基于 FP-growth 的数据关联改进算法

贺恒松[1] 李文明[2] 李文锋[2]
(1.南京电子技术研究所 南京 210013; 2.南京轨道交通系统工程有限公司 南京 210013)

摘 要: 随着现如今数据收集能力和存储能力的大大增强,大规模数据挖掘分析的重要性越来越显得重要。然而,对大规模数据的分析挖掘并不是一件容易的事情。因此,为了可以更高效的分析这些数据,很多新的算法和数据结构逐渐被引入到了数据挖掘分析中去。针对关联分析,提出了一种名为高效频繁模式挖掘(advanced frequent pattern mining,AFPM)算法。基于前置频繁模式树(pre-frequent pattern tree,PFP-tree)来提升关联分析的性能,并提供了相应的算法来实现基于这种数据结构的关联分析。通过大量的实验数据验证了这种新型的数据结构在关联分析问题上是优于频繁模式增长(FP-growth)算法。

关键词: 数据关联; FP-growth; 频繁项集
中图分类号: TP311;TN915.07 **文献标识码**: A **国家标准学科分类代码**: 520.1040

Advanced data association algorithm based on FP-growth

He Hengsong[1] Li Wenming[2] Li Wenfeng[2]
(1. Nanjing Research Institute of Electronics Technology, Nanjing 210023,China;
2. Nanjing Rail Transit Systems Co., Ltd., Nanjing 210013,China)

Abstract: With the current data collection capacity and storage capacity greatly enhanced, the importance of large-scale data mining analysis more and more important. However, the analysis of large-scale data mining is not an easy thing. Therefore, in order to be able to more efficient analysis of these data, many new algorithms and data structures are gradually introduced to the data mining analysis. This paper is based on the correlation analysis, based on this article, proposed a called advanced frequent pattern mining (AFPM) algorithm. This algorithm is based on the pre-frequent pattern tree (PFP-tree) to improve the performance of association analysis and provide the corresponding algorithm to implement the association analysis based on this data structure. It is proved that this new data structure is superior to FP-growth algorithm in association analysis problem through a large number of experimental data.

Keywords: data association; FP-growth; frequent itemsets

0 引 言

关联分析[1]是数据挖掘的一个重要方面,主要用于在大型数据集中发现隐藏的有意义的联系。所发现的联系可以用关联模式或者频繁项集[2]的来表示。在许多领域,关联模式的分析有着十分重要的意义,如在个性化产品质量控制[3],商品推荐[4],告警关联[5],网络入侵[6]很多领域,都有着很好的使用。正是因为关联分析能够帮助人们发现数据集中许多有价值的知识,所以自从关联分析的诞生到现在都是数据挖掘领域研究的热点。现在的关联分析方法主要有 Apriori,频繁模式增长(frequent pattern grouth,FP-growth)等多种算法模型,都侧重与不同的数据结构类型展开对数据的关联分析。

但是 Apriori 有很大的局限性,比如它对数据集的建立多次遍历产生不可低估的 I/O 开销,对于稠密数据集的处理性能较低等。尽管文献[7]提出了一些对 Apriori 算法的优化主要是在对对数据集在挖掘的过程中进行剪切,以及减少对数据集访问的 I/O 开销,但是上文提及的问题仍然存在。文献[8]提出了基于改进的数据集存储结构来进行处理,但是对于稠密数据集来说,处理效率较低。所以本文没有在 Apriori 的基础上进行算法改进,而是主要在 FP-growth 的基础上提出一种新的数据结构类型。而且现如今各个领域的人从多方面对 FP-growth 算法进行改进。文献[9]提出了一种新的增量式关联规则挖掘算法,提出了采用图片的方式来存储各种项集的支持度,但是这种方式不适合对于项数过大的数据集挖掘。文献[10]分别从不同的

收稿日期:2017-02

角度提出通过减少一次遍历排序直接构建 PFP-tree 来缩短时间,但是通过这种方法构建的 FP-tree 基本上规模都会很大,复用节点很低。文献[11]提出了使用支持度计数二维表的方法,来减少对条件模式基的第一次遍历。文献[12]提出了分割数据集,来进行 FP-growth 算法的挖掘,但是该算法引入了额外的数据遍历,且挖掘算法本身没有进行改进等。

