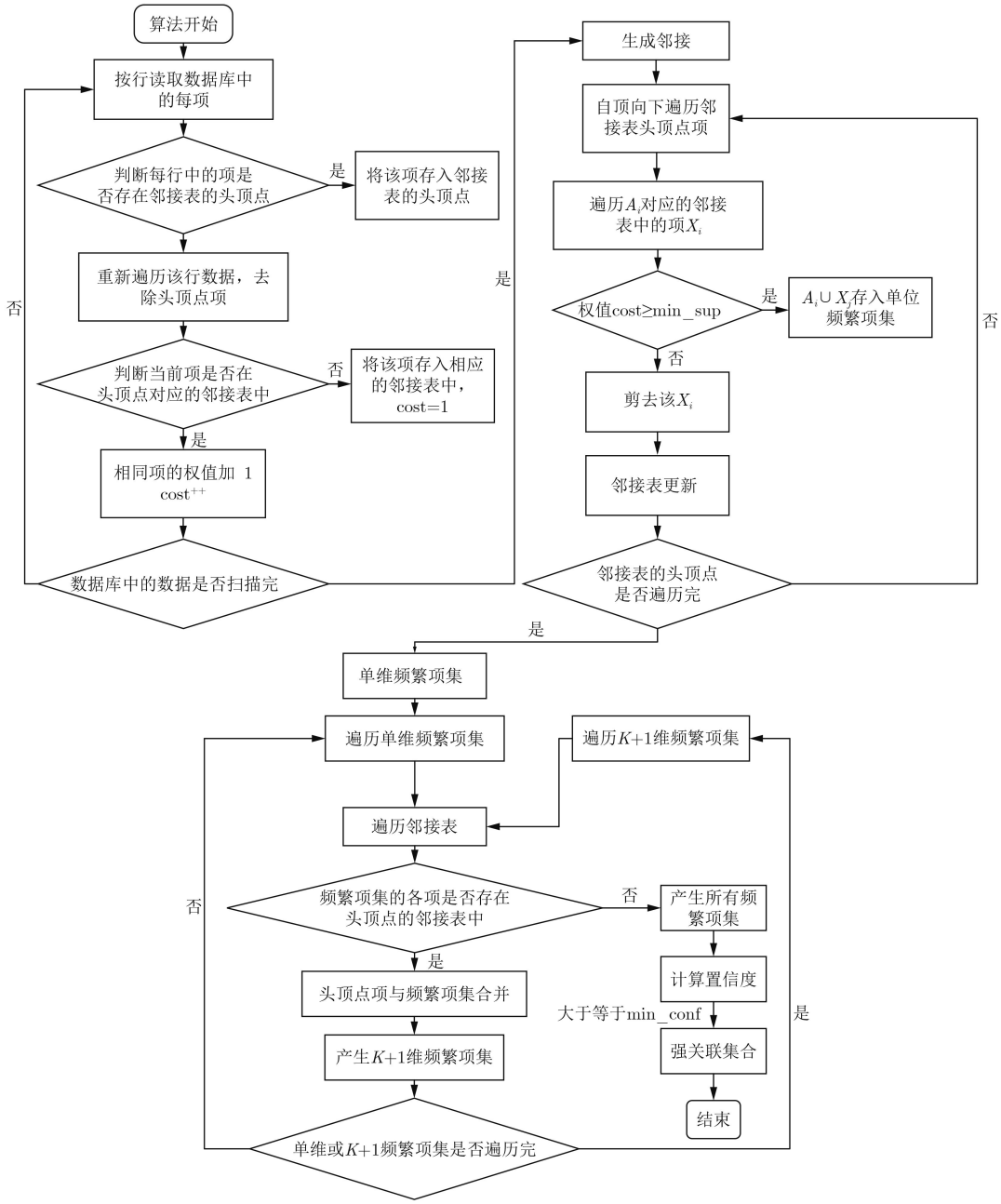

图 2 基于邻接表的最大频繁项集挖掘过程

$O(n+e)$，相对邻接矩阵，采用邻接表，算法在空间和时间整体性能上比较高。该算法只需遍历数据库一次，也不需要大量的排序和内存单元移动，便可生成最终的邻接表。该邻接表可以映射数据库中所有项的关联信息，顶点到每个邻接点的权重保存了相应的最大频繁度计数。以表1中的数据为例，设最小支持度阈值为0.3，生成邻接表如图3。将数据库中的数据项转为邻接表存储，此时邻接表便蕴含了所有数据项之间的关联信息。

