数据流中一种快速启发式频繁模式挖掘方法*

张 昕, 李晓光, 王大玲+, 于 戈

(东北大学 信息科学与工程学院,辽宁 沈阳 110004)

A High-Speed Heuristic Algorithm for Mining Frequent Patterns in Data Stream

ZHANG Xin, LI Xiao-Guang, WANG Da-Ling+, YU Ge

(School of Information Science and Engineering, Northeastern University, Shenyang 110004, China)

+ Corresponding author: Phn: +86-24-83687776, E-mail: dlwang@mail.neu.edu.cn, http://www.neu.edu.cn

Received 2004-11-29; Accepted 2005-03-11

Zhang X, Li XG, Wang DL, Yu G. A high-speed heuristic algorithm for mining frequent patterns in data stream. *Journal of Software*, 2005,16(12):2099−2105. DOI: 10.1360/jos162099

Abstract: Of the current approaches to frequent pattern discovery in stream data, the batch approach requires enough data, while the heuristic approach can deal with stream data directly. Although the average speed of the batch approach is higher, it cannot response on time and the query granularity is rough. This paper proposes an improved Lexicographic tree, IL-TREE (improved lexicographic tree), and gives a novel heuristic algorithm, called FPIL-Stream (frequent pattern mining based on improved lexicographic tree), which locates the historical patterns rapidly in the stage of updating the patterns and generating the new ones. Moreover, a policy for the titled window is integrated into the algorithm for recording the historical information in details. With the promise of the processing stream data on time, the algorithm reduce the average processing time greatly and provides a finer granularity of query.

Key words: data mining; data stream; frequent pattern; tilted window

摘 要: 在现有的数据流频繁模式挖掘算法中,批处理方法平均处理时间短,但需要积攒足够的数据,使得其实时性差且查询粒度粗;而启发式方法可以直接处理数据流,但处理速度慢.提出一种改进的字典树结构——IL-TREE(improved lexicographic tree),并在其基础上提出一种新的启发式算法 FPIL-Stream(frequent pattern mining based on improved lexicographic tree),在更新模式和生成新模式的过程中,可以快速定位历史模式.算法结合了倾斜窗口策略,可以详细记录历史信息.该算法在及时处理数据流的前提下,也降低了数据的平均处理时间,并且提供了更细的查询粒度.

关键词: 数据挖掘;数据流;频繁模式;倾斜窗口

中图法分类号: TP311 文献标识码: A

* Supported by the National Natural Science Foundation of China under Grant Nos.60473073, 60573090, 60503036 (国家自然科学基金)

作者简介: 张昕(1979−),男,内蒙古赤峰人,博士生,主要研究领域为数据挖掘与复杂网络分析;李晓光(1973−),男,博士生,主要研究领域为数据挖掘,Web 挖掘与信息检索;王大玲(1962−),女,博士,教授,CCF 高级会员,主要研究领域为数据挖掘与 Web 挖掘;于戈(1962−),男,博士,教授,博士生导师,CCF 高级会员,主要研究领域为数据库理论与技术.

随着流数据应用不断增多,数据流上频繁模式挖掘得到越来越多的关注[1−4].区别于传统的静态数据,数据流具有连续性、无序性、无界性及实时性的特点,通常要求算法能够实时地反映模式变化.在已有的数据流中,任意长度频繁模式挖掘算法可分为两类:批处理方法和启发式方法.批处理方法[1,2]处理速度较快,但需要积攒足够数据,无法满足实时性要求,并且查询粒度通常较粗.启发式方法[3,4]能够随流数据的到达直接进行处理,在一定程度上可以实时地反映频繁模式的变化,但对于高速到达的数据流,其处理速度仍然有限,且现有算法的模式统计计数不包含详细历史信息,使得模式估计与查询精度仍然较低.