针对上述所讨论的不足,本文提出了一种基于 FP-growth 的 AFPM 算法。首先,此算法采用了一种新的数据结构前置频繁模式树(pre-frequent pattern tree,PFP-tree)来存储预处理后的数据集;其次,是在 PFP-tree 上进行规模较小速度较快的低维度频繁项集挖掘;最后,在已经挖掘好的低维度频繁项集上进行更进一步的高维度的频繁项集挖掘。为了证明本算法的优越性,本文对多个不同状态的数据集进行了频繁项集挖掘实验较原算法和部分改进算法在一定情况下有一定地提升。

1 相关理论

项集和支持度计数 令 $I=\{i_1,i_2,\cdots,i_d\}$ 代表数据表中的所有项的合,而 $T=\{t_1,t_2,\cdots,t_N\}$ 是所有事务的集合。在关联分析中,包含 0 个或多个的集合称为项集,如果一个项集包含 K 个项则称为 K-项集。这里的事务 t_i 包含若干个项集。

关联规则(association rule)关联规则是形如 $X\rightarrow Y$ 的蕴含表达式,其中 X 与 Y 是不相交的项集,即 $X\cap Y=\varnothing$。

支持度和置信度是关联规则的强度度量。支持度可以用来表示事务中的项集的频繁程度,置信度是用来确定 Y 在 X 中的频繁程度。支持度 s 和置信度 c 的具体定义如下所示:

$$s(X\rightarrow Y)=\frac{\sigma(X\bigcup Y)}{N} \tag{1}$$

$$c(X\rightarrow Y)=\frac{\sigma(X\bigcup Y)}{\sigma(X)} \tag{2}$$

2 FP-growth 算法及优化分析

FP-growth 算法是在 FP-tree 自底向上进行探索挖掘的算法,详细叙述参考文献[7]。此算法主要分为两个方面,一是 FP-tree 的生成,另一部分是从 FP-tree 发现频繁项集。

1)FP-tree 的构成。首先,对获取到的大批量数据进行读入,遍历一次数据集,获得每个项的支持度计数,对其进行递减排列,删除非频繁项。其次,逐一将事务集里的各个项按照前面所获的项的支持度计数递减排列,再将其添加到 FP-tree 中去,同时更新所添加事务所在路径上的频度计数。

2)频繁项集的发现。这种频繁项集的发现采用了分治策略。首先,按照步骤 1)的获得的项的排列顺序,逐一查找以本项结尾的前缀路径。其次,根据获得的前缀路径,来产生条件 FP-tree,更新子树上的频度计数,把符合要求的项添加进频繁项集,然后在此基础上进行迭代生成高度更低的条件 FP-tree。最后,当访问到 Root 节点时结束算法。

通过 FP-growth 算法的分析,发现在其步骤 2)进行挖掘时,会产生大量的条件 FP-tree,然而每一层的条件 FP-tree 的数目都是上一层的条件 FP-tree 数目与其所包含的频繁项的乘积。因此,如果可以从所以本文,就从这个角度入手,通过改进的数据关联算法减少前期无用条件 FP-tree 的遍历,来降低时空复杂度。

3 AFPM 算法

3.1 PFP-tree

PFP-tree 是基于文献[9]的支持度序列树(support-ordered tree),对其进行了改进。PFP-tree 主要有 3 层树节点,每个树节点带有两个属性,分别是标签 l 和频度计数 w。标签 l 用来表示每个节点所代表的项 a_i($a_i\in I$)的内容,频度计数 w 用来表示每个节点所代表的项 a_i($a_i\in I$)在不同事务集中出现的总次数。在 PFP-tree 中,不同的节点层代表不同的项集:第 1 层代表是 1-项集,2 层代表的 2-项集,3 层代表 3-项集。通过 PFP-tree,可以快速的查找到数据集中的频繁 1-项集,频繁 2-项集和频繁 3-项集。

具体的特点如下所示。

1)数据的起始点是 *Root* 点,其 l 为 Null;

2)子节点按照支持度递减的顺序排列,即 $S(N_i)\geqslant S(N_j)$ && $i<j$;

