ISSN 1000-9825, CODEN RUXUEW
Journal of Software, Vol.19, No.10, October 2008, pp.2585−2596
DOI: 10.3724/SP.J.1001.2008.02585

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel/Fax: +86-10-62562563

挖掘数据流任意滑动时间窗口内频繁模式*

李国徽, 陈 辉+

(华中科技大学 计算机科学与技术学院,湖北 武汉 430074)

Mining the Frequent Patterns in an Arbitrary Sliding Window over Online Data Streams

LI Guo-Hui, CHEN Hui+

(School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan 430074, China)

+ Corresponding author: E-mail: chen_hui@smail.hust.edu.cn

Li GH, Chen H. Mining the frequent patterns in an arbitrary sliding window over online data streams. *Journal of Software*, 2008,19(10):2585−2596. http://www.jos.org.cn/1000-9825/19/2585.htm

Abstract: Because of the fluidity and continuity of data stream, the knowledge embedded in stream data is most likely to be changed as time goes by. Thus, in most data stream applications, people are more interested in the information of the recent transactions than that of the old. This paper proposes a method for mining the frequent patterns in an arbitrary sliding window of data streams. As data stream flows, the contents of the data stream are captured with a compact prefix-tree by scanning the stream only once. And the obsolete and infrequent items are deleted by periodically pruning the tree. To differentiate the patterns of recently generated transactions from those of historic transactions, a time decaying model is also applied. Extensive simulations are conducted and the experimental results show that the proposed method is efficient and scalable, and also superior to other analogous algorithms.

Key words: data stream; frequent pattern mining; sliding window; time decaying model

摘 要: 由于数据流的流动性与连续性,数据流所蕴含的知识会随着时间的推移而发生变化.因此,在绝大多数数据流的应用中,用户往往对新产生的流数据所包含的知识要比对历史流数据所包含的知识感兴趣得多.提出了一种挖掘数据流任意大小滑动时间窗口内频繁模式的方法 MSW(mining sliding window).当数据流流过时,该方法使用滑动窗口树 SW-tree 在单遍扫描流数据的条件下及时捕获数据流上最新的模式信息.同时,该方法还周期性地删除滑动窗口树上过期的及不频繁的模式分支,从而降低滑动窗口树的空间复杂度与维护代价.此外,该方法还应用时间衰减模型逐步降低历史事务模式支持数的权重,并由此来区分最近产生事务与历史事务的模式.大量仿真实验的结果表明,算法 MSW 具有较高的效率与优良的可扩展性,同时也优于其他同类算法.

关键词: 数据流;频繁模式挖掘;滑动时间窗口;时间衰减模型

中图法分类号: TP311 **文献标识码**: A

* Supported by the National Natural Science Foundation of China under Grant No.60873030 (国家自然科学基金); the National High-Tech Research and Development Plan of China under Grant No.2007AA01Z300 (国家高技术研究发展计划(863))

Received 2007-11-08; Accepted 2008-01-08

1 介 绍

数据流[1,2]是一个大小可能无限的数据序列,其中的数据元素连续高速到达.因此,在数据流处理中,不可能将所有的流数据元素全部存放到内存中进行处理,而数据流的算法也应该满足如下几个条件[3]:第一,分析数据流时,所有的数据元素最多只能访问 1 次;第二,尽管数据流中数据元素在连续不断地产生,但是分析数据流所需要的内存空间应该是有限的;第三,新产生的流数据必须能够尽快地处理;第四,当用户提交查询时,需能及时反馈最新的数据流分析结果.

挖掘数据流上的频繁模式能够为数据流应用提供重要的决策依据.但是,由于数据流的流动性与连续性,数据流上频繁模式信息随着流数据的连续产生而不断发生变化.在大多数数据流的应用中,用户往往更加关注数据流上最近事务数据所包含的模式信息.因此,挖掘数据流上最近的频繁模式成为流数据挖掘中的一项具有挑战性的工作.近年来,数据流上频繁模式挖掘工作受到广泛的关注,并取得了很多成果[4−10].Giannella 等人[4]提出了 FP-stream 算法来挖掘数据流上多个时间粒度的频繁模式.该方法应用倾斜时间窗口[4]策略以精细的时间粒度保存数据流上最近的频繁模式信息,而以粗糙的时间粒度保存历史的频繁模式.Chang 与 Lee[5]提出 estDec 算法来挖掘数据流上最近的频繁模式,该方法应用时间衰减机制逐步减少历史模式的支持数来区分最近事务的模式与历史事务的模式.Leung 与 Khan[8]提出了一种被称为 DStree 的模式树来维护数据流历史时间窗口内的模式信息.该模式树上的每个节点都使用多个计数器来分别记录该模式在数据流最近的多个数据分段内的支持数.当数据流流过时,该方法可以精确地挖掘数据流上最近几个数据分段内的频繁模式.

然而,上述的 FP-stream[4],estDec[5]与 DStree[8]算法仍然存在着明显的缺陷.首先,estDec 算法所使用的监测树(monitor tree)是一种松散的数据结构,未能有效利用模式树的存储空间,空间复杂度大.此外,当处理一个长度为 l 的事务时,该方法需要产生并测试 2^l−1 个中间候选模式,故算法的时间代价也很大.其次,在 DStree 算法中,模式树上的每个节点均使用多个计数器来分别记录模式在数据流不同数据分段内的支持数.尽管该方法能够精确挖掘数据流上的频繁模式,但是,当数据流模式挖掘的时间窗口大小是任意的或者不能事先确定时,该方法就显得非常不灵活.并且使用多个计数器也增加了模式树的空间复杂度.此外,当数据流流动时,不频繁模式以及历史事务的模式所对应的模式分枝一直保留在模式树上.随着流数据的不断产生,模式树的空间复杂度将迅速增加,因此,算法的可扩展性差.第三,尽管 FP-stream 方法仅需要维护数据流各个数据分段内的频繁模式,但是 FP-stream 算法中的模式树也是一种松散的树结构.另外,FP-stream 方法不仅需要维护数据流全局上的频繁模式树,还需要额外的时间与空间为数据流各个数据分段构建一个 FP-tree 树,然后从该 FP-tree 树上输出频繁模式并保存到全局模式树上.随着已处理数据分段数目的增多,FP-stream 算法在更新全局模式树时需要多次移动与合并各个节点上的倾斜时间窗口.更为重要的是,FP-stream 需要两次访问流数据元素才能捕获数据流上的频繁模式,这也在很大程度上限制了该方法挖掘高速数据流的能力.