针对启发式方法的挖掘效率和查询精度问题,本文提出一种改进的字典树结构 IL-TREE(improved lexicographic tree)用来高效存储频繁模式,并基于该结构给出了一种快速启发式算法 FPIL-STREAM(frequent pattern mining based on improved lexicographic tree),用以挖掘不同时间段上的频繁模式.IL-TREE 在更新模式时可以快速定位需更新的父模式,并在生成新模式时可以直接定位到需要判断的父模式,从而提高了算法时间效率.为了提高查询精度,FPIL-STREAM 采用了一种启发式倾斜窗口,当生成新的模式时,它可以更为准确地估计相应的窗口,并可以提供更细的窗口粒度,提高了查询结果的精确性.

1 背景知识

1.1 概念与定义

设 I_i 为第 i 个项,$I=\{I_1,I_2,\ldots,I_m\}$为 m 个项的集合.$T=\{T_1,T_2,\ldots,T_n\}$为事务集,其中事务 $T_i(i=1,2,\ldots,n)$是 I 的子集,即 $T_i\subseteq I$.事务可以由事务 ID 或事物本身所包含的项集来表示,如 T_1 或$\{I_1,I_2,I_3\}$.模式 P 是 I 的子集,且存在 $T_i(0<i\leq n)$使 $P\subseteq T_i$.如果模式 P 的支持度大于等于指定的阈值,则称为频繁模式.模式的长度为模式所包含的项的个数,长度为 k 的模式称为 k-模式.

定义 **1**(父/子模式). 设 k-模式 $P=\{I_1,I_2,\ldots,I_k\}$,对于 k'-模式 P',如果 $k'<k$,且$\forall I_i(I_i\in P')$,均有 $I_i\in P$,则称 P'为 P 的父模式,P 为 P'的子模式.

为了与树中父/子节点关系以及模式的派生关系对应,这里的父/子模式定义与 Aprior 算法[5]中的定义相反.

定义 **2**(数据流 ***D***). 数据流 D 是由不断到达的事务组成的动态增长事务集,即 $D=\{T_1,T_2,\ldots,T_i,\ldots,T_j,\ldots\}$,其中 T_i 为事务,如果 $j>i$,则 T_i 先于 T_j 到达;如果$|j-i|=1$,则 T_i 与 T_j 相邻.

定义 **3**(子窗口 ***w*** 与窗口 ***W***). 子窗口 w 是在一段时间内连续到达的事务集合,$w=\{T_i,T_{i+1},\ldots,T_j\}(0<i\leq j)$,其中 T_i 是起始事务,T_j 是截止事务.窗口 W 由一系列连续相邻的子窗口构成,$W=\{w_1,\ldots,w_i,\ldots,w_j\}$,并且$\forall i,w_i\subset W;\forall i,j,i\neq j,w_i\cap w_j=\varnothing$,如果 $j>i$,则 w_i 中的所有事务先于 w_j 到达;如果$|j-i|=1$,则 w_i 与 w_j 相邻.

子窗口 w 和窗口 W 的宽度为其包含的事务数,分别记为$|w|$和$|W|$.

定义 **4**(窗口支持度). 设 $f_P(W)$为窗口 W 内模式 P 的频数,$S_p(W)=f_P(W)/|W|$称为模式 P 在窗口 W 内的支持度.

由于本文讨论的情况都是在一定窗口下的,在不引起歧义的情况下,窗口支持度简称为支持度.

定义 **5**(窗口内频繁模式). 给定窗口 W,如果模式 P 的窗口支持度 $S_p(W)$大于等于指定支持度 $S(0<S<1)$,则称 P 是 W 内频繁模式,简称 P 在 W 内频繁.

在实际应用中,由于受到存储空间和运算速度的限制,一般频繁模式挖掘算法很难保证发现所有的频繁模式,为了提高挖掘准确性,通常由用户指定允许误差$\varepsilon(0<\varepsilon<1)$,并在算法中以较小支持度 $S\times\varepsilon$代替 S,以下本文在未作特殊说明的情况下,所述的频繁模式和窗口频繁模式是指支持度大于 $S\times\varepsilon$的模式.