3)任意节点 N_i(不包括第 3 层)的子节点 $C(N_i)$ 都存在:$N_j\in C(N_i)$,$N_i>N_j$。

其中,*Root* 表示 PFP-tree 的根节点,N_i、N_j 表示树上的节点,且 N_i 在 N_j 的右侧,$S(N_i)$ 来表示 N_i 节点的支持度,$C(N_i)$ 来表示 N_i 节点的子节点。如果节点 N_i 是第 1 层的节点,则此节点的 $S(N_i)$ 表示此节点中的项集 $\{a_i\}$ 所对应的支持度。如果是第 2 层的节点,则 $S(N_i)$ 表示的是项集 $\{a_h,a_i\}$ 项集的支持度,a_h 是 a_i 的父节点。如果是第 3 层的节点,则 $S(N_i)$ 表示的是项集 $\{a_g,a_h,a_i\}$ 项集的支持度,a_i 是 a_h 的父节点。

为了方便,采用 PT 来代表 PFP-tree。

图 1 显示了一个数据集,它包含 3 个事务 4 个项。图 1 中还包含了给 3 个事务读入后生成 PT 的流程。树上个每个结点都包含了一个项标签和支持度。初始的 PT 只包含一个 *Root* 结点。随后,采用如下的方法来扩充 PT。

1)从数据集中的条事务 1,从中提取 $\{A\}$,$\{B\}$,$\{A,B\}$ 这些项集,并把这个集合称作 P。

2)根据 P 中的项集 $\{A\}$,创建标记为 A 的节点。然后形成 $Root\rightarrow A$ 的路径,对该项集进行编码。对本项集的最后一项所对应的支持度进行加一更新。接下来,再逐一读入 P 中的项集 $\{B\}$ 和 $\{A,B\}$,进行上述操作。在 1 事务经

过后形成 PT，如图 1(a)所示。

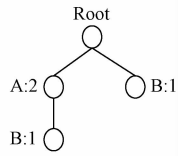
3)读入数据集中的事务 2，从中获取项集 {*B*}、{*C*}、{*D*}、{*B*,*C*}、{*C*,*D*}、{*B*,*C*,*D*}，同样把这个集合记为 *P*。

4)根据 P 中的项集 {*B*}，由于这个项集在步骤 2)后生成的 PT 中已经存在这个节点，所以结点 B 的支持度更新为 2，然后与其前项进行比较，发现节点 *A* 的支持度小于节点 *B*，这里将两者的顺序进行对调。接下来，在逐一读入 *P* 中的项集，对于在 PT 中已经有部分重叠的项集，则需在其对应重叠路径最后一个节点后添加子节点，并将对应的支持度更新到该子节点，在 2 事务经过后形成的 PT，如图 1(b)所示。

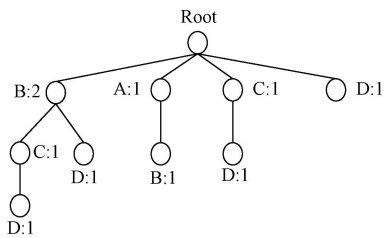
5)读入数据集中的事务 3，按照从步骤 3)～4)的顺序进行处理。

事务数据集

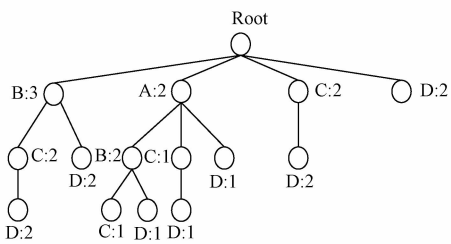
TID	项
1	{A,B}
2	{B,C,D}
3	{A,B,C,D}



(a) 读入TID=1之后



(b) 读入TID=2之后



(c) 读入TID=3之后

图 1　PFP-tree 生成示意图

在上述过程中，在步骤 4)中加入了比较的过程，主要是为了使得 PT 的有序性。这里的有序性是指属于同一个父节点的所有子节点按照支持度的大小顺序从左到右是递减的。正是因为有序性的存在，在查找 1-项集，2-项集和 3-项集时，效率会大幅度提高。举个例子来说，如果关联分析问题定义的支持度门限是 3，这里当查找到 *Root* 节点的子节点 *B* 时，发现其的支持度 2，则后面所有的节点都可以直接跳过，不在进行查找。

