Computer Science and Application 计算机科学与应用, 2018, 8(8), 1195-1206
Published Online August 2018 in Hans. http://www.hanspub.org/journal/csa
https://doi.org/10.12677/csa.2018.88131

Hans汉斯

An Improved Eclat Algorithm Based on Pruning Optimization and Indexing Intersection

Shixin Lv, Jie Huang

Information Engineering University, Zhengzhou Henan
Email: 492593799@qq.com, huangjie0922@126.com

Received: Jul. 29th, 2018; accepted: Aug. 13th, 2018; published: Aug. 20th, 2018

Abstract

In view of the problem of large number of candidate sets and low intersecting efficiency in the existing Eclat algorithm, an improved Eclat algorithm based on pruning optimization and index intersection is proposed. First, according to the properties of frequent itemsets, a candidate set optimization strategy based on pre pruning and post pruning is adopted, and that is to use pre pruning technology to tailor the number of itemsets to be linked to reduce itemset join operations, and use the transcendental property to prune the linked itemsets. Then a Boolean array intersection method based on transaction index is proposed, that is, to set and retrieve Boolean arrays by using the transaction identifier as an index to obtain the support count of the item set. Finally, the effectiveness of the method is tested on the classical dataset by designing contrast experiments. Experiments show that this method can effectively compress the size of the candidate sets and improve the efficiency of intersection calculation, especially in the case of small support threshold and large number of transactions, and the efficiency of the algorithm has been greatly improved.

Keywords

Association Rules, Eclat, Pruning Optimization, Intersection Operation, Operating Efficiency

基于剪枝优化与索引求交的改进Eclat算法

吕世鑫，黄　洁

信息工程大学，河南　郑州
Email: 492593799@qq.com, huangjie0922@126.com

收稿日期：2018年7月29日；录用日期：2018年8月13日；发布日期：2018年8月20日

文章引用：吕世鑫，黄洁．基于剪枝优化与索引求交的改进 Eclat 算法[J]．计算机科学与应用, 2018, 8(8): 1195-1206.
DOI: 10.12677/csa.2018.88131

摘　要

针对现有Eclat算法中普遍存在的候选集规模大、求交效率低的问题，提出了基于剪枝优化和索引求交的改进Eclat算法。首先根据频繁集的性质采用预剪枝和后剪枝相结合的候选集优化策略，即利用预剪枝技术裁剪待连接的项集数量以减少项集连接操作，同时利用先验性质对连接后的项集进行后剪枝处理；接着提出了一种基于事务索引的布尔数组求交方法，即通过将事务标识作为索引来设置并检索布尔数组，以获得项集支持度计数；最后通过设计对比实验，在经典数据集上测试该方法的有效性。实验表明，通过该方法能够有效压缩候选集规模，改善求交计算效率，特别是在支持度阈值小、事务数规模大的情况下，算法的运行效率得到了明显的提升。

关键词

关联规则，Eclat算法，剪枝优化，求交运算，运行效率

1. 引言

现如今，通过使用数据挖掘技术，我们可以从海量数据中挖掘有趣的信息。其中，关联规则挖掘是数据挖掘技术中较为重要的一种手段。它一般分为两个步骤：一是依据事先设定的支持度阈值找出所有符合条件的频繁项集；二是依据频繁项集及给定的置信度阈值产生关联规则[1]。其中，挖掘算法的性能主要取决于频繁项集的生成，因此识别或发现所有频繁项目集是关联规则挖掘算法的核心。随着数据挖掘技术理论研究的深入，各类关联规则挖掘算法也在不断地涌现。传统的算法主要包括 Apriori 算法、FP-growth 算法以及 Eclat 算法，后续的大部分算法都是在这三类算法的基础上进行相应的优化和改进。

Apriori 算法[2]使用一种称为逐层搜索的迭代方法，通过项目集元素数目的不断增长来逐步完成频繁项集的发现，核心思想是通过候选集生成和情节的向下封闭检测两个阶段来挖掘频繁项集，该算法设计思想简单，易于实现，但是产生了大量候选集，同时需要多次对事务库进行扫描，计算耗时过长；FP-growth 算法[3]使用一种称为频繁模式增长的方法，采取分治策略，将代表频繁项集的数据库压缩到一棵频繁模式树(FP 树)上，然后把这种压缩后的数据库划分成条件数据库，每个数据库关联一个频繁项或“模式段”，并分别挖掘每个条件数据库，这种方法可以显著地压缩被搜索的数据集的大小，该算法只扫描事务库两次，且无需产生候选集，相比 Apriori 算法性能有显著的提高，但由于所有项集都压缩在一棵树上，对内存要求较高，且递归算法设计复杂；Eclat 算法[4]的核心思想是将水平数据库转换成垂直数据库，然后将项集的 TID_set 进行交运算来得到项集的支持度，该算法由于只扫描一次事务库，且项集支持度是通过交运算得到的，大大减少了计算时间，但 Eclat 算法存在搜索空间大、连接操作频繁、求交运算耗时的问题。

本文选取上述几类算法中性能较好的 Eclat 算法，结合近年来该算法的相关研究，针对其普遍存在的问题和不足，从候选集的生成和支持度的计算两个方面对其加以优化和改进，提出了前后剪枝相结合的候选集优化策略以及利用数组索引取值计数的求交运算方法，以达到提高 Eclat 算法效率的目的。