1.2 相关技术与工作

数据流环境下的频繁模式挖掘效率和查询精度与两方面有关:模式的存储结构和历史数据的记录方式.目前较好的解决方法是采用字典树和倾斜时间窗技术.字典树(lexicographic tree)[6]是 Agarwal 等人提出的模式集的一种压缩存储结构,其建树规则为:设频繁 k-模式 $P=\{I_1,I_2,\ldots,I_k\}$,其中各项按全序关系 $\prec$ 有序,对所有 k'-频繁模式 $P'\subseteq P,k'=1\sim k$,若 $P'=\{I_i,I_{i+1},\ldots,I_{k'}\}$,则节点 P'的父节点对应模式为$(k'-1)$-模式$\{I_i,I_{i+1},\ldots,I_{k'-1}\}$.本文简称对应模式 P 的节点为节点 P,若事务 T 包含模式 P,则称事务 T 涉及节点 P.在基于字典树的启发式频繁模式挖掘算法中,判断模式 P 被字典树包含的条件之一是 P 的所有父模式都已包含于树中.但在判断时,除了父节点,只能对树

进行遍历查找 P 的其余父模式.尽管由于存在关系 $\prec$,只需遍历部分子树,但效率仍然很低.另外,字典树结构通常较宽,其更新现有模式时横向查找耗时较多.倾斜窗口策略是根据现实中人们对旧知识兴趣度较低、对新知识兴趣度高的假设,以不同粒度压缩存储历史数据,这样既保留了历史细节,又节省了空间.如 Giannella[1]提出的倾斜窗口策略用于批处理方法,其子窗口基本粒度等于数据流分片的大小.在文献[2]中提到,批处理方法的数据流分片如果过小,挖掘模式总数会大量增长,而固定阈值下的频繁模式集不变,故执行效率很低.因此,Giannella 的方法中窗口宽度的基本单位较大.

数据流中挖掘任意长度频繁模式算法主要有两种:批处理方法和启发式方法.批处理的方法主要思想是对数据流分片,通过分片上的局部频繁模式来求解全局频繁模式,其中有代表性的是 Giannella 等人提出的 FP-Stream 算法[1]和 Manku 提出的 Lossy Counting 算法[2].FP-Stream 算法以一个基于频数序的字典树来存储频繁模式,并利用倾斜时间窗记录模式的历史信息来回答用户对各个时间段的查询.FP-Stream 时间效率较好,但由于采用批处理方式,不能实时处理流数据和响应用户对当前数据的查询,并且算法时间窗口的粒度较粗,查询精度较低.Counting 算法与 FP-Stream 算法近似.启发式方法的主要思想是随着流数据的不断到达,由已知频繁模式逐步生成新的频繁模式.由于在启发式挖掘频繁模式过程中,长度为 k 的模式只有在其所有长度为 k–1 的父模式都被发现的前提下,才能被考虑,因而存在模式计数偏小的问题.Hidber 提出的 Carma 算法[3]对这个问题进行了改进,提出以父模式的频数估计新产生模式的频数,但算法中通过第二次扫描数据来保证最大误差,从而使其在数据流挖掘中的应用受到限制.Chang 等人的 estDec 算法[4]采用了字母序的字典树存储模式,应用了类似 Carma 算法的估计频数思想,并通过按时间指数进行衰减的策略,保证只挖掘最近发生的频繁模式.该算法也是以新模式的父模式中的最小计数来近似代替新模式的实际频数,并由阈值 S 与模式长度 k 约束频数上限.

与批处理方法相比,启发式方法不需要积累数据,具有较好的实时性和查询精度.但在判断字典树是否增加节点以及更新节点的代价较大,且模式统计计数不包含详细历史信息,使得模式估计与查询精度较低.

2 FPIL-STREAM

2.1 IL-TREE

定义 **6**(直接/间接父模式). 给定字典树 T,设 P'为 k-模式 P 的(k–1)-父模式,若在字典树 T 中 P'为 P 的父节点,则称 P'为 P 的直接父模式,否则称 P'为 P 的间接父模式.