PT 还有一个优点就是，它支持数据集的动态扩展，每当有新的事务加入到数据集中后，即可立即加入到 PT 中去。这是因为此数据结构在构造的过程中是与支持度门限值无关的，只是对数据集中事务的重组。

具体的 PT 构造在数据预处理阶段的伪代码如下。

算法 1 PT 构造算法

输入：数据集中的事务 *C*

输出：完整 PT，3 层索引地址 *I*

```
Start
1   构造 PT，用 Y 代替
2   for (k = 1; k ≤ 3;k++) do
3   for each T in C do// T 是事务集中的一条事务
4     从 C 中获取所有的 K -项集，存储进 C_k
5     for each X in C_k do
6     遍历 Y，沿着 X 代表的路径来定位节点 Z
7     if( Z is exist)
8       更据要求增加或者减少 Z 节点的支持度
9       if( Z.Count == 0)
10        删除 Z 节点
11        endif
12        else
13          产生一个新的节点 P，并标签其支持度(Count)为 1
14          把该节点按照自左到右降序进行插入
15      endif
16      endfor
17  endfor
18  endfor
```

由于这里 PFP-tree 只是一个存储结构，所以这里对于 PFP 的产生是在数据预处理阶段进行的。考虑到 PT 是一个 3 层树结构，二层的节点 N_2 和 3 层的节点 N_3 的数量会大大的超过一层的节点 N_1，因此就两层节点存储在 XML 文件中，方便读取。虽然这样做加大了 I/O 操作，但是根据文献[5]，增加的 I/O 操作对于它对后续挖掘算法性能的提升是不重要的。

3.2　关联挖掘算法

本论文提出中所提到算法是在 3.1 节的 PT 构建好的基础上，从数据集中挖掘频繁的关联项集。本关联算法可以分为两个部分，一个部分是在 PT 上快速查询获取满足支持度门限的 1-项集，2-项集和 3-项集，叫做前置前置频繁模式树挖掘(PFP-tree mining，PTM)。另一个部分就是在获取到 1-项集，2-项集和 3-项集后，运用这些已经获取的频繁项集的优势，结合 FP-growth 算法的优点，挖掘那些规模大于 3 的项集，叫做前置频繁模式树后挖掘(PFP-tree next mining，PTNM)。通过这种方式，结合了 PTM 算法查找 1，2，3-频繁项集快速查找和 PTNM 算法能够对高维度频繁项集的快速挖掘的优点，提高关联分析的效率。

PTM 算法算法思路：从 PT 的 *Root* 节点开始，向下遍历去查询满足支持度门限的节点。在查询过程中，充分利

用 PT 有序性的特点,按照深度优先的原则,自左至右进行查询,一旦遇见支持度小于门限值的节点,则可以直接结束该节点所在的某个子节点集合的查旭。为了更加形象直观地展示算法,这里引用图 1 中的数据和 PT,在图 2 中展示具体流程,设定的支持度门限值为 2。

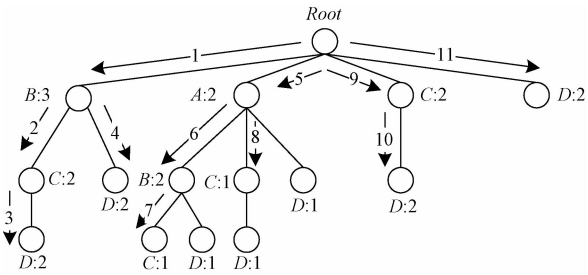


图 2 支持度门限为 2 时,PT 的遍历路径

通过图 2,可以看出,当支持度门限为 2 时,在对数进行第 7 次和第 8 次遍历时,遇到支持度为 1 的节点,不满足门限要求,所以直接跳过对应子节点集合中的后续节点。通过这 11 次的遍历获取了本数据集中所有的 1,2,3-频繁项集,相对于 FP-growth 算法,速度大幅度的提升。

为了进一步突出算法的优越性,这里讨论了当支持度门限值发生变化后,算法执行的流程。图 3 展示了当支持度门限改变为 3 时,对 PT 的遍历路径。PTM 算法在第 3 次对 PT 进行查询时就停止了。因为当访问完集合 B:3 后,在去访问 B:3 的第一个子节点和第一个兄弟节点,发现支持度都不满足门限要求,所以可以直接判断后续的项集都不会满足门限要求。所在 3 次遍历后,便得到了所要的频繁项集 $\{B\}$。