由于数据库中的数据项有大量重复的元素，如果对邻接表采用线性存储，则每存储一个元素都要对邻接表进行一次遍历，查找重复的元素。随着数据量的增加，遍历所消耗的时间也就越多，则无法保证频繁项集挖掘的实时性。而哈希表，采用散列存储的方式，将存储的数据以关键字的形式转换为hash值，根据hash值对数据进行操作。因此，采用哈希表对邻接表进行存储，可以大大降低数据存储和查找的时间消耗，使时间复杂度几乎可以视为常数，提高频繁项集挖掘的速率。

遍历数据库生成邻接表算法描述如下：

(1) 定义图的存储结构HashMap<node_list, Edge>, node_list存储头结点，Edge存储邻接点的信息。

(2) Struct Edge{　//边结点的定义

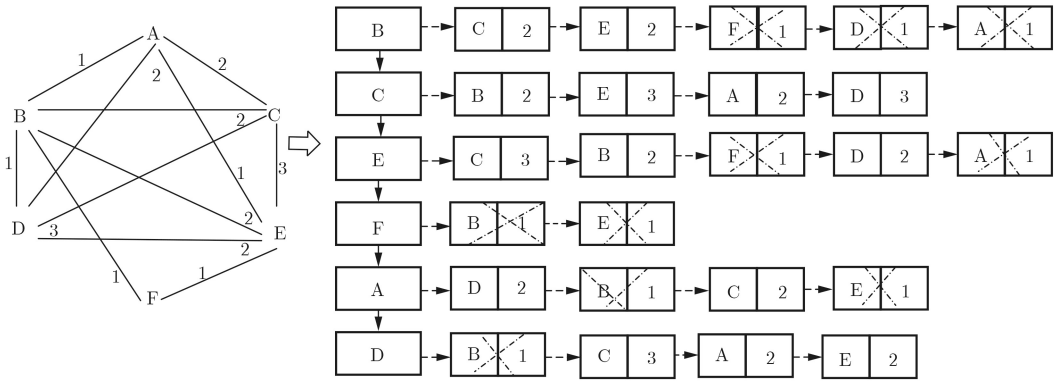

图 3 由数据集生成邻接表

String node　//边的另一顶点的位置

int cost　　//边上的权值，即项的频繁度

Edge<String, int> *link　　//下一条边链指针

} Edge

(3) readDataset(文件路径)；//找到相应的文件，按行读取数据集

while(按行读取数据且不为空){

//将每行数据去除空格后分割存入数组，作为参数传入构造邻接表的函数。}//end of while

(4) setEdge(node[])；//将从文件中读取的数据，添加到邻接表中

for (依次读取数组node[]中的项X_i){

//判断邻接表HashMap<node_list, Edge>中的顶点项是否含有X_i，若$X_i \in$ node_list，则继续判断node[]中除去项X_i后的其余各项$X_{j\neq i}$是否已存在于Edge中，如果已经存在，则边的权值cost自增1，否则cost值设置为1，并将$X_{j\neq i}$添加到Edge中。若$X_i \notin$ node_list，则直接将X_i添加到node_list中。} // end of for