本文提出了一种能够高效挖掘数据流任意大小滑动时间窗口[11−13]内频繁模式的方法 MSW(mining sliding window).该方法使用结构紧凑的滑动窗口树 SW-tree 来压缩存储数据流滑动时间窗口内的频繁模式;当数据流流过时,仅需单遍访问流数据元素,滑动窗口树可以增量捕获数据流上最新的模式信息.同时,方法还周期性地对滑动窗口树进行剪枝,删除那些过期的和不频繁的(infrequent)模式所对应的分支,从而减小滑动窗口树的空间复杂度与维护代价.此外,该方法还通过减少历史事务模式支持数的权重方式来区分历史事务模式与新产生事务模式.与 FP-stream,estDec 和 DStree 方法相比,MSW 方法具有以下优点:第一,与其他 3 种方法所使用的模式树相比,MSW 方法的滑动窗口树 SW-tree 更能高效地维护数据流上的模式信息;第二,MSW 方法具有更强的可扩展性和适应性;第三,MSW 方法更适合于挖掘数据流上任意大小滑动时间窗口内的频繁模式.

本文第 2 节介绍挖掘数据流频繁模式的预备知识.第 3 节详细讨论滑动窗口树以及挖掘数据流任意大小滑动时间窗口内频繁模式挖掘的算法.第 4 节分析算法的正确性.第 5 节通过仿真实验学习算法的性能,并将算法与其他同类算法进行对比分析.第 6 节总结全文.

2 预备知识

本节主要介绍挖掘数据流滑动时间窗口内频繁模式的相关概念,并提出模式支持数的衰减机制.

2.1 相关概念

数据流 $DS=\{T_1,T_2,\ldots,T_i,\ldots\}$为一个大小可能无限的事务数据序列,其中 T_i 为第 ith 个产生的事务.数据流中任何事务均包含唯一的事务标识 *tid*,任何事务中的数据元素集合均为数据集 $A=\{a_1,a_2,\ldots,a_m\}$的一个子集,其中 A 是一个按照全序 $\prec$ 有序($\prec$为一种预定义的排序关系)的有限数据集合.滑动时间窗口 *SW* 包含数据流 *DS* 上最近的 N 个事务数据,N 表示 *SW* 的大小.任何模式 P 都为数据集 A 的一个子集,P 中包含元素的数目称为模式 P 的长度,记为$|P|$.长度为$|P|$的模式称作$|P|$-模式.如果一个事务 T 包含模式 P 中的所有数据元素,则称该事务包含模式 P.模式 P 在滑动时间窗口 *SW* 内的支持数等于 *SW* 内所有包含模式 P 的事务数据的个数,记为 $freq(P,SW)$.

定义 1. 将事务 $T=\{a_1,a_2,\ldots,a_k\}$中的数据元素按照全序 $\prec$ 重新排列后得到的新数据序列 $T'=\{a_1',a_2',\ldots,a_k'\}$称为事务 T 的投影.如果没有特别说明,下文中我们不区分事务和事务的投影.

定义 2. 对于一个事务 $T=\{a_1,a_2,\ldots,a_k\}$与一个 m-模式 $P=\{b_1,b_2,\ldots,b_m\}(m\le k)$,如果$\forall i,1\le i\le m$,总有 $a_i=b_i$ 成立,则称模式 P 为 T 的 m-前缀模式.如果 $m=k$,则称 P 为事务 T 的最大前缀模式.

定义 3. 给定最小支持度门限$\theta(0<\theta<1)$和最大许可误差$\varepsilon(0<\varepsilon<\theta)$,则对于大小为 N 的滑动时间窗口 *SW* 内任意一个模式 P,(1) 如果 $freq(P,SW)\ge\theta N$,那么 P 为频繁模式;(2) 如果 $freq(P,SW)<\varepsilon N$,那么 P 为不频繁模式;(3) 如果$\varepsilon N\le freq(P,SW)<\theta N$,那么 P 为临界频繁模式;而不频繁模式与临界频繁模式统称为非频繁模式.

在数据流频繁模式挖掘中,尽管我们仅关心那些频繁模式,但是我们还必须维护那些临界频繁模式,因为随着时间的推移,它们很可能会变为频繁模式.此外,我们还必须丢弃那些不频繁模式,因为不频繁模式的数量相当多,维护它们的代价极大.此外,丢弃那些不频繁模式所带来的可能误差不大于ε.因此,在挖掘数据流上的频繁模式时,通常以$(\theta-\varepsilon)$作为最小支持度门限来保证数据流频繁模式挖掘无漏报[7,9].

2.2 时间衰减模型

由于数据流的流动性与连续性,数据流中蕴含的知识会随着时间的推移而发生变化.而在实际的数据流应用中,最近产生事务所蕴含的知识往往要比历史事务的知识有价值得多.因此,在数据流频繁模式挖掘时,人们更希望挖掘出最近产生事务的频繁模式,而忽略那些历史事务的模式.

本文应用时间衰减模型 TDM(time decay model)逐步衰减历史事务模式支持数的权重,并由此来区分新产生事务与历史事务的模式.如果记模式支持数在单位时间内的衰减比率为衰减因子$f(0<f\le 1)$,记事务 T_i 到达时模式 P 的衰减支持数为 $freq_d(P,T_i)$,那么,当数据流上的第 1 个事务 T_1 到达时,$freq_d(P,T_1)=r$,当第 2 个事务 T_2 到达时,有 $freq_d(P,T_2)=freq_d(P,T_1)\times f+r$.类似地,当第 ith 个事务 T_i 到达时,模式 P 的衰减支持数可以由下式得到:

$$freq_d(P,T_i)=\begin{cases}r, & \text{if } i=1\\ freq_d(P,T_{i-1})\times f+r, & \text{if } i\ge 2\end{cases},\text{其中 } r=\begin{cases}1, & \text{if } P\subseteq T_i\\ 0, & \text{otherwise}\end{cases}.$$

3 频繁模式挖掘

本节首先介绍数据流上的滑动窗口树 SW-tree,并详细讨论基于该树的数据流滑动时间窗口内频繁模式挖掘的算法 MSW.