由启发式算法的基本思路可知,生成 k-模式 P 只需考虑(k–1)-父模式.以下简称 k-模式 P 的(k–1)-父模式为 P 的父模式.同理,简称 k-模式 P 的(k+1)-子模式为 P 的子模式.

定义 **7**(**IL-TREE**). 给定字典树 T,对 T 中任意 k-模式 P(k>1),令 P 包含指向其间接父模式的指针(Plink),称 T 为 IL-TREE.

IL-TREE 中节点 Node 结构有以下几个域:(1) ChildList,指向子节点的指针链表;(2) Item,为节点所包含的项,简称节点项,根节点项为空;(3) CountWindow,为该节点倾斜窗口;(4) Parent,指向父节点的指针;(5) Plink Array,指向间接父模式指针数组;(6) TempArray,是随事务而变化的指针数组,包含指向当前事务所涉及到的子节点的指针.若子节点不存在,则数组中的对应项为空.TempArray 长度为事务长度减去节点项在事务中的序号.当节点为根节点时,TempArray 数组长度等于事务长度.不失一般性,本文中 IL-TREE 采用字母序.图 1 为一个 IL-TREE 例子,其中节点 5 代表模式{a,c},节点 7 代表模式{a,b,c}.IL-TREE 具有如下性质:

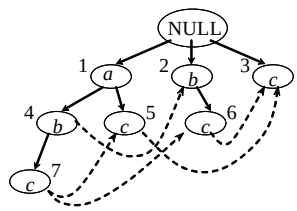

Fig.1 IL-TREE structure

图 1 IL-TREE 结构

性质 **1**. 对 IL-TREE 中的 k-模式 P={I_1,I_2,…,I_k},节点 P 只需包含(k–1)个 Plink 且 P 的 Plink 指向的节点所包含项为 I_k(证明略).

2.1.1 IL-TREE 节点更新

更新现有节点的主要过程是结合 Plink 结构,深度优先遍历 IL-TREE,沿途更新相关节点倾斜时间窗,并将

节点位置写入其父节点的 TempArray 数组.这里给出递归算法.

算法 **1**. IL-TREE 更新算法 Update(*node*,*T*).

输入:IL-TREE 结点 *node*,事务 $T=\{I_1,I_2,\ldots,I_k\}$;

输出:更新现有节点后的 IL-TREE.

```
1 for (node 的所有间接父模式节点 Father)
2     if (Father. CountWindow 的截止事务不为 T){                    //未被 T 更新
3         更新 Father. CountWindow;初始化 Father. TempArray;
4         将 Father 的位置记入其父节点 TempArray 数组的相应位置;}   //父节点 TempArray 已初始化
5 if (node. CountWindow 的截止事务不为 T){                          //未被 T 更新
6     更新 node.CountWindow;初始化 node.TempArray;}
7 for (T 的所有项 I_i,i 为项的序号)
8     if (node.TempArray [i]=NULL){                                 //Child 未作为间接父模式节点被更新过
9         if (存在 node 的子节点 Child.Item＝I_i){
10            令 node.TempArray [i]指向 Child;Update (Child,{I_{i+1},…,I_k});}}
11     else {Update (node.TempArray [i]所指向的子节点,{I_{i+1},…,I_k});}
12 if (node.TempArray 中存在 NULL) { push node to Queue;}          //node 节点可能生成新的子节点
```

其中,步骤 3、步骤 6 中倾斜窗口更新是指将窗口中包含事务 *T* 的子窗口的频数加 1,然后令其截止事务为事务 *T*.如最近的子窗口的截止事务的序数整除子窗口基本粒度,则在原窗口截止事务一端增加新的子窗口.在步骤 12 中,对于可能有子节点生成的节点,使用全局队列 Queue 保存,当 IL-TREE 扩展时,用于判断子节点是否生成.