3.2 最大频繁项集挖掘

在经过一次遍历数据库后得到的邻接表，保存了所有事物项集的频繁信息和关联信息。在进行最大频繁项集挖掘时，充分利用该邻接和频繁项集的两个性质，即如果X是频繁项集，则X的任意非空子集都是频繁项集；如果X是非频繁项集，则X的任意超集都是非频繁相集。首先，遍历邻接表的各顶点项A_i和对应邻接点项$\{X_i|X_i \in \text{Edge}_i\}$，如果$X_i$的权值cost即频繁度数，满足$\text{cost} \geq \min_\text{sup}$，则取$A_i \cup X_i$形成2-项频繁项集，否则剪去$X_i$。然后根据初始得到的2-项频繁项集去置换顶点项$A_j$，即如果满足$(A_i \cup X_i) \subseteq \text{Edge}_i$，取该邻接点集合对应的头顶点$A_j$，令$A_i \cup X_i \cup A_j$便得到更高一维的频繁项集。以此类推，利用$k$-项频繁项集去置换顶点项，取并集形成$k$+1项频繁项集，直到置换不出顶点，该算法结束，可得到所有的最大频繁项集。

算法描述如下。

(1)用集合HashMap<node_list, int> frequent存储最大频繁项集，min_sup为最小支持度阈值，GraphEdge为遍历数据库各项目集后产生的邻接表。

(2)setFrequent(min_sup)；//遍历邻接表，根据最小支持度阈值获取2-项频繁项集，并剪去小于最小支持度阈值的项，以便在获取多维频繁项集时减少遍历个数。

for each$\{A_i|A_i \in$ node_list$\}$ do//自顶向下依次获取顶点项A_i。

{ for (从左到右依次遍历顶点项A_i对应的邻接点$X_i \in$ Edge)

if邻接点项X_i的cost<min_sup，则从邻接表中删除该顶点对应的X_i项。

else if $(A_i \cup X_i) \notin$ frequent，将$A_i \cup X_i$添加到frequent中，直到A_i的所有邻接点X_i遍历完。$A_i \cup X_i$与$X_i \cup A_i$是相同的集合，frequent中只存一份。}//end of for

//当GraphEdge中的所有顶点项集A_i遍历完，2-项频繁项集则全部找出。}//end of for each

(3)getPattern(frequent)//以开始挖掘的2-项频繁项集为基础，置换多维频繁项集。

for each$\{I_i|I_i \in$ frequent$\}$ do //依次取frequent集合中已存在的频繁项集。

for (自顶向下依次遍历邻接表GraphEdge中所有的顶点以及所有顶点的邻接点){

if $I_i \in$ Edge，取该邻接点的顶点A_i。

if $A_i \cup I_i \notin$ frequent，将$A_i \cup I_i$存入frequent中，直到frequent中的频繁项集遍历完，并更新frequent重复步骤(3)，直到再找不到$I_i \in$ Edge，则成功找出所有的频繁项集，程序结束。

4 算法性能分析

4.1 时间性能分析

在进行最大频繁项集挖掘的过程中，Apriori算法需要多次遍历数据库，并产生、处理和保存大量的候选集，而FP-Growth算法只需要两次遍历数据库，且不产生大量的候选集，因此FP-Growth算法在时间性能上优于Apriori算法。本文先采用理论分析，对FP-Growth算法与本文提出的基于邻接表的最大频繁项集挖掘算法进行时间性能上的比较，以下是用于性能分析的符号：

n：整个数据库中事务的数量；n_1：原始数据库对应FP-Tree中的项目集数量，即FP-Tree中的叶子节点数量；n_2：每条事务集T_i的平均项目数量；tr_i：从原始数据库中读取事务项i的时间；tl_i: FP-Growth算法统计数据库中的每项频率的时间；El_i：基于邻接表的算法统计数据库中的每项频率的时间；ts_i: FP-Growth算法按顶头表顺序给每条事务集排序消耗的时间；tin_i：将每项数据插入FP-Tree中的时间；Ein_i：将每项数据插入邻接表的时间；tf_i：从FP-Tree中得到频繁项集的时间；Ef_i：从邻接表中得到频繁项集的时间；T_{FP}: FP-Growth算法从原始数据库到最终找出所有频繁项集的时间；T_{E}：基于邻接表的算法从原始数据库到最终找出所有频繁项集的时间。

$$T_{\mathrm{FP}}=\sum_{i=1}^{n}(\mathrm{tr}_i+\mathrm{tl}_i+\mathrm{tin}_i+\mathrm{tf}_i) \tag{6}$$