3.1 滑动窗口树SW-tree

为了适应挖掘数据流滑动时间窗口内的频繁模式,本文设计了一种被称作为滑动窗口树 SW-tree 的前缀模式树来增量维护数据流上的模式信息.SW-tree 是一种基于频繁模式树 FP-tree 的改进前缀模式树,它在存储结构上明显区别于 FP-tree,具体说明如下:(1) FP-tree 上除根节点外的节点包括 3 个数据域:*item-name*,*count*,*node-link*.而在 SW-tree 上,除根节点外,每一个节点还增加一个新的数据域 *tid*,它用来记录最近的包含该节点所表示模式那个事务的标识(*tid*);(2) FP-tree 各分支上的节点均按其支持数的降序排列,而 SW-tree 各分支上的节

点按照某种预先定义的全序$\prec$关系排列;(3) FP-tree 使用频繁数据项头表(frequent-item-header table)对模式树上的数据项进行索引,且频繁数据项表中的各个数据项也按照其支持数的降序方式排列.而在我们的 SW-tree 中,使用数据项头表(item-list table)对滑动窗口树上的数据项进行索引,并且表中的数据项按照与 SW-tree 相同的全序关系排列;(4) FP-tree 的频繁数据项头表的每一个表项包含 *item-name* 与 *head of node-link* 两个数据域,而 SW-tree 数据项头表还增加了 *tid* 与 *count* 数据域.

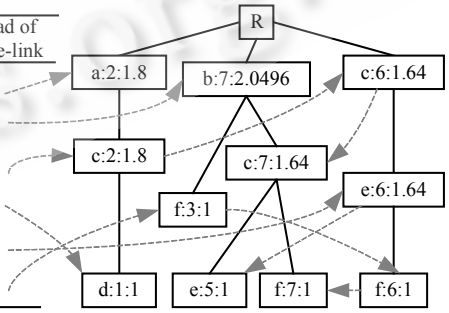
尽管 SW-tree 在存储结构上要比 FP-tree 略微复杂些,但是 SW-tree 却具有能够胜任增量挖掘数据流频繁模式的性质.首先,FP-tree 各分支上的节点均按照其支持度降序的方式排列,使支持度大的数据项有更多的机会共享相同的前缀模式,从而使得模式树的空间代价最小.然而,按照数据项支持数降序的方式排列模式树上的节点,需要至少两次扫描数据集,这在高速的数据流应用中是不允许或者代价是极其昂贵的.而在 SW-tree 上,数据项按照预先确定的全序关系排列.因此,当新的流数据到达时,可以立即将其模式信息增量更新至滑动窗口树上,而不必多次扫描历史的流数据来确定数据项的支持数排列关系.其次,当流数据不断到达时,数据流上数据项支持数的排列顺序是动态变化的.如果仍然按照数据项支持数降序的方式排列模式树各分支上的节点,则必须不断调整模式树上各节点之间的顺序.显然,这样的代价是极其庞大的.而 SW-tree 以预先定义好的全序关系排列滑动窗口树上的数据项,因此,节点之间的排列顺序是固定的,不会随着流数据的到达而发生变化.因此,SW-tree 比 FP-tree 更适合动态维护数据流上的模式信息.第三,SW-tree 上增加的数据域 *tid* 可以帮助计算模式的衰减支持数而无需维护模式在数据流上的详细历史记录.此外,通过检查节点的 *tid*,还可以快速发现滑动窗口树上过期无效的模式分枝.第四,在数据项头表中增加 *tid*,*count* 可以快速发现滑动窗口树上所维护的频繁数据项,这将有助于提高模式输出的效率;同时也有助于快速发现过期与不频繁模式,加快滑动窗口树的剪枝速度.尽管 SW-tree 与 FP-tree 的存储结构存在较大的差异,但是 SW-tree 仍然继承了 FP-tree 的重要性质,并且还具有如下性质:

性质 1. 滑动窗口树 SW-tree 上任何节点的支持数与时间戳都不小于其子孙节点的支持数与时间戳.

合理性.滑动窗口树 SW-tree 采用前缀子模式复用方式压缩存储数据流上的频繁模式,当多个事务包含相同的前缀子模式时,为了压缩滑动窗口树的存储空间,滑动窗口树用同一个分支保存这些相同的前缀子模式.因此,距离滑动窗口树根节点近的节点有更多机会共享前缀子模式.另外,当将一个事务的模式信息增量更新至滑动窗口树上时,总是先更新离根节点近的节点,然后按照事务投影中数据项的排列关系依次更新滑动窗口树上的其他节点.因此,任何事务只要其包含滑动窗口树上某个节点,则必包含该节点的所有祖先节点.综上所述,显然有滑动窗口树上任何节点的支持数与时间戳都不小于其子孙节点的支持数与时间戳.下面我们将通过一个具体的例子来说明 SW-tree.如图 1 所示.

tid	Items	Projection
1	d c a	a c d
2	c a	a c
3	b f	b f
4	c e	c e
5	e b c	b c e
6	e c f	c e f
7	c b f	b c f

Item	tid	Count	Head of node-link
a	2	1.8	
b	7	2.049 6	
c	7	3.541 824	
d	1	1	
e	6	2.44	
f	7	2.209 6	



(a) Example of data stream (b) The SW-tree built on example data stream

(a) 数据流示例 (b) 基于示例数据流的滑动窗口树

Fig.1

图 1

由于滑动窗口树上各分支节点的顺序与数据流中数据项出现的频数无关.当新的事务数据到达时,可以立

即对其进行处理并将其包含的模式信息增量更新到SW-tree上,而不必等到其他事务数据到达后才能确定数据项之间的排序关系,并且流数据的增量处理仅需要单遍扫描流数据.另外,滑动窗口树上各个节点都记录着包含该节点数据项最近的那个事务的 *tid*,当需要计算节点所表示模式的衰减支持数时,可以方便地确定模式支持数衰减的比重;同时,也便于区分滑动窗口树上那些过期模式所对应的分支.

当数据流流过时,滑动窗口树SW-tree仅需要维护数据流滑动时间窗口 *SW* 内事务的模式,因此,其树的大小不大于 $\sum_{T_i \in SW}|Length(T_i)|$,高度不大于 $\max(|Length(T_i)|+1)$,这里,T_i 为滑动时间窗口 *SW* 中的一个事务,$|Length(T_i)|$表示 T_i 中数据项的个数.

3.2 增量更新

随着时间的推移,新的事务数据进入滑动时间窗口 *SW* 而历史的事务数据从 *SW* 中移出,因此,*SW* 内事务数据的集合在不断地发生变化.为了实时地维护 *SW* 内事务数据集最新的模式信息,当新的事务数据到达时,必须及时对其进行处理,并将其模式信息增量更新至滑动窗口树 SW-tree 上.此外,在滑动窗口树 SW-tree 上,由于各分支上节点排列的顺序与各数据项在数据流中出现的先后顺序及出现的频率无关.因此,流数据的处理及滑动窗口树的增量更新不依赖数据流中未来达到的流数据.当新的事务数据 $T=\{a_1,a_2,\ldots,a_k\}$ 到达时,可以执行下面的算法1,将事务 *T* 所包含的模式信息增量更新至滑动窗口树 SW-tree 上.