2.1.2 IL-TREE 扩展

IL-TREE 的扩展算法采用较低支持度阈值 S_{in} 代替用户指定的支持度 *S*,这里取 $S_{in}=S\times\varepsilon(0<\varepsilon<1)$.子窗口基本粒度 *N* 为 $1/S_{in}$,即每个新发现的模式都被认为频繁.这样,当增加模式时,根据性质 1,如果所考察的节点均存在项为 I_k 的子节点,则该模式可以插入.新模式的节点项为 I_k,其 Plink 分别指向已判断存在的子节点.

算法 **2**. IL-TREE 扩展算法 ExtendTree().

输入:更新原有节点后的 IL-TREE;

输出:插入新节点后的 IL-TREE.

```
1 while (Queue 不为空)                                  // Queue 为全局队列
2   node=pop Queue;                                     // node 为队列头节点,其 Item 为 I_j
3   for (node.TempArray 数组中的每一个元素,i 为其序号){
4       Add= FALSE;                                     //布尔变量 Add 指示是否生成新节点
5       if (node.TempArray [i]指向 NULL){
6           Add=TRUE;node 父节点的 Item 为 I_k;         //若父节点为根节点,k=0
7           if (node 父节点的 TempArray [i+j–k]为 NULL) {Add=FALSE;}
8           if (node.Plink Array 中,存在一个 Plink,其指向一个 TempArray [i]为 NULL 的节点){
9               Add=FALSE;}}                            //性质 2
10      if (Add) {node 下生成节点项为 I_{i+j} 的子节点,调用算法 3 估计其时间窗;}}
```

2.2 启发式倾斜窗口策略

启发式的倾斜窗口策略主要包括两部分:一部分是因为随着时间变化,模式的频繁情况也会改变,要根据模式的倾斜窗口对 IL-TREE 进行修剪;另一部分是因为较长的模式发现得较晚,使得新发现的模式频数偏小,需要根据一定的策略来对新模式的倾斜窗口进行估计.本文在 Giannella 提出的倾斜窗口的基础上,提出一种更加简单、有效的策略:若模式在其窗口长度内为不频繁,则将其删除.

定理 **1**. 设当前时刻 j 下,模式 P 的窗口为 $W=\{T_1,\ldots,T_i,\ldots,T_j\}$,若 P 在 W 内不频繁,则 P 在任意一段时间段 $[i,j]$,$1<i<j$ 内,均不频繁(证明略).

推论.设模式 P'是 P 的子模式,P'的窗口是 W'.若 P 在 W 内不频繁,则 P'在 W'内也不频繁(证明略).

定理 1 的推论说明若删除模式 P,其所有子模式同时也要删除.修剪基本方法是:对于窗口基本粒度 N,当 N 个事务到达后扫描 IL-TREE,对所有未包含在 N 个事务的模式,增加计数为 0 的子窗口,然后根据阈值 S_{out} 判断是否删除,其中 S_{out} 为指定阈值,本文为简化讨论取 $S_{out}=S_{in}$.

定理 **2**. 设模式 P 窗口的起始事务 T_i,模式 P'是 P 的子模式,P'窗口的起始事务为 T_j,则有 $i\leq j$(证明略).

推论. 设 P 的窗口为 W,子模式 P'的窗口是 W',则有$|W|\geq|W'|$(证明略).

定理 **3**. 设模式 P 为新发现的 k-模式,W 是 P 估计的窗口,则应有 $f_P(W)\leq k$(证明略).