当FP-Tree接近二叉树时，FP-Growth算法递归的时间最少，则式(6)可变为

$$T_{\mathrm{FP}}=\sum_{i=1}^{n}(\mathrm{tr}_i+\mathrm{tl}_i+\mathrm{ts}_i)+\sum_{i=1}^{n}\log_2(i)+n_1\log_2(n_1) \tag{7}$$

$$T_{\mathrm{E}}=\sum_{i=1}^{n}(\mathrm{tr}_i+\mathrm{Ein}_i+\mathrm{Ef}_i+\mathrm{El}_i) \tag{8}$$

本文借助哈希表辅助存储邻接表，因此在最坏遍历的情况下式(8)可接近于

$$T_{\mathrm{E}}=\sum_{i=1}^{n}(\mathrm{tr}_i+\mathrm{Ef}_i)+O(n)+n_2O(n_2) \tag{9}$$

在构建FP-Tree时，需要对每条事务集按照顶头表的顺序排序，并且统计每项的频率也要遍历顶头表。而在构建邻接表时，不需要对每条事务集排序，在统计每项的频率时，只简单对哈希表遍历，所以可得

$$\sum_{i=1}^{n}(\mathrm{tr}_i+\mathrm{tl}_i+\mathrm{ts}_i)>\sum_{i=1}^{n}(\mathrm{tr}_i+\mathrm{El}_i) \tag{10}$$

$$\sum_{i=1}^{n}\log_2(i)=\log_2(n!) \tag{11}$$

由斯特林公式和积分放缩法可知

$$\log_2 n!>\log_2\left(n^{n+\frac{1}{2}}\mathrm{e}^{-n}\right) \tag{12}$$

每条事务集的平均项目数量小于FP-Tree中的叶子节点数量，结合式(3)，式(12)可得

$$O(n)+n_2O(n_2)<\log_2\left(n^{n+\frac{1}{2}}\mathrm{e}^{-n}\right)+n_1\log_2(n_1) \tag{13}$$

由式(13)可知本文提出的基于邻接表的算法时间性能上优于FP-Growth算法。

4.2 空间性能分析

Apriori算法在挖掘最大频繁项集的过程中，会产生大量的候选项集。若频繁1-项集的个数为10000，则Apriori算法需要产生候选2-项集的个数会超过10 M；若挖掘频繁项集的长度为100，则必将产生大于2100个的候选项集。尽管有剪枝操作，但是所需的候选集数量可能依然非常大。虽然FP-Growth算法，能够在不产生候选集的情况下挖掘出所有的频繁项集，在空间性能上优于Apriori算法，但依然存在不足[11]。

FP-Growth算法采用递归调用自身的方法来解决划分得到的子问题，因此，每当要解决一个子问题时，就要为其建立新的条件FP-Tree，并且这个过程是递归进行的，所以在一个子问题得到完全解决之前，其生成的所有条件FP-Tree都要完整地保存在栈内存中。各个条件FP-Tree它们之间相互独立，因此会占用大量存储空间，尤其对于大型密集型数据，体现尤其明显。据统计，FP-Growth算法生成的条件FP-Tree的数目与它产生的频繁项集的数目处于同一个数量级。随着越来越多递归生成的条件FP-Tree充斥内存，挖掘工作很可能无法继续在内存中进行。并且递归算法本身的执行就需要辅助空间，一般递归算法空间复杂度等于递归的深度N与每次递归所需辅助空间的乘积。假如数据量很大，递归所需要消耗的空间也就越多。

而本文提出的基于邻接表的算法，在挖掘频繁项集的过程中，只需要消耗将数据库读存为邻接表所消耗的空间，而该空间大小与FP-Tree大小近似。不需要创建条件FP-Tree，顶头表，以及频繁的递归操作，在空间性能上优于FP-Growth算法以及Apriori算法。