2. 关联规则相关概念

设 D 是事务数据库，T 是一个事务，$T \subseteq D$。I 是一个项目集，项目集中项目的数量为 I 的长度。设 X 是项目集 I 的一个子集($X \subseteq I$)，其中包含 k 个项目，则称 X 为 k 项集。每一个事务 T 都有唯一的标识符(TID)并包含一个项目集。项集 X 的支持度，记为 Support(X)，它表示在数据库 D 中包含项集 X 的事务数与总事务数之比。如果 X 的支持度大于等于一个预先设置的支持度阈值(最小支持度，记为 min_sup)，则称 X 为频繁项集[5]。

关联规则用形如 $X \Rightarrow Y$ 的蕴含式来表达，它表示两个项目集 X 和 Y 之间的某种关系，其中 $X \subseteq I$，$Y \subseteq I$，$X \cap Y = \varnothing$。$X \Rightarrow Y$ 的支持度，记为 $\text{Support}(X \Rightarrow Y)$，它表示在 D 中同时包含项集 X 和 Y 的事务数占事务总数的百分比，即 $P(X \cup Y)$；$X \Rightarrow Y$ 的置信度，记为 $\text{Confidence}(X \Rightarrow Y)$，它表示在 D 中同时包含项集 X 和 Y 的事务数与包含 X 的事务数之比，或者说，在包含项目集 X 的所有事务中含有项目集 Y 的事务数所占的百分比，即 $P(Y|X)$。如果 $\text{Support}(X \Rightarrow Y)$ 大于等于 min_sup，同时 $\text{Confidence}(X \Rightarrow Y)$ 大于等于 min_conf (预先设置的置信度阈值，即最小置信度)，则称 $X \Rightarrow Y$ 是强关联规则[6]。

3. Eclat 算法及改进算法

3.1. Eclat 算法

3.1.1. 算法主要思想

Zaki 在 2000 年提出的 Eclat 算法是基于深度优先搜索的策略，采用垂直数据格式、等价类技术、交集运算等等。Eclat 算法的主要步骤如下：扫描数据库以获得所有频繁 1-项集；从频繁 1-项集开始生成候选 2-项集，然后通过筛选掉非频繁的候选项集来获得所有频繁 2-项集；从频繁 2-项集生成候选 3-项集，然后通过筛选掉非频繁的候选项集来获得所有频繁 3-项集；重复上述步骤，直到无法生成候选项集为止。上述步骤概括起来包括两个阶段：一是通过自连接产生候选项集；二是通过求交运算获得支持度。

1) 产生候选集

与 Apriori 算法一样，Eclat 算法也采用连接操作来产生候选项目集，即通过连接两个 k-项目集生成(k + 1)-项目集。两个 k-项集连接的条件是它们各自的前 k-1 个项必须相同。例如，有两个 3-项集：$l_{31} = \{I_1, I_2, I_3\}$ 和 $l_{32} = \{I_1, I_2, I_4\}$，它们各自的前两项都为$\{I_1, I_2\}$，因此 l_{31} 与 l_{32} 能连接生成一个 4-项集：$l_4 = \{I_1, I_2, I_3, I_4\}$。Eclat 算法通过使用等价类[7]的概念将搜索空间划分为多个不重叠的子空间，有相同前缀的项目集被划分为同一个类，而候选项目集的生成只在同一个类里操作。这种等价类的技术显然可以提高候选项目集生成的效率，并减少项目的占用内存。

2) 求交运算

Eclat 算法求交运算是通过逐项比较事务 tid 的方式进行的。设 l_{21} 和 l_{22} 是两个项目集，$\text{TID\_set}(l_{21}) = \{1,3,5,6\}$，$\text{TID\_set}(l_{22}) = \{3,4,5,6\}$。假设 l_3 为 l_{21} 与 l_{22} 连接后生成的，$\text{Support}(l_3) = S$，$\text{TID\_set}(l_3) = \text{TID\_set}(l_{21}) \cap \text{TID\_set}(l_{22})$。设 P_1 和 P_2 是两个分别指向 TID_set(l_{21})和 TID_set(l_{22})各自第一个 TID 的指针。获取 l_{21} 和 l_{22} 交集的过程如图 1 所示。由于生成(k + 1)-项集过程和两个 k-项集求交的过程是同时进行的，因此可以立即得到(k + 1)-项集的支持度。

3.1.2. 算法实现过程

下面以具体示例阐述算法的实现过程。给定表 1 所示垂直数据格式的事务库，假设支持度阈值为 0.5，对表中数据进行频繁项集的挖掘，具体演示过程如图 2 所示。在图 2 中，每个椭圆内部包含了项集、项集支持度计数以及项集对应的事务集，其中粗体标注的椭圆为频繁项集，其余椭圆为候选非频繁项集。

图 2 显示，对于表 1 中的数据，共生成了 20 个候选项目集(除去 1-项集)，执行了 20 次连接操作和求交运算，产生了 9 个频繁项集(除去频繁 1-项集)。

3.1.3. 算法主要缺陷

尽管 Eclat 算法的效率很高，它仍然有以下一些缺陷：1) 由于候选集的产生是在等价类中完成的，没有像 Apriori 算法一样利用向下封闭性质[8]对候选集进行剪枝优化操作，因此候选项目集的数量超过了 Apriori 算法。2) 项集的支持度是通过交运算获得的，相比 Apriori 算法通过扫描事务库获取支持度的方式在效率上有很大提高，但是当项目集包含的事务集规模很大时，逐项的求交运算导致效率依然不尽人意。为此，近年来许多人开始关注 Eclat 算法并提出了一些改进算法。