增量更新滑动窗口树的算法如图2所示.如果用$|Update(a)|$表示将单个数据元素 *a* 更新至滑动窗口树上的代价,则将事务 *T* 更新至滑动窗口树上的代价为 $O(|Update(a)|\times|Length(T)|)$,其中,$|Length(T)|$表示事务 *T* 中数据元素的数目.

算法 1. 增量更新滑动窗口树 IncrementUpdate(*T*,*Root*).

输入参数:$T=\{a_1,a_2,\ldots,a_k\}$,待处理的事务数据 Root,滑动窗口树 SWP-tree 的根节点.

输出结果:更新后的滑动窗口树.

```
1   定义 SWP-tree 上的初始插入位置 InsertNode,并初始化 InsertNode 为 Root;
2   for (int l=1;l≤k;l++){
3       获取事务 T 的 l-前缀模式 P;
4       在 InsertNode 的孩子节点中查找能够表示模式 P 的节点;
5       if (查找成功,且称该节点为 node){
6           更新该节点的计数器与时间戳;
7       } else {
8           创建一个新的节点 node,并将其插入到 SWP-tree 上作为 InsertNode 的孩子节点,用于表示模式 P;
9           完成数据项头表与节点 node 的指针连接;
10      }
11      set InsertNode=node;
12  }
```

3.3 快速剪枝

为了不丢失数据流上的模式信息,在滑动窗口树增量更新时,事务数据所包含的不频繁模式也同时保存到滑动窗口树上.随着时间的推移,不频繁模式的数量将迅速增加,而导致滑动窗口树的空间复杂度及维护代价大为增加.另外,当历史事务从滑动时间窗口中移出时,保存在滑动窗口树上的一些模式也可能变成过期模式.这些过期的或者不频繁的模式不仅对挖掘数据流滑动时间窗口内频繁模式没有任何帮助,反而会增加滑动窗口树的维护代价及模式输出的代价.为了减少维护滑动窗口树的时间与空间代价,必须定期地对滑动窗口树进行剪枝,删除树上那些过期的和不频繁模式所对应的分枝.

如果记滑动时间窗口 *SW* 的大小为 *N*,剪枝开始时数据流的当前事务为 T_n,那么可以分3种情况对滑动窗口树上进行剪枝:

(1) 对于数据项头表中任意一个数据项 *item*,如果$|T_n.tid-item.tid|\geq N$,那么滑动窗口树上所有与 *item* 同名的节点均记录着过期的模式;

(2) 对于数据项头表中任意一个数据项 *item*,如果$|T_n.tid-item.tid|<N$,但$item.count\times f^{|T_n.tid-item.tid|}<\varepsilon N$,那么滑动窗口树上所有与 *item* 同名的节点均记录着不频繁模式;

(3) 对于数据项头表中任意一个数据项 *item*,*node* 为滑动窗口树上与 *item* 同名的节点,如果$|T_n.tid-item.tid|<N$, $item.count\times f^{|T_n.tid-item.tid|}\geq\varepsilon N$,但是$|T_n.tid-tNode.tid|\geq N$,则节点 *tNode* 记录着过期模式.

对于第一和第三种情况,根据性质 1,如果某个节点 *tNode* 记录着过期模式,则其子孙节点均记录着过期模式.在滑动窗口树剪枝时,可以直接删除以该节点为根的子树.此外,如果$tNode.count\times f^{|T_n.tid-tNode.tid|}\geq\varepsilon N$,则还说明节点 *tNode* 所记录的模式曾经频繁出现.尽管此时节点 *tNode* 的祖先节点不应该被剪枝,但是那些包含节点 *tNode* 的历史事务对它们支持数的影响仍然较大.因此,还需要根据下式更新节点 *tNode* 所有祖先节点的支持数以消除影响:

$$node_i.count=node_i.count-tNode.count\times f^{|T_n.tid-tNode.tid|}\geq\varepsilon N.$$

式中,$node_i$ 为节点 *tNode* 的一个祖先节点.显然,删除滑动窗口树上过期模式节点仅仅消除了历史事务对当前模式挖掘结果的影响,而不会引入模式挖掘的误差.

对于第 2 种情况,由于滑动窗口树与数据项头表中的数据元素均按照预定义的顺序排列.因此,如果某个节点记录着不频繁模式,我们仍然不能保证其子孙节点所表示数据项也是不频繁的.因此,对于不频繁模式节点情况,只能删除该节点的信息,而不能删除其子孙节点.此外,尽管节点所记录的模式信息仍然属于滑动窗口内事务的模式信息,也是有效的,但是根据下面的定理 1,可以删除这类节点,并且由于删除这些节点所带来的误差很小以至于可以忽略.

定理 1. 如果 *item* 为滑动窗口内的一个不频繁数据项,那么删除 *item* 给滑动窗口内频繁模式挖掘带来的误差将不超过ε.

证明:假设将滑动时间窗口 *SW* 分割成一系列的子窗口$SW_1,SW_2,\ldots,SW_k$,其中SW_k为距离当前时刻最远的子窗口.记第 $i(1\leq i\leq k)$个子窗口的大小为$|SW_i|$,则滑动时间窗口 *SW* 的大小为$\sum|SW_i|$.对于滑动窗口内任意的一个数据项 *item*,如果 *item* 在子窗口SW_i内的支持数为$freq(item,SW_i)$,那么,*item* 在滑动时间窗口 *SW* 内真实的支持数$freq(item,SW)=\sum_{i=1}^{k}freq(item,SW_i)$.如果$\exists m,1\leq m\leq k$,并且有下式成立:

$$\forall i,m\leq i\leq k,\sum_{i=m}^{k}freq(item,SW_i)<\varepsilon\sum_{i=m}^{k}|SW_i|.$$

那么,当删除数据项 *item* 在子窗口$SW_m,SW_{m+1},\ldots,SW_k$内的信息后,得到的 *item* 在滑动窗口内的支持数为近似的支持数$\widehat{freq}(item,SW)$,且$\widehat{freq}(item,SW)=\sum_{i=1}^{m-1}freq(item,SW_i)$.根据文献[4]中的 Claim 3.5.1 可知,*item* 在滑动时间窗口 *SW* 内近似支持数$\widehat{freq}(item,SW)$与真实支持数$freq(item,SW)$的误差将不大于ε. □

如果选取$(\theta-\varepsilon)$为最小支持度门限从滑动窗口树上输出数据流滑动时间窗口内的频繁模式,那么删除滑动窗口树上的不频繁模式将不会造成模式挖掘的漏报.综上所述,我们可以采用如下文中算法 2 对滑动窗口树进行剪枝.剪枝时,我们采用自顶向下的方法遍历数据项头表中的数据项.对于其中任意一个数据项,如果它是不频繁的,则删除模式树上所有与其同名的节点;如果它是过期的,则删除模式树上所有与其同名的节点以及它们的子孙节点;否则遍历模式树上与其同名的节点,并删除那些过期的节点以及它们的孩子节点.显然,当删除某个过期节点时,其所有的孩子节点可以直接删除,而无需对其进行重新检查.因为,滑动窗口树的剪枝无需遍历树上的所有节点,故提高了模式树剪枝的速度.