5 实验及结果分析

实验1因为支持度在一定程度上反映了最大频繁项集挖掘算法的准确性，所以该实验应用基于本

文提出的算法、FP-Growth算法以及Apriori算法分别对表1中的数据进行最大频繁项集挖掘，对比分析所有挖掘出的最大频繁项集的支持度如表2所示。

表 2　3种算法的最大频繁项集挖掘结果

Apriori	FP-Growth	基于邻接表的算法	支持度
(A,C:2)	(A,C:2)	(C,A:2)	0.3
(D,A:2)	(A,D:2)	(A,D:2)	0.3
(B,C:2)	(B,C:2)	(B,C:2)	0.3
(E,B:2)	(B,E:2)	(B,E:2)	0.3
(D,C:3)	(D,C:3)	(C,D:3)	0.5
(C,E:3)	(E,C:3)	(C,E:3)	0.5
(D,E:2)	(D,E:2)	(E,D:2)	0.3
(D,A,C:2)	(A,C,D:2)	(C,A,D:2)	0.3
(E,C,B:2)	(B,C,E:2)	(B,C,E:2)	0.3
(D,C,E:2)	(D,C,E:2)	(C,D,E:2)	0.3

实验2为了验证本文算法的时间性能，利用购物车商品关联竞赛数据(Kaggle竞赛)，该数据集包含了100多万条数据，近400种不同的商品。实验首先对数据进行初步处理，将数据划分为稠密数据集与稀疏数据集两类，然后在相同的实验机与条件下，应用本文算法、FP-Growth算法以及Apriori算法分别处理两类数据，从时间性能上进行对比。

为了反映处理稀疏数据集，3种最大频繁项集挖掘算法在不同数据量情况下的计算速率和变化趋势。本文选取的稀疏数据集平均长度为10，并从中随机选取6个部分(1×10^5, 2×10^5, 3×10^5, 4×10^5, 6×10^5)作为测试数据。在相同的支持度下对算法加载不同事务数量进行测试，测试结果如图4(a)所示。为反映3种算法在处理稠密数据集时，时间性能随事务数量的变化趋势，本文选取平均长度为25的相对稠密数据集。考虑处理稠密数据的计算时间不易收集，本文将该稠密数据化为数据量比较小的6个部分(5.0×10^3, 1.0×10^4, 1.5×10^4, 2.0×10^4, 2.5×10^4, 3.0×10^4)。同样，在相同的支持度下对算法进行测试，测试结果如图4(b)。

测试过滤掉无效事务后3种算法的时间性能对比，本文采用长度为10的稀疏事务数据和长度为25的相对稠密事务数据，对两类数据集都加载全部事务，每次选择不同的支持度计数。稀疏事务数据集数量为3.0×10^5，稠密事务数据集数量为2.0×10^4，实验结果如图5(a)和图5(b)所示。可以看出随着支持度计数的增加，有效事务变得越来越少，本文提出的基于邻接表的算法变化比较平缓，依然相对优越。当支持度计数增加到一定值时，这3种算法的执行时间趋于一个稳定值，这部分时间主要是遍历数据库，且不会发生变化。

通过实验结果可知，本文提出的基于邻接表的算法能有效挖掘数据库中的最大频繁项集。一方面，在相同的最小支持度阈值条件下处理相同的数据，3种算法的时间开销都随着数据量的递增而增