3.1.4. 现有改进算法

Eclat_Diffsets 算法[9]采用布尔矩阵与差集技术来提高交集计算效率并减少内存占用，该算法采用布尔矩阵存储 itemset 和 TID_set，并通过二进制运算来计算交集，这种方式能明显改善支持度获取效率；熊忠阳等人提出的 HEclat 算法[10]采用散列布尔矩阵来改善交集计算，该算法使用散列布尔矩阵来存储

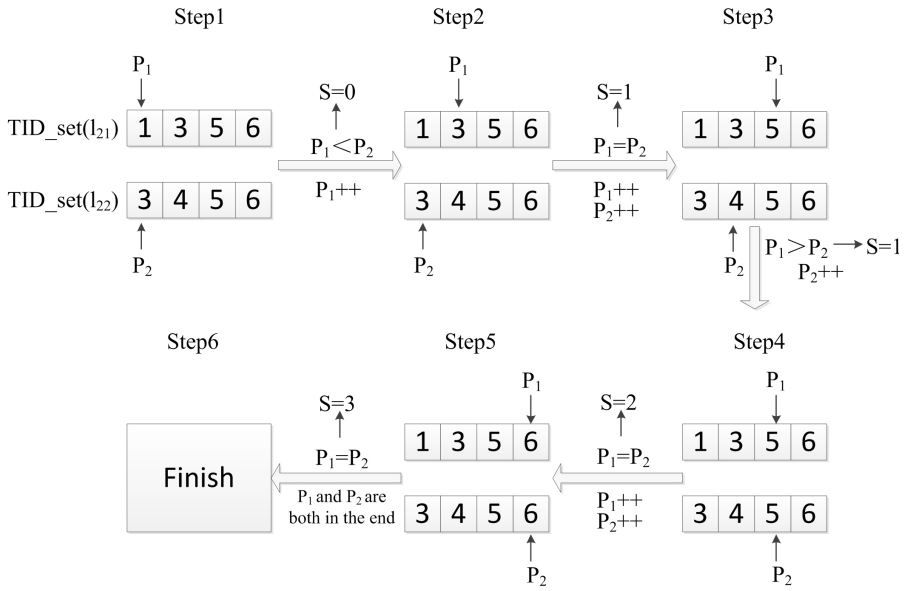

Figure 1. Example of intersection operation process in Eclat algorithm
图 1. Eclat 算法求交运算过程示例

Table 1. Vertical data format of a database
表 1. 数据库的垂直数据格式

Item	TID_set
I_0	1, 2, 6
I_1	2, 3, 4, 6
I_2	4, 5
I_3	2, 3, 4, 6
I_4	1, 2, 3, 5
I_5	3, 4, 5, 6
I_6	2, 3, 4, 5