算法 2. 滑动窗口树剪枝 PruningTree(*Root*,*N*,*T*).

输入参数: Root,滑动窗口树的根节点;

N,滑动时间窗口的大小;

Tc,剪枝开始时数据流上的当前事务.

输出结果:剪枝后的滑动窗口树.

```
1   for (自顶向下获取数据项表中的每一个数据项 item){
2       if (item 为过期数据项){
3           删除滑动窗口树上所有与 item 同名的节点以及它们的子孙节点;
4       } else (item 为不频繁数据项){
5           删除滑动窗口树上所有与 item 同名的节点;
6       } else {
7           for (滑动窗口树上任意一个与 item 同名的节点 node){
8               if (node 记录着过期模式){
9                   删除节点 node 及其子孙节点;
10      }}}
11      if (滑动窗口树上所有与 item 同名的节点均被删除)
12          将 item 从数据项头表中删除;
13  }}
```

3.4 模式输出

根据数据流应用的特点,当用户提交数据流滑动时间窗口内频繁模式查询请求后,系统要求能够及时地响应用户请求并在有限的时间内反馈查询的结果.文中采用基于 FP-growth[14]的方法来从滑动窗口树上输出数据流滑动时间窗口内的频繁模式.由于滑动窗口树 SW-tree 与 FP-tree 在存储结构上存在着差异,并且它们各自所维护的模式信息的对象也存在着差异.因此,本文对 FP-growth 算法进行了改进以适应从滑动窗口树上输出频繁模式集.

由于滑动窗口树上不仅维护着频繁的模式信息,还包含着临界频繁的模式信息.因此,当模式输出时,需要首先判断滑动窗口树上节点数据项是否频繁.只有当节点为频繁数据项时,才从滑动窗口树上输出所有包括该数据项的频繁模式.从滑动窗口树上输出频繁模式的详细步骤说明如下:首先,从数据项头表的头部开始依次读取数据项 e;第二,如果数据项 $freq_d(e,SW)\geq(\theta-\varepsilon)N$,则将 e 添加到一个称为频繁数据项集(frequent items set,简称 FIS)的集合中;第三,从滑动窗口树上构建 e 的条件模式基,并且该模式基仅包含那些在 FIS 中的数据项,否则执行步骤 5;第四,从数据项 e 的条件模式基中输出所有包含 e 的频繁模式;第五,从数据项头表中读取数据项 e 的下一个元素,然后重复上面第二步和第三步,直到数据项头表中的所有数据项均被处理.

4 正确性分析

为了评价算法的正确性,文中引入数据流频繁模式挖掘正确性的评价标准:覆盖率[7]($Coverage=\frac{P_A\cap P_B}{P_B}$,其中 P_A 为运用算法挖掘数据流得到的频繁模式集,P_B 为数据流上真实的频繁模式集).为了保证模式挖掘无漏报,实现模式挖掘覆盖率到达 100%,则要求在模式挖掘时能够输出所有真实的频繁模式.

下面,我们通过讨论数据流滑动时间窗口内频繁模式挖掘的一种极端情况来分析算法 MSW 的正确性.如图 2 所示,滑动时间窗口 SW 的大小为 N,当前事务为 T_n,并且 SW 中只有阴影部分的 θN 个事务包含模式 P.如果记模式 P 在滑动时间窗口 SW 内的衰减支持数为 $freq_d(P,SW)$,那么有 $freq_d(P,SW)=f^{N-\theta N}+\ldots+f^{L-1}+\ldots+f^{N-1}$,其中,$T_i$ 为包含模式 P 的一个事务,L 为事务 T_i 与当前事务 T_n 之间的距离,f^{L-1} 为事务 T_i 对模式 P 的衰减支持数 $freq_d(P,SW)$ 的影响,f 为衰减因子.由于包含模式 P 的事务为最早进入滑动时间窗口的 θN 个事务,它们距离数据流上的当前事务 T_n 最远.因此,在时间衰减模型的作用下,模式 P 的支持数受衰减影响的程度最大.也就是说,在图示情况下,模式 P 的衰减支持数 $freq_d(P,SW)$ 将减小至其最小值.

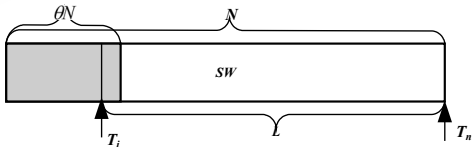


Fig.2 The model of mining a sliding window of data stream

图 2 挖掘数据流滑动时间窗口内频繁模式的模型

而实际上,模式 P 在滑动时间窗口 SW 内的真实支持数 $freq(P,SW)=\theta N$,故 P 是频繁的.因此,为了保证模式

挖掘的正确性,必须保证 $freq_d(P,SW)\geq(\theta-\varepsilon)N$,即必有 $f^{N-\theta N}+...+f^{L-1}+...+f^{N-1}\geq(\theta-\varepsilon)N$ 成立.由此,我们可以得到衰减因子 f 的取值区间:

$$1\geq f\geq \sqrt[(2N-\theta N-1)]{[(\theta-\varepsilon)/\theta]^2}\ .$$

因此,当挖掘数据流滑动时间窗口内频繁模式时,只要保证衰减因子 f 的值满足上式取值区间条件,则必能保证衰减支持数最小的频繁模式能够输出,亦即保证模式挖掘的覆盖率为 100%.

为了保证模式挖掘结果的覆盖率,在挖掘数据流上的频繁模式时,通常以$(\theta-\varepsilon)$为最小的支持度门限输出模式,这将导致一部分临界频繁模式被误当作频繁模式而被输出.在时间衰减模型下,如果衰减因子 f 的值取上限为 1(即不衰减模式的支持数),那么所有支持数大于或者等于$(\theta-\varepsilon)N$ 的临界频繁模式都将被当作频繁模式输出.然而,当衰减因子 f <1 时,受时间衰减模型的影响,将有一部分在 f=1 时被当作频繁模式输出的临界频繁模式而不能够输出,从而减少了模式挖掘的误报.在实际的数据流频繁模式挖掘中,为了减少误报,总是选择衰减因子 f 取其最小值.