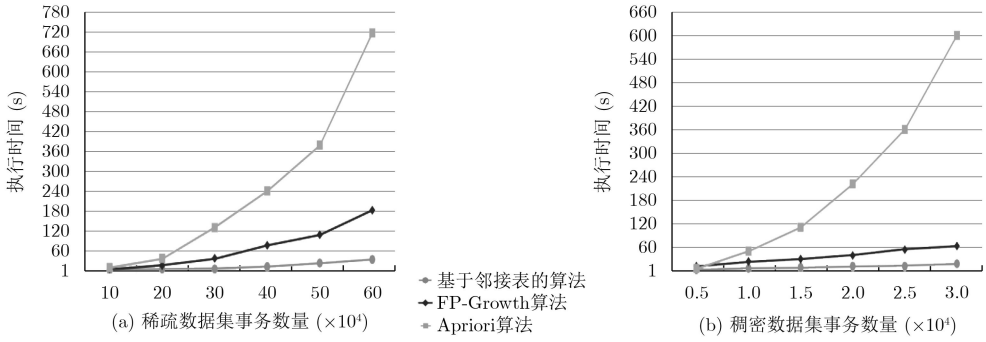

图 4 处理稀疏与稠密数据集不同事务数量的效率对比图

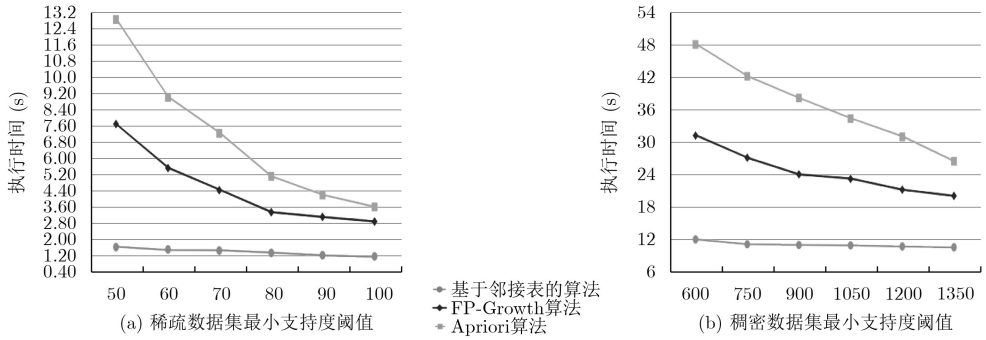

图 5 处理稀疏与稠密数据集不同支持度计数的效率对比图

加，而本文提出的算法有较好的时间性能，且时间开销的变化率比较缓慢，数据量越大优势越加明显。尤其是在处理稠密数据集时，本文提出的算法优势更加突出。另一方面，本文算法只需遍历数据库一次，在处理相同数量的事务数据集下，无论稀疏数据还是稠密数据，随着支持度计数的增加，该算法的执行时间变化也比较平缓，接近于一次遍历数据库所消耗的时间。实验证明本文提出的基于邻接表的最大频繁项集挖掘算法，能够充分利用计算机的性能和内存空间，高效地完成大规模数据的频繁项集挖掘。

6 结论

本文提出一种基于邻接表的最大频繁项集挖掘算法，该算法只需遍历数据库一次，大大减少了I/O操作，同时对邻接表的存储采用哈希表，提升了数据遍历的速率。虽然该算法对邻接表的存储需要增加额外存储空间，但不需要频繁地操作数据库和进行递归挖掘，也不需要产生和存储大量候选集，以及建立顶头表和大量复杂的排序操作。很大程度地降低了算法的时间消耗和空间消耗，尤其在处理稠密数据集时，该算法表现出高效的性能。理论分析和实验结果表明，本文算法较Apriori算法以及FP-Growth算法具有一定的优越性。

参 考 文 献

[1] WU Xindong, KUMAR V, QUINLAN J R, *et al*. Top 10 algorithms in data mining[J]. *Knowledge and Information Systems*, 2008, 14(1): 1–37. doi: 10.1007/s10115-007-0114-2.

[2] FASIH H and SHAHRAKI M H N. Incremental mining maximal frequent patterns from univariate uncertain data[J]. *Knowledge-Based Systems*, 2018, 152: 40–50. doi: 10.1016/j.knosys.2018.04.001.

[3] 易彤, 徐宝文, 吴方君. 一种基于FP树的挖掘关联规则的增量更新算法[J]. 计算机学报, 2004, 27(5): 703–710. doi: 10.3321/j.issn:0254-4164.2004.05.017.
YI Tong, XU Baowen, and WU Fangjun. A FP-tree based incremental updating algorithm for mining association rules[J]. *Chinese Journal of Computers*, 2004, 27(5): 703–710. doi: 10.3321/j.issn:0254-4164.2004.05.017.