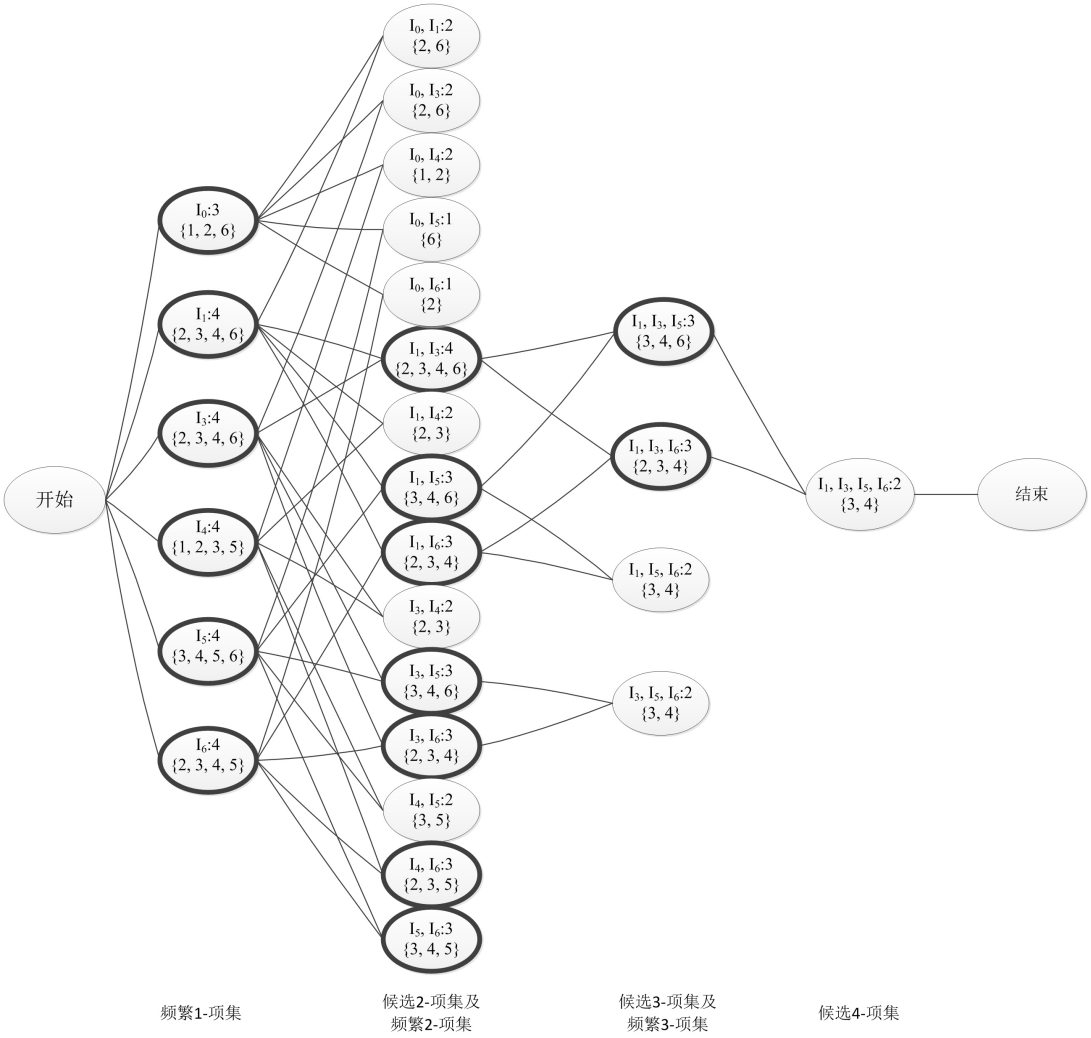

Figure 2. A demonstration of the process of mining frequent itemsets by Eclat algorithm
图 2. Eclat 算法挖掘频繁项集过程演示

项目集的 TID_set，当计算两个项目集的交集时，只需要在两个布尔矩阵上使用位“与”操作，散列布尔矩阵只有在数据库的事务数规模不是很大的时候才能提高效率；Eclat_opt 算法[11]则采用双层哈希表技术加快候选子集的搜索速度，通过项集集合划分链表技术减少项集连接的比较判断，并利用事务标识(tid)失去阈值技术加快交集计算的速度，这些技术能够对候选 3-项集进行充分剪枝，减小搜索空间的同时缩短了生成候选集的时间，此外提高了交集计算的速度。通常情况下，Eclat_opt 算法比其他 Eclat 改进算法更有效。现有的改进算法虽然能够在不同程度上改善原始 Eclat 算法的效率，但是这些算法仍然有一些缺陷，比如大量无用的候选集产生，耗时的求交运算等。

3.2. 改进的 Eclat 算法

针对 Eclat 算法存在的缺陷，本节从候选集的生成和支持度的计算两个方面分别对原始算法进行优化和改进，其具体策略如下：

3.2.1. 候选集生成的优化

为防止大量候选集的产生而导致过多的求交运算，需要在候选集的数量上进行进一步的优化压缩。

为此，本文借助 Apriori 算法中候选集剪枝优化的思想，采用两种剪枝策略相结合的方式对 Eclat 算法中的候选集进行优化处理。一是在连接生成候选集 C_k 之前，对 L_{k-1} 进行预先剪枝，筛选掉那些在连接后一定是非频繁的项集，以便省去重复的连接操作，即预剪枝优化；二是在连接生成候选集 C_k 之后，利用频繁项集的先验性质，对候选集再剪枝处理，即后剪枝优化。通过两种剪枝策略相结合的方式，充分压缩最终参与求交运算的候选集规模。

1) 预剪枝优化

预剪枝优化的思想利用了以下性质：若 L_{k-1} 项集中存在项目 i，使得 i 在所有频繁项集 L_{k-1} 中出现的次数少于 $k-1$ 次，则与项集 L_{k-1} (i)连接后生成的 C_k 一定是非频繁的(L_{k-1}(i)指的是包含项目 i 的 L_{k-1} 项集)。

对上述性质采用反证法：假设现在有个 C_k 是频繁项集，根据频繁项集先验性质，它的所有子集也都是频繁项集，因此它的所有 $k-1$ 项子集都属于 L_{k-1}，任意项目 i 在该 C_k 的 $k-1$ 项子集中出现的次数都为 $k-1$ 次，则项目 i 在所有频繁集 C_k 的 $k-1$ 项子集中总共出现的次数至少为 $k-1$ 次，与性质中的条件相矛盾，故 C_k 是非频繁项集。

2) 后剪枝优化

为进一步压缩候选集的规模，对 C_k 进行后剪枝优化处理，其剪枝思想利用频繁项集先验性质[12]：频繁项集的所有子集都是频繁的，也就是说，若某个项集是非频繁的，则它的超集也一定是非频繁的。

上述性质在 Apriori 算法中采用的较多，本文将其引用到 Eclat 算法中，其证明过程不再赘述。后剪枝具体步骤如下：对每个 C_k 进行 $k-1$ 项子集的分解，判断所有 $k-1$ 项子集是否在 L_{k-1} 的集合中，若存在 $k-1$ 项子集不在 L_{k-1} 的集合中，则删除该 C_k。

下面以 $L_2=\{\{I_1I_2\},\{I_1I_4\},\{I_1I_5\},\{I_1I_6\},\{I_2I_6\},\{I_5I_6\}\}$ 为例，用上述两种剪枝策略相结合的方法阐述 C_3 产生的具体过程。在 L_2 中，只有 I_4 出现的次数小于两次，因此筛选掉 $\{I_1I_4\}$，$L_2'=\{\{I_1I_2\},\{I_1I_5\},\{I_1I_6\},\{I_2I_6\},\{I_5I_6\}\}$；对 L_2’自连接，根据等价类理论，只有$\{I_1I_2\}$、$\{I_1I_5\}$和$\{I_1I_6\}$相互间可以连接，连接后的 $C_3'=\{\{I_1I_2I_5\},\{I_1I_2I_6\},\{I_1I_5I_6\}\}$；对于$\{I_1I_2I_5\}$，由于子集$\{I_2I_5\}$不在 L_2 中，因此筛选掉$\{I_1I_2I_5\}$，则最终的 $C_3=\{\{I_1I_2I_6\},\{I_1I_5I_6\}\}$。