根据上面 f 的取值区间,衰减因子 f 的最小值仅与最小支持度门限θ、最大许可误差ε以及滑动时间窗口的大小 N 有关.当用户选定θ与ε后,f的最小值与 N 的值成正比.因此,当滑动时间窗口的大小发生变化后,只需为新的滑动时间窗口重新选取一个合适的衰减因子即能保证模式挖掘的正确性.

5 实验仿真

实验在 CPU 为 PIV1.8GHZ、内存为 768MB、操作系统为 WIN2003 的 PC 机上进行,所有的实验程序均采用 Visual C++实现.实验中的模拟数据由 IBM 模拟数据产生器(http://www.almaden.ibm.com)产生,除了数据项的个数设置为 10K 以外,其他所有参数均使用默认值.实验中,用户许可误差θ设置为 0.1%,最大许可误差ε固定为 $0.1\times\theta$.实验结果中,算法所需要的存储空间为算法维护数据流上频繁模式信息所需要的存储空间,而算法的平均处理时间包括该模式树的维护时间、模式树剪枝时间以及从模式树上输出模式所需要的时间.如图 3 所示.

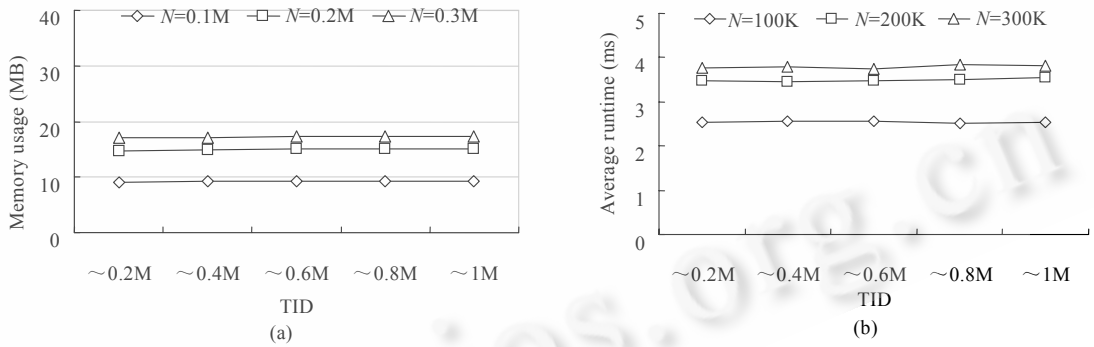


Fig.3 The performance study of MSW on T10I4D100k

图 3 算法 MSW 在数据流 T10I4D1000K 上运行的性能学习

实验通过分析算法 MSW 在数据流 T10I4D1000K 上运行所需要的最大内存空间与平均处理时间来学习算法的基本性能.实验中,数据流先被分割成大小为 0.2M 的数据分段.在算法运行过程中,每隔 0.1M 个事务执行次滑动窗口树剪枝操作.图 3(a)所示为算法在数据流各个数据分段上运行时所需最大内存空间.由于周期性的滑动窗口树剪枝删除了模式树上过期与不频繁的模式分支,当滑动时间窗口大小固定时,滑动窗口树上维护的事务数据集的大小不变.尽管新的流数据在不断产生,但滑动窗口树的大小基本不变.当滑动时间窗口增大时,滑动窗口树需要维护更多事务的模式信息,故模式树的大小有所增加.然而,当滑动窗口树所维护的事务数据集增大时,更多的事务可能复用相同的模式分支.因此,尽管滑动时间窗口的大小成倍增加,但滑动窗口树的增幅却很小.此外,由于 MSW 算法的时间复杂度与滑动窗口树的空间复杂度有着密切的关系.如图 3(b)所示,当滑动时间窗口的大小固定时,由于滑动窗口树在数据流各个数据分段上的空间复杂度也基本不变,因此 MSW 算法在

各个数据分段上的平均处理时间变化很小.当滑动时间窗口的大小增加时,由于滑动窗口树的空间复杂度增加,因此,算法的平均处理时间也相应增加.同样,算法的时间复杂度的增幅比滑动时间窗口大小的增幅要小得多.

第二,实验通过分析算法 MSW 挖掘数据流 T10I4D1000K 上频繁模式的覆盖率来学习算法的正确性.实验数据流首先被分割成 10 个大小为 0.1M 的数据分段.当算法运行在数据流的前 4 个数据分段上时,滑动时间窗口的大小设定为 0.1M.在第 4 个数据分段结束处,将滑动时间窗口的大小分别上调为 0.2M,0.3M 与 0.4M,并按照调整后的窗口大小继续进行模式挖掘.实验分别统计算法在第 2~第 8 个数据分段模式挖掘的覆盖率(coverage).如图 4 所示,在第 2~第 4 个数据分段,模式挖掘的覆盖率一直为 100%.在第 4 个数据分段结束处,当滑动时间窗口突然增大时,由于在滑动窗口树剪枝操作中删除了数据流滑动时间窗口内除了最近 0.1M 个事务之外的所有历史事务的模式信息,使一部分事务模式信息丢失,从而导致模式挖掘的漏报与覆盖率降低.并且滑动窗口的增幅越大,丢失模式信息的历史事务就越多,故覆盖率的降幅也就越大.但是,当滑动时间窗口的大小改变并重新选定衰减因子的数值后,随着新的流数据的到达,模式挖掘结果的覆盖率将逐渐增加,直到恢复为 100%.同样地,我们也进行了将滑动时间窗口大小逐步减小的实验,即在挖掘数据流频繁模式的前 4 个数据分段里,滑动时间窗口的大小设定为 0.4M,而第 4 个分段结束处,将滑动时间窗口的大小分别下调为 0.3M,0.2M 与 0.1M.实验结果表明,模式挖掘的覆盖率一直保持为 100%.这说明,当滑动时间窗口减小时,不会丢失事务的模式信息,故不会造成模式挖掘的误差.