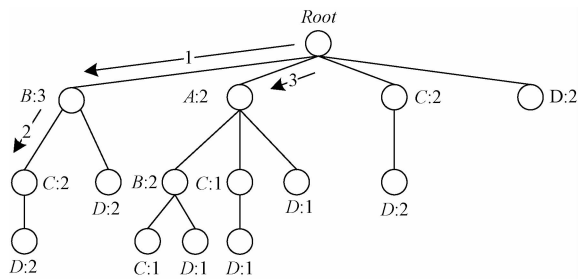


图 3 支持度门限为 3 时,PT 的遍历路径

为了方便接下来算法对生成的频繁项集的访问,降低复杂度,本算法中提出一种树型数据结构来存储部分频繁 3-项集。如图 4 所示,这里假设所采集的频繁项集按照支持度递减的排列是 $\{A,B,C,D,E\}$,获得的 3-频繁项集为 $\{B,A,D\}$、$\{C,D,E\}$、$\{B,D,E\}$。注意这里的 3-项集是不包含支持度最高的项的,具体原因在性能分析中介绍。然后,将这些 3-项集按照支持度递减的顺序,存储到频繁 3-项项集树 T_3 中。

通过上述两个实例的说明,充分说明了此算法的思想,

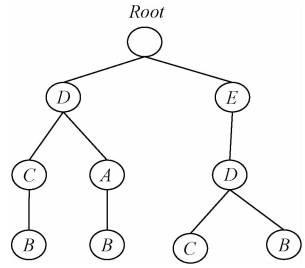


图 4 频繁项集存储结构

下面给出此算法的实现伪代码。

算法 2:PTM 算法

输入:支持度门限 $S\%$

输出:规模为 1,2 和 3 的频繁项集

相关参量说明:

I_f 表示 L_1 中支持度最高的一项

N_x^y 表示 x 节点的第 y 个子节点

NC_x 表示 x 节点的子节点数目

$Count$ 表示节点上的支持度

$|D|$ 表示数据集 D 中所有事务的个数

I_n 为 n 节点所代表的项集

```
1  for (i = 0;i < NC_Root ;i++) do
2  X = N_i^Root
3  if( X.Count > |D| * S% )
4    把 I_X 加入到频繁 1-项集 L_1 中
5    for (j = 0;j < NC_x ;j++) do
6    Y = N_j^X
7    if( Y.Count > |D| * S% )
8      把 I_Y 加入到频繁 2-项集 L_2 中
9      for (z = 0; z < NC_Y;z++) do
10       If( N_Z^Y.Count > |D| * S% )
11         把 I_{N_Z^Y} 加入到频繁 3-项集 L_3 中
12         If( I_{N_Z^Y} 中不包含 I_f )
13           存储进频繁 3-项集树 T_3 中
14         end if
15       end if
16       end for
17   end if
18   end for
19 end if
20 end for
21 运行 PTNM 算法
```

PTNM 算法思路:PT 中第一层节点是数据集 D 所有项按照递减顺序所得到的排序,在 PTNM 中直接调用过来,再对数据集中的事务进行读取,按照文献[8]中的 FP-Growth 算法构建 FP 树。在构建完 FP 树后,利用在 PTM 中已经获得频繁 1,2,3-项集,利用先验原理,在频繁 3-项集

的基础上直接按 3 项集中项的顺序来调用迭代生成条件 FP-tree，获取各个规模 $K>3$ 的频繁项集。最终，完成获取所有频繁项集的目的。

PTNM 算法伪代码如下所示。

算法 3：PTNM 算法

输入：支持度门限 S，数据集 D，频繁 3-项集 LT_3

输出：频繁项集 $L(K>3)$

相关参数：

L_Y^X 表示 LT_3 上 X 节点的第个 Y 子节点

LC_X 表示 LT_3 上 X 节点的子节点数目

T_X 表示项 X 对应的条件 FP 树

Start

1　构建 FP 树 T

2　for ($i=0$; $i<LC_{\text{Root}}$; i++) do

3　$X=L_i^{\text{ROOT}}$

4　以 X 为基础，构造 X 条件 Fp 树 T_X

5　for ($j=0$; $z<LC_X$; z++) do

6　　$Y=L_j^X$

7　　在 T_X 基础上，构造 Y 的条件模式基，在此基础上构造条件 FP 树 T_Y

8　　for ($z=0$; $z<LC_Y$; z++) do

9　　R= L_Z^Y

10　　构建对应的频繁 3-项集 P

11　　在 T_Y 基础上，构造 Y 的条件模式基，在此基础上构造条件 FP 树 T_Z

12　　L=FP-growth(T_Z,P)

13　end for

14　end for

15　end for

算法 4：FP-growth(Y,α)