3.2.2. 支持度计算的改进

在 Eclat 算法中交集的计算是最耗时的环节，它需要一个接一个地比较两个项集的 tid，当项集的 TID_set 规模很大时这种方式依然很低效。尽管已有的算法做了相应改进，但是这些算法求交集的基本思想与原始 Eclat 算法是类似的。为此，本文提出一种基于事务索引的布尔数组取值计数法来计算交集以获得项集的支持度。其步骤如下：

首先，创建一个布尔型的数组，数组的长度由事务的数量确定，数组所有元素初始化为 0；

其次，将其中一个项集的 tids 作为索引，索引对应的数组元素设置为 1；

最后，将另一个项集的 tids 作为索引，取出索引对应的数组元素，如果取出的元素为 1，则把该元素的索引加入交集中，并增加一次支持度计数，如果取出的元素为 0，则跳过。

下面仍以 l_{21} 和 l_{22} 两个项目集为例，用本文给出的求交方法阐述具体过程如图 3 所示。在图 3 中，$\text{TID\_set}(l_{21})=\{1,3,5,6\}$，$\text{TID\_set}(l_{22})=\{3,4,5,6\}$。因为数据库的事务数为 6，则初始化布尔数组 $\text{Array}=(0,0,0,0,0,0)$。将 l_{21} 中的 tids 作为索引，把指向 Array 中的对应元素设为 1，即 Array [1] = 1，Array [3] = 1，Array [5] = 1，Array [6] = 1，则 $\text{Array}=(1,0,1,0,1,1)$。以 l_{22} 中的 tids 为索引，检索 Array 中相应的元素 Array [3]，Array [4]，Array [5]，Array [6]，其中值为 1 的分别是 Array [3]，Array [5]和 Array [6]，因此 $\text{TID\_set}(l_3)=\{3,5,6\}$，$\text{Support}(l_3)=3$。

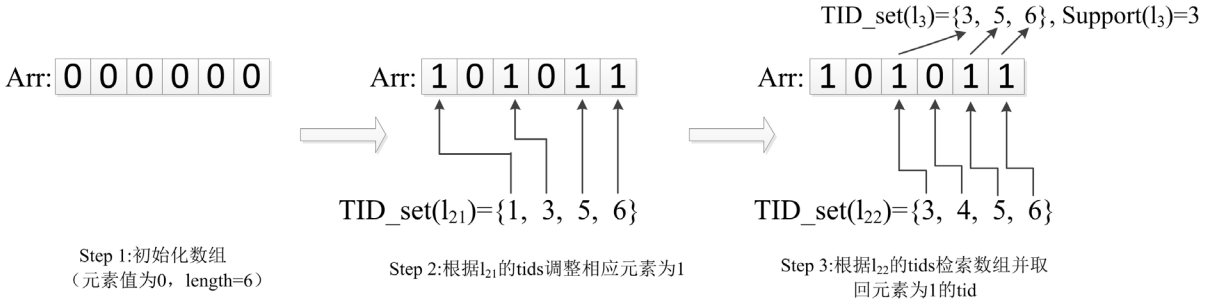

Figure 3. An example of improving the support degree calculation process of Eclat algorithm
图 3. 改进 Eclat 算法支持度计算过程示例

上述示例表明，通过使用新的求交方法，只需要设置和检索布尔数组的元素即可求得交集和支持度，避免了传统方法中逐项比较的过程。

3.2.3. 改进算法的具体实现

综合上述两方面的优化策略，改进后的 Eclat 算法实现步骤如下：

1、扫描事务数据库一次，为每个项目获取事务标识符(tid)，将数据库转成垂直格式；

2、在候选集 C_k’产生之前，先对 L_{k-1} 剪枝，筛选得到 L_{k-1}’：计算所有项目在 L_{k-1} 集合中出现的次数，删除项目频度小于 k－1 次的项目集；

3、利用等价类技术对 L_{k-1}’自连接，生成候选集 C_k’；

4、对 C_k’进一步剪枝，筛选得到 C_k：列出 C_k’项集的所有 k － 1 项子集，删除子集不在 L_{k-1} 集合内的项目集；

5、利用改进的求交运算方法得到 C_k 的 TID_set 和 Support，并根据预先设定的阈值确定最终的频繁项集 L_k；

6、循环 2 至 5 步过程，直到 L_k 的项集数量小于(或等于)k，终止算法。

改进 Eclat 算法的主程序伪代码如下：

输入：事务数据库 D，最小支持度阈值 min_sup

输出：频繁项集 L，项集支持度 L.Sup，项集对应的事务集 L.TID_set

1) D’=Scan(D); //扫描事务库 D 一次，得到垂直格式的数据库 D’

2) L_1=find_frequent_1_itemsets(D’);

3) $C_2=L_1 \infty L_1$; //通过自连接生成 C_2

4) L_2=New_quick_support_count(C_2, TID_set);

5) For(k=3;$L_{k-1}\neq\emptyset$;k++){

6) Prune(L_{k-1}); //对 L_{k-1} 剪枝

7) Lm$\in L_{k-1}$’, Ln$\in L_{k-1}$’;

8) If (Lm[1]=Ln[1]∧Lm[2]=Ln[2]∧…Lm[k-2]=Ln[k-2]∧Lm[k-1]＜Ln[k-1])

9) c=Lm∪Ln;

10) If (k-1)-subsets of c $\notin L_{k-1}$

11) Then delete c from C_k; //利用先验性质采取后剪枝策略

12) $C_k=c\cup C_k$;

13) L_k=New_quick_support_count(C_k, TID_set); }//调用改进的求交方法计算项集支持度，并筛选出 L_k

14) Return $L=\cup_k L_k$;