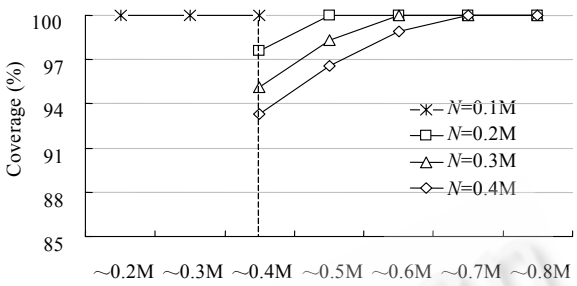


Fig.4 Correctness study of MSW on T10I4D100K

图 4 算法 MSW 在数据流 T10I4D1000K 上运行的正确性学习

第三,为了学习算法 MSW 处理长事务、长模式数据流的能力,实验分别考察 MSW 算法在数据集 T15I6D1000K,T15I10D1000K,T20I10D1000K,T20I15D1000K 以及 T25I20D1000K 运行的性能.同样,每个实验数据流都被分割成大小为 0.2M 个事务的数据分段,实验分别统计算法 MSW 在各个数据流各个数据分段上运行时所需要的最大内存空间与平均处理时间.如图 5 所示,当数据流的平均事务长度 T 增加时,由于滑动窗口树维护单个事务所需要的节点数增加,因此树的空间复杂度将有所增加.但是数据流的平均模式长度 I 对滑动窗口树的大小没有任何影响.另外,数据流平均事务长度与平均模式长度增加时,一方面,滑动窗口树空间复杂度的增加将导致维护滑动窗口树的时间代价增大,另外,由于数据流平均模式长度的增加将增加模式输出的时间复杂度,因此算法运行的平均处理时间有所增加.但实验结果显示,算法时间与空间复杂度增加的幅度都不大.

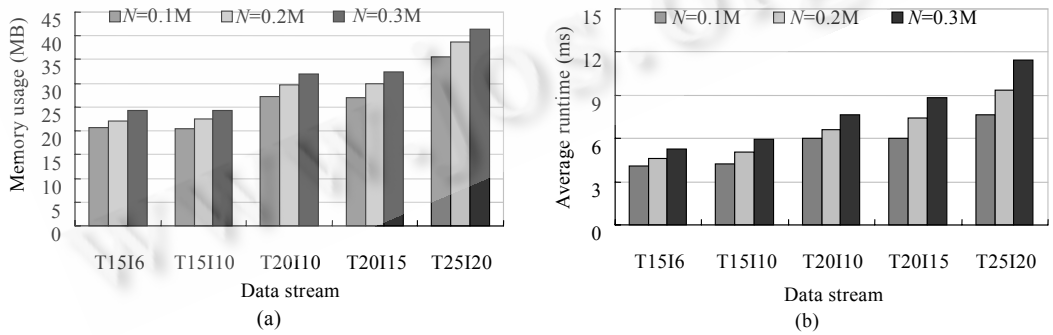


Fig.5 Performance study of MSW on different data stream

图 5 算法 MSW 挖掘不同数据流的性能学习

第四,实验通过将算法 MSW 与 FP-stream[4],estDec[5]及 DStree[8]进行对比来评价算法的性能.对比实验所使用的数据流为 T10I4D1000K,数据流首先被分割成大小为 0.2M 的数据分段,实验分别统计 4 种算法在数据流各

个数据分段上运行所需要的最大内存空间与平均处理时间.如图 6(a)所示,当开始进行数据流频繁模式挖掘时,estDec 方法所需要的内存空间要远远高于其他几种方法,约为 MSW 方法的 4 倍,DStree 方法与 FP-stream 方法相差不大,MSW 方法最小.随着已处理的流数据的增多,DStree 方法所需要的内存空间呈线性增加的趋势,FP-stream 方法的空间复杂度也在缓慢增加,而其他两种方法基本保持不变.这是因为,在这 4 种方法中,estDec 方法的模式树是一种松散的存储结构,需要使用更多的空间来存储数据流的模式信息.但是,estDec 方法周期性地删除了模式树上的不频繁模式与过期模式,使模式树仅仅维护数据流上最近的重要模式信息,因此,算法的空间复杂度在数据流各个数据分段上变化不大.而 DStree 树一直保留着历史事务的模式信息,尽管在模式挖掘开始时,DStree 方法的空间复杂度并不大,但是随着已处理流数据的增多,DStree 树所需的存储空间将迅速增加,当数据的大小达到 1M 时,DStree 所需要的内存空间约为 MSW 方法的 5.8 倍.尽管 FP-stream 方法仅仅维护数据流上的频繁模式,但是 FP-stream 方法的模式树存储结构松散.并且在处理数据流上的各个数据分段内的流数据时,还需要为每一个数据分段构建一个 FP-tree.另外,随着已处理流数据的增多,历史时间窗口内的频繁模式将一直保存在 FP-stream 的模式树上,因此,FP-stream 方法的最大存储空间随流数据的增多而增加.但是,相对于 DStree 方法,FP-stream 方法的空间复杂度增幅要小得多.当数据的大小达到 1M 时,FP-stream 方法所需要的内存空间约为 MSW 方法的 1.9 倍.