由定理 3,设模式 P 为新发现的 k-模式,W 是 P 估计的窗口,则$|W|$应不大于 k/S_{in},否则估计结果没有意义.结合定理 2 的推论,对于窗口 W 的宽度上限有:

$$|W|\leq\min\{\min\{|W'|(W'\text{是 }P\text{ 父模式的窗口})\},k/S_{in}\} \tag{1}$$

新模式的窗口估计算法见算法 3,其中,步骤 2 用式(1)估计新模式窗口大小.步骤 4 中,因 *node* 的 k 个父模式窗口粒度不同,在估计时假设每个子窗口内计数均匀分布,将其全都化成最细粒度,即除了最近的子窗口外,每个子窗口宽度都为 N.由事务 T 生成新模式 P,则 P 的父模式均被 T 更新,其倾斜窗口截止事务同为 T,故 P 的父模式最近子窗口的宽度也相同.由定理 3,在细化后每个子窗口内,P 的计数仍不大于任意一个父模式,并以此来分段估计 W.另外,算法 3 包含子窗口修剪(步骤 11~步骤 12),这是因为 W 按 P 的可能发生情况估计,结果也就可能包含不频繁子窗口,而这些子窗口对算法是无意义的,故要将其去除.

算法 **3**. 新模式的窗口估计算法 Estimate(node).

输入:IL-TREE 结点 *node*;

输出:结点 *node* 估计的窗口.

```
1  根据 node 的 Plink 找到 node 的全部父模式,假设 node 有 k 个父模式;
2  估计子窗口数 Width= min{min {|W'|(W'是 node 的父模式的时间窗) },k/S_in }/N;   //式(1)
3  For (node 的所有父模式 Parent)
4      node.Parent.CountWindow={…w_i,…,w_width-1,w_width}                    //窗口细化
5  最大遗漏次数 Lost=k;
6  While (Width>0 并且 Lost>0){                    //达到子窗口数或遗漏次数上限,估计过程结束
7      f_P (w)=min{min{ node 的所有父模式的 f_P (w_width)},Lost};            //估计子窗口频数
8      node. CountWindow=node. CountWindow∪w;                                //子窗口数增加
9      Lost=Lost−f_p(w);                                                     //遗漏次数递减
10     Width--;}                                                             //逐渐接近宽度上限
11 While(f_P(W)/|W|<S)                                                       //修剪不频繁子窗口
12     在 node. CountWindow 中去掉最久的子窗口;
```

2.3 数据流频繁模式挖掘算法

数据流频繁模式挖掘算法 FPIL-STREAM 的主要过程是当一个事务 T 到达后,根据事务 T 对 IL-TREE 进行更新与扩展,使到目前为止的所有频繁模式均包含于树中.每隔 N 个事务,算法对 IL-TREE 进行扫描,维护树中模式的窗口倾斜化以及修剪不频繁模式.

子窗口基本宽度单位 N 为 $1/S_{in}$,原因如下:在启发式频繁模式挖掘过程中,1-模式 P 由于没有父模式存在,故加入 IL-TREE 只需判断 P 是否频繁.而且 P 的倾斜窗口也无须估计,只需简单将起始事务置为最近一次扫描 IL-TREE 的事务,截止事务置为当前事务.若 N 较大(大于 $1/S_{in}$),P 要发生超过一次才能加入 IL-TREE.这使得以后对 2-模式的窗口估计偏大若取 N 为 $1/S_{in}$,由定理 3 可以保证其频数不会估计得过大.N 值越小,查询精度越高.虽然取 N 小于 $1/S_{in}$,定理 3 仍然成立,但为维护窗口倾斜化,扫描 IL-TREE 的次数会增多,算法效率将显著下降.

故在 FPIL-STREAM 中,取 N 为 $1/S_{in}$.具体算法见算法 4.算法 4 始终维护了蕴含频繁模式的 IL-TREE,在任意时刻通过已指定阈值遍历树或按字典序索引查找,可以得到所需的频繁模式集,再由模式自身的倾斜窗口可以查询其历史频繁信息.

算法 **4**. FPIL-STREAM.