其中，预剪枝函数 Prune 伪代码如下：

输入：频繁项集 L_{k-1}

输出：筛选后的频繁集 L_{k-1}'

1) For all items I_n in L_{k-1}{

2) If count(I_n) in L_{k-1}＜k-1

3) Then delete $L_j(I_n \in L_j, L_j \in L_{k-1})$ from L_{k-1};}

4) Return L_{k-1}';

改进的支持度计算函数 New_quick_support_count 伪代码如下：

输入：候选集 C_k，事务集 TID_set

输出：频繁集 L_k

1) For all itemsets $C \in C_k${

2) Int Array[size]={0};//初始化数组，size 为事务 tids 长度

3) Set Array[Lm.TID_set]=1;//设置 Lm 索引对应元素为 1

4) If Array[Ln.TID_set]=1//检索 Ln 索引对应元素

5) Then get the tids and join in C.TID_set;//取出值为 1 的 tids

6) C.Sup=Length(C.TID_set);

7) If C.Sup＜min_sup

8) Delete C from C_k;}

9) $L_k=\{C \in C_k | C.Sup \geq min\_sup\}$;

10) Return L_k;

4. 实验设计及结果分析

4.1. 实验设计

为了验证改进 Eclat 算法的效率，本文通过实验分别将不同优化策略的改进算法与 Eclat 算法进行比较，并将 Apriori 作为参照算法进行分析说明。实验平台为 Intel Core i5，2.5 GHz，内存 8 G，操作系统 Windows 10 64 位。实验采用 UCI 标准数据集里的稠密型实际数据集 Mushroom 和 IBM Almaden 生成器生成的稀疏型合成数据集 T10I4D100K，这两个数据集经常被用于各类关联规则算法的测试。数据集的特征如表 2 所示。

实验一：将本文的改进算法与 Eclat 算法以及 Apriori 算法进行对比实验，测试对象为稠密数据集：Mushroom，测试各算法在不同支持度阈值下产生的候选集数量变化情况。测试结果如图 4 所示。

从图中结果来看，三种算法产生的候选集数量都随着支持度阈值的增加而逐渐减少，但是在同一支持度阈值下，Eclat 算法生成的候选集数量最多，Apriori 算法次之，本文的改进算法则最少。此外，三种算法在同一支持度下挖掘的频繁项集数量都是相同的。

实验二：将改进算法与 Eclat 算法以及 Apriori 算法进行对比实验，测试对象为稠密数据集：Mushroom，测试各算法在不同支持度阈值下执行时间的变化情况。测试结果如图 5 所示。

从图中结果看，三种算法的执行时间都随支持度阈值的减小而增大，但在同一支持度阈值下，本文算法执行效率要优于传统的 Eclat 算法和 Apriori 算法。其中，在支持度阈值较大时，三者在时间效率上区别不是很大，但随着支持度阈值的逐渐减小，三种算法执行时间的差距将越拉越大，改进算法的效率优势将体现的更加明显。此外，从 min_Sup = 0.25 开始，Apriori 算法在执行时间上有一个大幅的跃升。

实验三：将改进算法与 Eclat 算法、仅采用剪枝策略的 Eclat 算法以及仅改变求交运算的 Eclat 算法进

Table 2. Test data set related information
表 2. 测试数据集相关信息

Database	Number of transactions	Number of items	Average length	Type
Mushroom	8124	120	23	稠密
T10I4D100K	100000	1000	10	稀疏

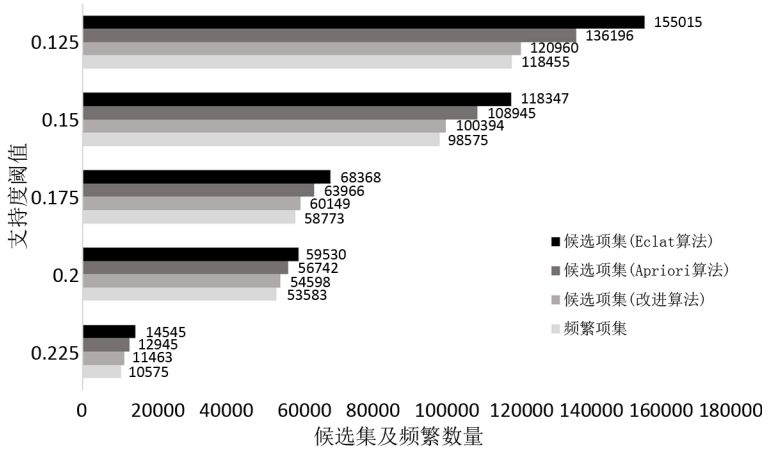

Figure 4. Number of candidate sets under different support thresholds on Mushroom dataset
图 4. Mushroom 数据集上不同支持度阈值下候选集数量