输入：已经构造好条件 FP 树 Y，项集 α，最小支持度 S；

输出：事务数据集 Y 中的频繁项集 L ；

L 初值为空

1　if Y 只包含单个路径 P then

2　for each β in 路径 P 中节点的每个组合 do

3　　产生项目集 $\alpha\cup\beta$，其支持度 support 等于 β 中节点的最小支持度数；

4　　return $L=L\cup$ 支持度数大于 S 的项目集 $\alpha\cup\beta$

5　end for

6　else　　//包括了多个支路

7　for each Y 的头表 L_1 中的每个频繁项 f do

8　　产生一个项目集 $\beta=f\cup\alpha$，其支持度等于 f 的支持度

9　　构造 β 的条件模式基 B，并根据该条件模式基 B 构造 β 的条件 FP 树 T_β

10　　if $T_\beta\neq\varnothing$ then

11　　递归调用 FP-growth(T_β,β)；

12　　end if

13　end for

14　end if

3.3　性能分析

本文所讨论的算法的性能分析，主要是从 PT 着手，着重分析算法的空间复杂度和时间复杂度。

PTM 算法主要在是挖掘维度比较低的频繁项集。算法主要是在 PT 上挖掘频繁项集，从上述伪代码，可以看出其的时间复杂度接是 O(N)。因为只遍历了一次 PT 便可以获得所有的低维频繁项集，故其挖掘的效率特别高。

同时，PTM 算法的空间复杂度主要是涉及到 PT 第 1 层的存储，后两层的数据主要是存储在硬盘上的，所以这部分算法的空间复杂度，主要取决于数据集中的非重复项的多少。而数据集的项数一般都不会很大，因此，算法的空间复杂度也不会很大。

对于 PTNM 算法，该算法的空间复杂度主要是在挖掘高维度频繁项集过程中生成条件 FP-tree，但是相对于 FP-Growth 来说，它是在频繁 3-项集的基础上有序进行挖掘的。

假设这里完整的 FP-tree 节点为 N 个，频繁项为 M，则可以假设 FP-growth 在第 1 层遍历是每个访问对应条件模式基的个数大致为 $N/2$，这样第 1 层遍历所要花费的时间和空间复杂度都是一般大致接近于 $MN/2$。这个数值主要是有数据集的特性来确定，对于不同的数据集可能会有很大的偏差。

而 FP-Growth 算法对第 2 层遍历基本上是在第 1 层生成的 $D(D\leqslant M(M-1))$棵条件 FP-tree 基础上进行的，所要遍历的点基本上对应的条件树节点的平均数与所含频繁项数的乘积，这里假设所有条件树节点的平均值是 $N_1(N_1<N)$，平均所含的频繁项为 Z 个，则可以得出时空复杂基本为 $DN_1/2$。

同样，FP-growth 进行第 3 遍遍历是在上一层所生成的 $E(E\leqslant(M-2)Z)$棵条件树的基础上进行挖掘，所以这里假设条件树节点的平均值为 N_2，平均所含的频繁项为 Y 个，则这样可以得出其的时空复杂度基本上是 $EN_2/2$。注意 E 的上限是 MZ 。所以可以看出前 3 层的时空复杂度基本基本为($MN_1/2+MN/2+EN_2/2$)。

本文的 PTNM 算法在获取高维度频繁项集之前，在 PTM 算法获取频繁 3-项集的过程中加入了分析来减少遍历节点。由于 FP-tree 都是由支持度递减排列项的事务集所构成，所以其挖掘的终结条件便是挖掘到支持度最高的子节点或者是 *Root* 节点。在所以 PTM 算法，加入了对于支持度最高项的检测，如果含有支持度最高的项，将其加入到 PTNM 将要遍历的频繁 3-项集树 LT_3，便可以直接跳过对这些项集的遍历。