[4] 陈安龙, 唐常杰, 陶宏才, 等. 基于极大团和FP-Tree的挖掘关联规则的改进算法[J]. 软件学报, 2004, 15(8): 1198–1207. doi: 10.13328/j.cnki.jos.2004.08.012.
CHEN Anlong, TANG Changjie, TAO Hongcai, *et al*. An improved algorithm based on maximum clique and FP-Tree for mining association rules[J]. *Journal of Software*, 2004, 15(8): 1198–1207. doi: 10.13328/j.cnki.jos.2004.08.012.

[5] BUI H, VO B, NGUYEN H, *et al*. A weighted N-list-based method for mining frequent weighted itemsets[J]. *Expert Systems with Applications*, 2018, 96: 388–405. doi: 10.1016/j.eswa.2017.10.039.

[6] AGRAWAL R and SRIKANT R. Fast algorithms for mining association rules in large databases[C]. The 20th International Conference on Very Large Data Bases, San Francisco, 1994: 487–499.

[7] APILETTI D, BARALIS E, CERQUITELLI T, *et al*. Frequent itemsets mining for big data: A comparative analysis[J]. *Big Data Research*, 2017, 9: 67–83. doi: 10.1016/j.bdr.2017.06.006.

[8] 肖波, 徐前方, 蔺志青, 等. 可信关联规则及其基于极大团的挖掘算法[J]. 软件学报, 2008, 19(10): 2597–2610.
XIAO Bo, XU Qianfang, LIN Zhiqing, *et al*. Credible association rule and its mining algorithm based on maximum clique[J]. *Journal of Software*, 2008, 19(10): 2597–2610.

[9] HAN Jiawei, PEI Jian, and YIN Yiwen. Mining frequent patterns without candidate generation[C]. The 2000 ACM SIGMOD International Conference on Management of Data, Dallas, 2000: 1–12. doi: 10.1145/342009.335372.

[10] KARIM 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. doi: 10.1016/j.ins.2017.11.064.

[11] 吉根林, 杨明, 宋余庆, 等. 最大频繁项目集的快速更新[J]. 计算机学报, 2005, 28(1): 128–135. doi: 10.3321/j.issn:0254-4164.2005.01.016.
JI Genlin, YANG Ming, SONG Yuqing, *et al*. Fast updating maximum frequent itemsets[J]. *Chinese Journal of Computers*, 2005, 28(1): 128–135. doi: 10.3321/j.issn:0254-4164.2005.01.016.

[12] 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. doi: 10.13328/j.cnki.jos.2003.09.012.
SONG Yuqing, ZHU Yuquan, SUN Zhihui, *et al*. An algorithm and its updating algorithm based on FP-Tree for mining maximum frequent itemsets[J]. *Journal of Software*, 2003, 14(9): 1586–1592. doi: 10.13328/j.cnki.jos.2003.09.012.

[13] VIJAYARANI S and SHARMILA S. Comparative analysis of association rule mining algorithms[C]. 2016 International Conference on Inventive Computation Technologies, Coimbatore, 2016: 1–6. doi: 10.1109/INVENTIVE.2016.7830203.

[14] LIU Li, YU Shuo, WEI Xiang, *et al*. An improved Apriori-based algorithm for friends recommendation in microblog[J]. *International Journal of Communication Systems*, 2018, 31(2): e3453. doi: 10.1002/dac.3453.

[15] 连志春, 伊凤新. 一种改进的频繁模式树生长算法[J]. 应用科技, 2008, 35(6): 47–51. doi: 10.3969/j.issn.1009-671X.2008.06.012.
LIAM Zhichun and YI Fengxin. An improved frequent pattern tree growth algorithm[J]. *Applied Science and Technology*, 2008, 35(6): 47–51. doi: 10.3969/j.issn.1009-671X.2008.06.012.

殷 茗：女，1978年生，博士，副教授，主要研究方向为企业信息化、信息管理与信息系统、电子服务.

王文杰：男，1992年生，硕士，主要研究方向为数据挖掘、机器学习.

张煊宇：男，1995年生，硕士，主要研究方向为信息管理与信息系统.

姜继娇：男，1979年生，博士，副教授，主要研究方向为行为金融与风险管理.