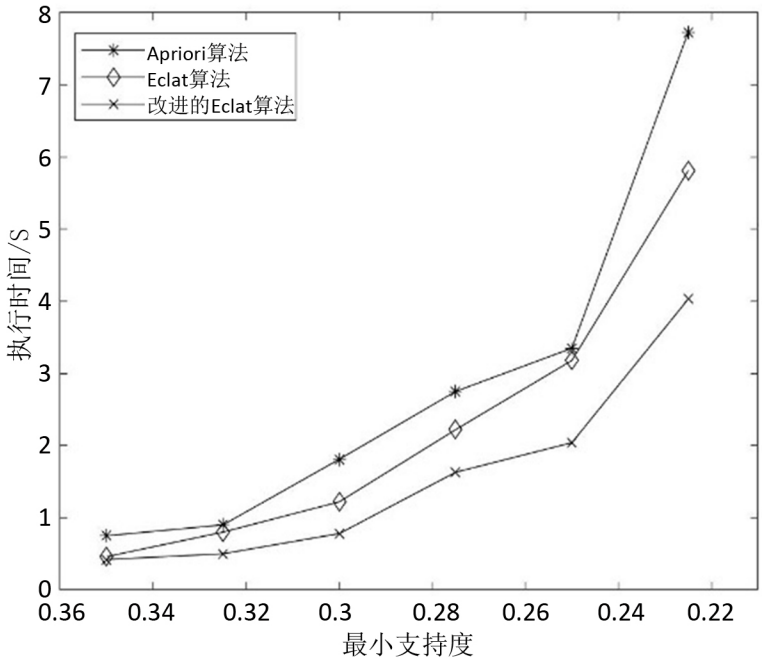

Figure 5. Number of candidate sets under different support thresholds on Mushroom dataset
图 5. Mushroom 数据集上不同支持度阈值下候选集数量

行对比实验，测试对象仍为稠密数据集：Mushroom，比较 Eclat 算法在不同优化策略下执行时间随支持度阈值变化情况。测试结果如图 6 所示。

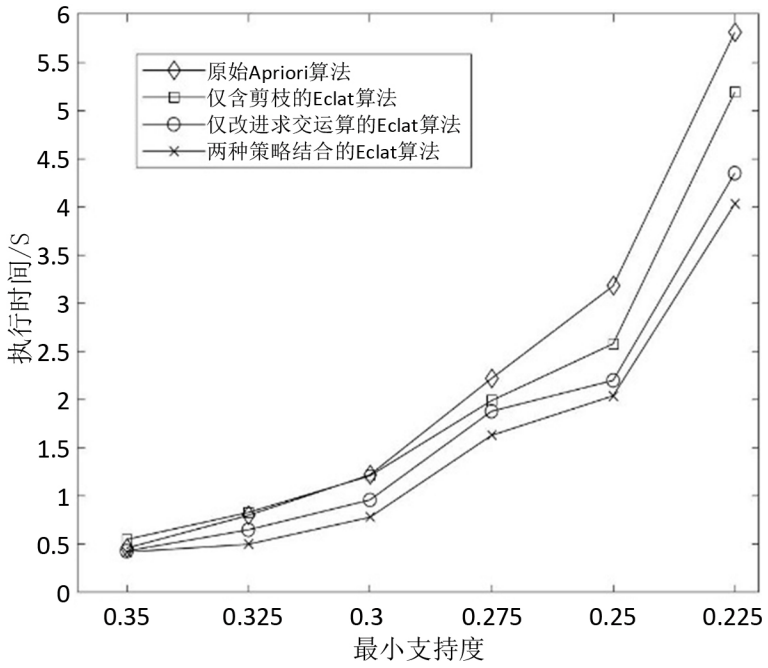

Figure 6. Execution time of Eclat algorithm under different optimization Strategies on Mushroom dataset
图 6. Mushroom 数据集上 Eclat 算法在不同优化策略下执行时间

从实验结果来看，不同优化策略的 Eclat 算法执行时间都随支持度阈值的减小而增加，在同一支持度阈值下，候选集优化和支持度计算改进相结合的算法时间效率要优于原始算法以及仅采用单一优化策略的算法。其中，在两种单一优化算法中，改进求交运算的优化方式要比剪枝优化方式算法运行效率更高，而没有采取任何优化策略的 Eclat 算法执行效率最低。

实验四：将改进的 Eclat 算法与原始 Eclat 算法进行对比实验，测试对象为稀疏数据集：T10I4D100K，测试在支持度阈值一定的情况下两种算法运行时间随事务规模变化情况。其中，事务规模从 20,000 到 100,000 变化，最小支持度为 0.01。测试结果如图 7 所示。

从实验结果来看，在给定的支持度阈值下，两种算法的运行时间都是随着事务集数量的增加而增大，但在同一事务集规模下，改进算法的运行时间要比原始算法更低，运行效率更高。

4.2. 结果分析

在实验一中，由于支持度的阈值不断提高，导致满足最小支持度的项集数量减少，即频繁 k-项集数量逐渐减少，从而连接产生的候选(k + 1)-项集数量也逐渐减少，因此可以看到算法的候选集数量是随着支持度阈值的增加而减少的。同时，由于 Apriori 算法本身利用了先验性质采取后剪枝策略，其候选集数量相比未采取任何候选集优化策略的 Eclat 算法要少，而本文的改进算法不仅利用了后剪枝策略，而且在连接之前还采取了预剪枝策略，前后剪枝策略的结合进一步压缩了候选集数量，因此在一定支持度下改进算法能够产生最少的候选集数量。

在实验二中，由实验一的分析结果可知，支持度阈值的减小会使候选集规模逐渐增大，大量的候选集数量需要更多的连接操作和求交运算，大大增加了时间开销，因此算法的执行时间会随着支持度的减小而增大。同时，虽然 Apriori 算法的候选集数量要比 Eclat 算法少，但执行效率反而要比 Eclat 低，这是由于算法的大部分时间开销都发生在获取支持度的阶段，前者支持度的计算采用的是重复扫描数据库事务的方式，这种方式占用的时间开销很大，尤其在事务规模庞大的情况下，而后者采用了逐项求交的方式，避免