输入:数据流 $D=\{T_1,T_2,\ldots,T_n\}$;

输出:蕴含频繁模式的 IL-TREE.

```
1   For (D 中每个事务 T_i)
2   Update(Root,T_i);
3   ExtendTree(Root,Queue);
4   if (i mod N=0)                                  //每 N 个事务(窗口基本宽度)对树进行一次剪枝
5       for (树中每个节点 node){
6           if (node 为新节点或被最近 N 个事务更新)
7               令 node. CountWindow 为 T_i;
8       else{
9               生成子窗口 w,其截止事务为 T,令 f_p(w)=0;
10              node.CountWindow = node.CountWindow∪w;
11              if (f_p(node.CountWindow)< S_out)
12                  删除 node 及其所有子节点;}
13          if (node 未被删除)                      //node 频繁
14              对 node. CountWindow 进行倾斜化}
```

上面步骤 2 中,由 Apriori 原理[5],更新频繁模式时,其子集也是频繁的,也要进行更新.利用 IL-TREE 的性质 1,即节点与其 Plink 节点的项相同,对模式 P 的子模式进行快速查找.由 Plink 结构可知,其与 TempArray 结合直接定位子模式,要比在树中横向查找更快.步骤 3 中,由 IL-TREE 性质 1 可知,节点与其 Plink 节点所包含项相同.故在判断 Plink 时,根据节点 TempArray 数组空值的位置,直接查找 Plink 的 TempArray 数组中相应位置的元素.另外,子窗口基本粒度 N 设为 $1/S_{in}$,较批处理方法下的窗口粒度要细,而因为启发式挖掘频繁模式时,只有少量无用模式进出 IL-TREE,这样算法可以在保证较高时间效率的同时提高查询精度.步骤 9~步骤 12 为模式的修剪.步骤 14 中,对 IL-TREE 中未被删除的节点,按文献[1]的方法进行窗口倾斜化.

3 性能分析

因为 FP-Stream 算法[1]的时间效率较好,是目前较为流行的算法,我们选择 FP-Stream 作为对比算法.实验环境为 DELL PC 机,CPU 为 PentiumIII-933MHz,内存 384M,操作系统为 Windows 2000.实验数据采用文献[7]提供的数据生成器,分别生成虚拟数据集 Set1,Set2,Set3 和 Set4.一般地,在其他参数相同、而项数较多的情况下,启发式算法效率较好[3].为体现 FPIL-STREAM 算法的改进效果,实验数据集的项数选取了较小的数值,参照文献[1],各个数据集的事务平均长度分别为 3,5,7 和 10,项数取 1K,事务数 1 000K,其余参数取默认值.在所有实验中,误差 ε 固定为 10%.

首先,我们比较了在同一个数据集和相同的支持度阈值下,算法对每 100K 事务的平均处理时间.若数据集中事务平均长度过短,挖掘结果意义不大,而 FP-Stream 算法中事务平均长度最长的数据集为 Set3,所以二者只在数据集 Set3 上进行比较.图 2 给出在数据集 Set3 上,支持度阈值 S=0.5%,FPIL-STREAM 算法与 FP-Stream 算法随事务到达的平均处理时间比较.实验结果表明,FPIL-STREAM 算法优于 FP-Stream 算法.

然后,我们对二者在不同阈值下每 100K 事务的平均处理时间进行比较.图 3 给出二者在数据集 Set3 上、不同阈值下平均处理时间的比较.结果表明,FPIL-STREAM 算法在较低阈值下,平均处理时间要优于 FP-Stream 算法.在高阈值下,查找父模式的次数随着所挖掘模式的平均长度的缩短而减小,使得 IL-TREE 改进效果不明显.而且,由于启发式算法本身的局限,处理时间随阈值增高下降幅度偏小.并且 FPIL-STREAM 算法所取窗口基

本宽度以及扫描树结构次数与阈值有关,使得高阈值下计算开销加大.

最后,我们比较了二者在不同的数据集上每 100K 事务的平均处理时间的比较.图 4 给出了二者在固定阈值 S=0.5%、不同数据集上平均处理时间的比较.在事务平均长度较长的情况下,FPIL-STREAM 算法平均处理时间要优于 FP-Stream 算法,FP-Stream 算法只在事务平均长度小于 5 时才稍有优势.这是由于在较短的事务平均长度下,所挖掘的模式长度也较短,使得 IL-TREE 改进效果不明显,与高阈值下的情况类似.