除了在 PTNM 算法的遍历数据集遍历的过程中加入了分析剪切外，由于 PTNM 算法是直接在频繁 3-项集的基

础上进行挖掘的,所以这里耗费的时空相对来说会减少很多。假设通过 PTM 生成的频繁 3-项集树 LT_3 3 层的节点说分别是 M_1、M_2、M_3,生成的 FP-tree 节点树是 N。这样,在 PTNM 算法第一次遍历时,需要是在第 1 层 M_1 个节点上进行条件树的生成。考虑到不会所有的频繁 1-项集的都会是频繁 3-项集的子项,故可得 $M_1 \leqslant M$。在这种情况下,所耗费的时空复杂度 $M_1N/2$。

当 PTNM 算法在遍历到第 2 层时,假设这时的条件树平均节点为 N_1,这里所要遍历的条件 Fp 树是 M_2 棵,与 LT_3 树的第 2 层节点树相同。相对于 FP-growth 算法,这里的 M_2 由于都是频繁 3-项集的前缀,不包含那些满足频繁 2-项集的条件 FP-tree,并且那些子树条件模式基的遍历与构建都省略了,故可得 $M_2 \leqslant D$。其的时空复杂度为 $M_2N_1/2$。

当 PTNM 进行第 3 次遍历时,由于是获得频繁 3-项集的条件 FP-tree,以与 FP-growth 算法相比,所耗费的时空复杂度基本一致。

总的看来,AFPM 算法在 PTM 算法中的 O(N)的遍历挖掘低维度频繁项集和高维频繁项集挖掘的总的时空复杂度是优于 FP-growth 算法的。只是在所有频繁 2-项集都是频繁 3-项集子集,且频繁 3-项集都不包含支持度最高的项的情况下,本文的算法性能会与 FP-growth 差不多。

4 实验结果与分析

实验采用的环是一台 CPU 为 Intel Core i5-2430M,内存为 4G,操作系统为 64 位 Win7 系统的 PC。实验使用 Java 来实现,JDK 的版本是 1.8.0。这里所采用的是通过 IBM Quest Data Generator 产生的 T25.I20.N10K.D100K (11MB),T2.I2.N32 K.D640 (10MB)和 T25.I20.N32K.D640K(73MB)3 个数据集来进行测试,使用 D_1,D_2,D_3 来表示。这里使用 Tw.Ix.Ny.Dz,来表示数据集,w 表示数据集中事务集的平均规模,x 表示潜在频繁项集的平均规模,y 表示事务集不同项规模,z 表示数据集事务规模。