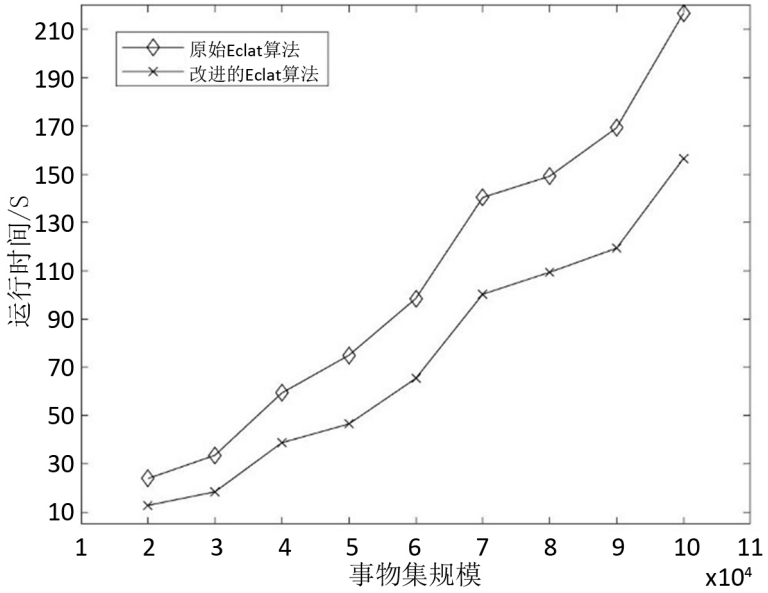

Figure 7. The running time of two algorithms on the T10I4D100K dataset under different transaction set sizes
图 7. T10I4D100K 数据集上两种算法在不同事务集规模下运行时间

了对数据库的重复扫描，所以 Eclat 算法效率要比 Apriori 算法高。本文的改进算法不仅对候选集的产生做了进一步优化，还对求交运算做了改进，时间开销进一步降低，从结果来看确实能够起到提高算法效率的目的。

对于实验三，不管是候选集优化还是改进求交运算，都能不同程度的提高 Eclat 算法的效率，候选集的前后剪枝在减少了连接判断操作的同时压缩了后续求交运算的项集规模，改进的求交运算则避免了逐项比较的问题，但对提高算法效率的贡献程度来看求交的优化方式比候选集的优化方式效果要好，这也验证了影响 Eclat 算法运行效率的主要因素在于求交运算过程。

对于实验四，由于事务集数量的增加，导致项集的 TID_set 规模越来越大，事务的 tids 越来越长，在求交运算时不管是原始的逐项比较还是本文的索引方法所耗费的时间都将会越来越久，因此随着事务数的增加算法运行时间也在增加。

5. 结束语

本文通过分析 Eclat 算法中普遍存在的候选集数量大、项集求交效率低的问题，提出了一种新的优化和改进思路，即利用前后剪枝相结合的策略来减少项集连接操作并压缩候选集规模，同时采用基于事务索引的布尔数组取值计数方式改进求交运算。实验表明，不管是稠密数据集还是稀疏数据集，该方法都能在一定程度上减小 Eclat 算法的时间开销，提高算法的挖掘效率，特别是在支持度阈值小、事务数规模大的情况下。此外，本文的改进算法由于减小了项集的搜索空间，内存占用更小，空间开销也得到了一定的降低。后续的研究工作将集中在算法的具体应用上，以验证本文算法在解决实际问题时的适用性和有效性。

基金项目

国家自然科学基金(61501513)。

参考文献

[1] 肖文, 胡娟, 周晓峰. 基于 MapReduce 计算模型的并行关联规则挖掘算法研究综述[J]. 计算机应用研究, 2018(1).

[2] Imielienskin, T., Swami, A. and Agrawal, R. (1993) Mining Association Rules between Set of Items in Large Databases. *ACM Sigmod Record*, **22**, 207-216. https://doi.org/10.1145/170036.170072

[3] Han, J., Pei, J. and Yin, Y. (2000) Mining Frequent Patterns without Candidate Generation. *ACM SIGMOD International Conference on Management of Data*, Dallas, Texas, 15-18 May 2000, 1-12. https://doi.org/10.1145/342009.335372

[4] Zaki, M.J. (2000) Scalable Algorithms for Association Mining. *IEEE Transactions on Knowledge and Data Engineering*, **12**, 372-390. https://doi.org/10.1109/69.846291

[5] 崔妍, 包志强. 关联规则挖掘综述[J]. 计算机应用研究, 2016, 32(2): 330-334.

[6] 周英, 卓金武, 卞月青. 大数据挖掘: 系统方法与实例分析[M]. 北京: 机械工业出版社, 2016.

[7] 韩家炜, Micheline, K., 裴健. 数据挖掘：概念与技术[M]. 第 3 版. 北京: 机械工业出版社, 2012.

[8] 饶正婵, 范年柏. 关联规则挖掘 Apriori 算法研究综述[J]. 计算机时代, 2012(9): 11-13.

[9] Zaki, M.J. and Gouda, K. (2003) Fast Vertical Mining Using Diffsets. *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, Washington, D.C., 24-27 August 2003, 326-335. https://doi.org/10.1145/956750.956788

[10] 熊忠阳, 陈培恩, 张玉芳. 基于散列布尔矩阵的关联规则 Eclat 改进算法[J]. 计算机应用研究, 2010, 27(4): 1323-1325.

[11] 冯培恩, 刘屿, 邱清盈, 等. 提高 Eclat 算法效率的策略[J]. 浙江大学学报(工学版), 2013, 47(2): 223-230.

[12] 曾雷. 关联规则挖掘中 Apriori 算法的研究[D]: [硕士学位论文]. 重庆: 重庆交通大学, 2016.