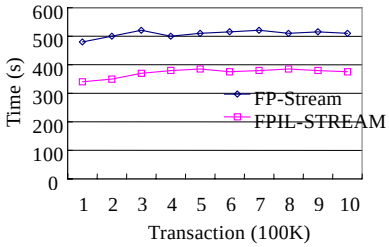

Fig.2 Runtime on set3 with S=0.5%

图 2 S=0.5%,两种算法在 Set3 数据集上的运行时间

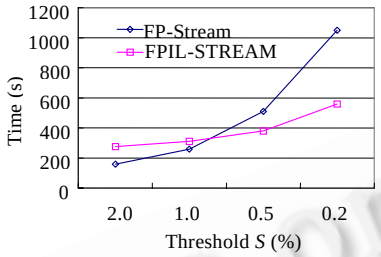

Fig.3 Runtime on set3 with different threshold

图 3 不同支持度阈值下两种算法在 Set3 数据集上运行时间

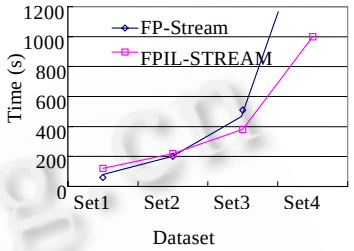

Fig.4 Runtime on different dataset with S=0.5%

图 4 S=0.5%,两种算法在不同数据集上运行时间

4 结 论

本文提出了一种快速挖掘数据流中的频繁模式的方法,其主要贡献在于:(1) 提出了一种改进字典树结构 IL-TREE,并设计该结构的快速启发式增长算法,可以高效地更新现有模式以及在生成新模式时直接定位父模式,时间效率高;(2) 将倾斜窗口思想与启发式方法结合,提出改进的倾斜窗口策略,细化窗口粒度,使得查询更加精确;(3) 结合 IL-TREE 与改进的倾斜窗口策略,提出数据流频繁模式挖掘算法 FPIL-STREAM,可以同时保证执行速度与查询精度.实验证明,算法的时间效率较好,与较快的批处理算法相比也有优势,且本文给出的是递归算法,时间效率还有很大的可提高空间.

References:

[1] Giannella C, Han JW, Pei J, Yan XF, Yu PS. Mining frequent patterns in data streams at multiple time granularities. http://maids.ncsa.uiuc.edu/documents/readings/fpstm03.pdf

[2] Manku GS, Motwani R. Approximate frequency counts over data streams. In: Bernstein P, Ioannidis Y, Ramakrishnan R, eds. Proc. of the 28th Int'l Conf. on Very Large Data Bases. Hong Kong: Morgan Kaufmann Publishers, 2002. 346−357.

[3] Hidber C. Online association rule mining. In: Delis A, Faloutsos C, Ghandeharizadeh S, eds. Proc. of the ACM SIGMOD Int'l Conf. on Management of Data (SIGMOD 1999). Philadelphia: ACM Press, 1999. 145−156.

[4] Chang J, Lee W. Finding recent frequent itemsets adaptively over online data streams. In: Lise G, Ted E. S, Pedro D, Christos F, eds. Proc. of the 9th ACM SIGKDD Int'l Conf. on Knowledge Discovery & Data Mining. Washington: ACM Press, 2003. 226−235.

[5] Agrawal R, Srikant R. Fast algorithms for mining association rules. In: Beeri C, *et al*., eds. Proc. of the 20th Int'l Conf. on Very Large Databases. Santiago: Morgan Kaufmann Publishers, 1994. 487−499.

[6] Agarwal RC, Aggarwal CC, Prasad VVV. A tree projection algorithm for finding frequent itemsets. Journal on Parallel and Distributed Computing, 2001,61(3):350−371.

[7] http://www.almaden.ibm.com/software/quest/Resources/index.shtml