由于 PT 的构造是在数据预处理阶段就完成的,而且一次生成多次使用,所以其的构造时间不计入到本次实验所统计的时间内。

图 5 显示了在不同数据集特的条件对本文算法和传统的 FP-growth 算法在支持度变化条件的获取满足支持度门限要求的频繁项集的性能比较。

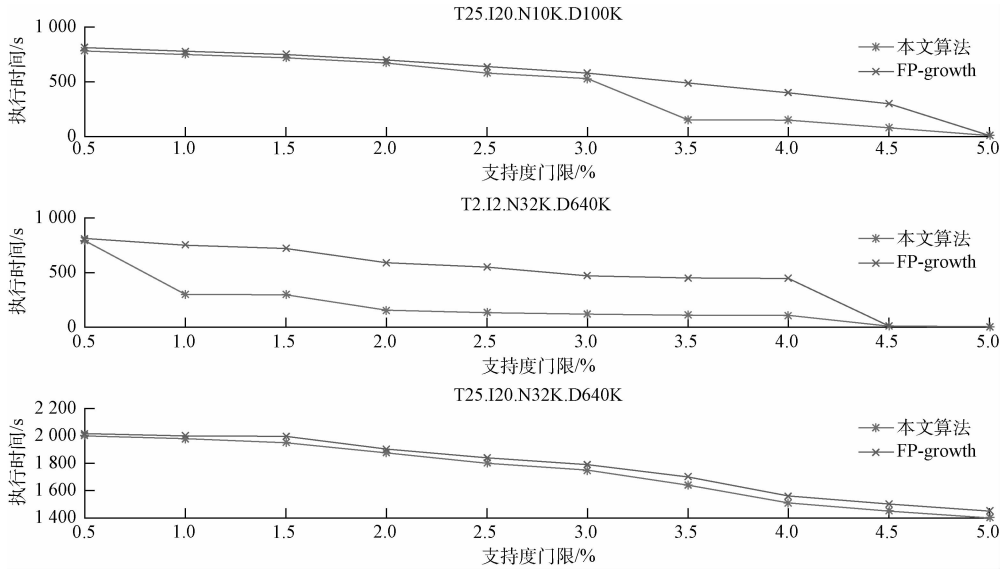


图 5 本文算法和 FP-growth 算法在 3 种不同数据集中不同支持度情况下的执行时间

由图 5 可以看出,改进后的算法确实是优于传统算法的。通过对图中的 D_1,D_3 数据集运行图的分析,可以看出,当数据集本身的潜在频繁模式规模较大且支持度较低时,频繁模式的规模大且数目多,传统 FP-growth 算法和 APM 算法所花费的时间都很多,但是 APM 算法在已经获取的 3-项集基础上对数据集中的频繁模式进行快速挖掘,相对于 FP-growth 算法效率有一定的提升。而且通过对图中 D_1,D_2 数据集运行时间的分析,可以发现当数据高过某个节点后,APM 算法的性能大幅高于 FP-growth 算法,随着支持度的增加,频繁模式的规模在变小,这样当规模降到 3-项集规模时,APM 算法就可以 O(N)的效率来遍历查找频繁项集,挖掘效率得到了大幅的提升。

5 结 论

本文所提出的 PT 数据结构,在一次构建后可支持多次查询,大幅度降低了频繁项集挖掘中数量较大的 1-项集,2-项集和 3-项集的挖掘难度。同时,结合了 PT 数据结构,对传统的 FP-growth 算法进行了改进,降低在挖掘过

程中条件 FP-tree 的生成。通过实验验证了本文算法的有效性，在执行的时间复杂度和空间复杂度上是优于传统关联挖掘算法的。当支持度发生变化时，算法也可以挖掘出比较可靠的频繁模式。在后续的工作，将会着手将上述算法应用到通信告警关联性分析中去。

参考文献

[1] TAN P N，STEINBACH M，KUMAR V. 范明，范宏建，译. 数据挖掘导论[M]. 北京：人民邮电出版社，2011.

[2] 郑劼诚. 基于改进 FP-树的关联规则增量式更新算法的研究与应用[D]. 南昌：南昌大学，2014.

[3] 李存荣，张开敏，杨明忠. 关联知识规则在产品质量控制中的应用[J]. 仪器仪表学报，2004，25(S1)：966-968.

[4] 张志宏，寇纪淞，陈富赞，等. 基于关联分析的多目标商品组合选择方法[J]. 系统工程学报，2011，26(1)：132-138.

[5] 穆倩. 关联分析技术在通信网告警处理系统中的研究与应用[D]. 保定：华北电力大学，2013.

[6] 王中生，刘猛. 基于重定向技术的 Honeynet 入侵模式算法研究[J]. 电子测量技术，2008，31(12)：27-30.

[7] 吴陈，李丹丹. 基于粗糙集的关联规则挖掘方法的研究与应用[J]. 电子测量技术，2016，39(7)：44-48.

[8] 崔贯勋，李梁，王柯柯，等. 关联规则挖掘中 Apriori 算法的研究与改进[J]. 计算机应用，2010，30(11)：2952-2955.

[9] 王晗，孔令富. 一种新的增量式关联规则数据挖掘方法研究[J]. 仪器仪表学报，2009，30(2)：438-443.

[10] 杨云，罗艳霞. FP-Growth 算法的改进[J]. 计算机工程与设计，2010，31(7)：1506-1509.

[11] 操漫成. 基于 MFP-tree 的关联规则挖掘算法研究[D]. 合肥：合肥工业大学，2008.

[12] 刘喜苹. 基于 FP-growth 算法的关联规则挖掘算法研究和应用[D]. 长沙：湖南大学，2006.

作者简介

贺恒松，硕士研究生，主要研究方向为数据挖掘与分析。

E-mail：hehengs@163.com

李文明，硕士，研究员级高级工程师，主要研究方向为数据分析、轨道交通信息化。