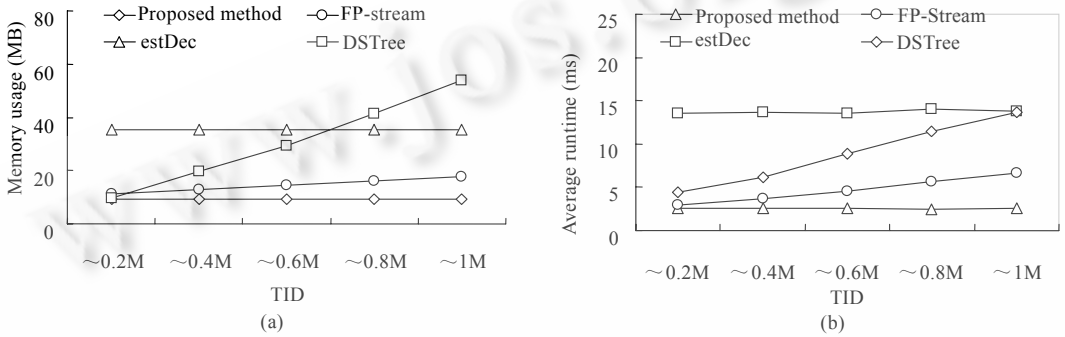


Fig.6 Performance comparison of the four methods

图 6 4 种方法的性能比较分析

此外,实验还比较了 4 种算法在数据流各个数据分段上运行的平均处理时间.如图 6(b)所示,当开始挖掘数据流频繁模式时,estDec 方法的平均处理时间远远高于其他 3 种方法,约为 MSW 方法的 5 倍,DStree 方法次之,FP-stream 方法与 MSW 方法相差不大.当已处理的流数据增多时,DStree 与 FP-stream 方法的平均处理时间都逐渐增大,但 DStree 方法的增速大于 FP-stream 方法.比较而言,MSW 方法与 estDec 方法的平均处理时间相对稳定.这是因为 estDec 方法采用 Apriori-Like 的方法处理数据流上的每一个事务,首先产生事务所包含的全部子模式,然后通过搜索模式树分别测试它们的频繁性,并将它们中的重要模式信息更新到模式树上,因此,算法 estDec 的平均处理时间相对较长.但是,由于 estDec 方法的模式树在数据流各个数据分段上的空间复杂度变化不大,因此,该方法在数据流各个分段上的平均处理时间也变化不大.而 DStree 树的存储空间随着已处理流数据的增多而迅速增加,因此维护 DStree 以及从 DStree 上输出模式的时间代价也将大为增加,当数据流的大小达到 1M 时,该方法的平均处理时间为 MSW 方法的 5.3 倍.对于 FP-stream 方法,除了因为模式树空间复杂度增加导致算法的平均处理时间增加之外,随着已处理数据分段的增多,FP-stream 方法还需要多次移动与合并模式树节点上的倾斜时间窗口,这也增加了算法的平均处理时间.

第五,实验通过比较 4 种算法处理长事务长模式数据流的能力来学习分析算法的可扩展性.实验数据流分别为 T15I6D200K,T15I10D200K,T20I10D200K,T20I15D200K 以及 T25I20D200K.为了避免各种算法的平均处理时间受数据流大小的影响,实验数据流的大小固定为 200K.实验分析比较 4 种算法的平均处理时间.如图 7(a)所示,当平均事务长度 T 与平均模式长度 I 增加时,4 种方法的平均处理时间均有所增加,但是,estDec 方法的增幅最大,达到 4 倍;其次为 FP-stream 方法与 DStree 方法,分别为 3.5 倍与 2.5 倍;增幅最小的为 MSW 方法,仅为

1.8 倍.这因为当事务的平均长度增大时,estDec 方法需要产生并测试更多的中间候选模式.而当事务的平均模式长度增加时,将有更多的模式信息需要更新到模式树上.因此,当平均事务长度特别是平均模式长度增加时,算法的平均处理时间将大为增加.FP-stream 方法不仅需要维护数据流上的全局模式树,还需要为数据流每个数据分段内的事务数据构建一个局部模式树.当数据流平均模式长度增加时,数据流上频繁模式的数量将大为增加,这不仅增加了数据流全局模式树的维护时间,而且增加了各个数据分段内频繁模式树的构建时间.而比较而言,DStree 方法与 MSW 方法受数据流平均事务长度及平均模式长度的影响较小.

第六,实验还对比分析了 4 种方法挖掘不同大小数据流的能力.实验数据流为 T10I4D1000K,实验使用 4 种方法分别挖掘数据流上大小从 0.1M~1M 个事务数据窗口内的频繁模式,并分析比较了 4 种方法的平均处理时间.如图 7(b)所示,当数据窗口比较小时,estDec 方法的平均处理时间最大,约为 MSW 方法的 5 倍;而其他 3 种方法相差不大,约为 3ms~4ms.但是随着数据窗口的增大,4 种方法的平均处理时间均在增加.其中,DStree 方法的增速最快,FP-stream 方法次之,estDec 方法与 MSW 方法增速相对缓慢.当数据窗口大于 0.5M 时,DStree 方法的平均处理时间超过 estDesc 方法.由于 estDec 方法需要为每一个事务产生大量子模式并测试它们的频繁性,因此当挖掘相对小的流数据集合时,estDec 方法的效率比其他 3 种方法中的任何一种都低.当数据窗口增大时,各种方法的模式树都需要维护更多事务的模式信息,因此它们的平均处理时间均有所增加.但是,在这 4 种方法中,由于 DStree 方法一直维护着数据流上所有事务的全部模式信息,当数据流的大小增加时,DStree 方法的平均处理时间迅速增大并超过 estDec 方法.而其他 3 种方法则周期性地删除模式树上的不频繁模式及过期模式,降低了模式树的维护代价.因此,算法的平均处理时间增幅相对缓慢.而在这 3 种方法中,当数据窗口增大时,FP-stream 方法需要多次移动并合并模式树上节点的倾斜时间窗口,故 FP-stream 方法平均处理时间的增幅相对要大得多.比较而言,MSW 方法总体上的时间效率较高,并且算法的平均处理时间受数据流大小的影响较小.因此,在挖掘数据流上任意大小时间窗口内频繁模式时,MSW 算法优于其他 3 种方法.

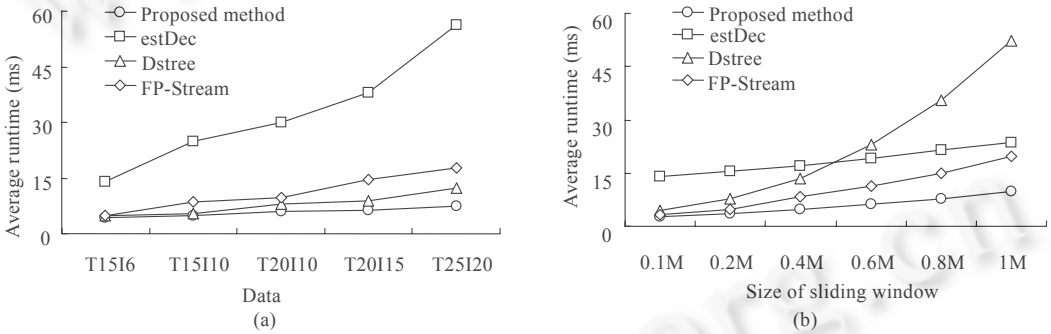


Fig.7 Scalability comparison of the four methods

图 7 4 种方法的可扩展能力比较

6 总 结

由于数据流的流动性与连续性,数据流所蕴含的知识会随着时间的推移而发生变化.因此,在挖掘在线数据流频繁模式时,将新产生事务的模式信息与历史事务的模式信息区分开来,发现数据流上最近的变化趋势显得尤为重要.本文提出了一种挖掘数据流上任意大小滑动时间窗口内频繁模式的方法 MSW.当数据流流过时,该方法使用滑动窗口树 SW-tree 仅在单遍扫描流数据的条件下捕获数据流上最新的模式信息;同时,通过周期性的剪枝操作极大地减少了滑动窗口树上过期与不频繁模式,降低了滑动窗口树的维护代价,提高了算法的可扩展能力.此外,该方法还通过逐渐降低历史事务模式支持数权重的方法来区分新产生事务的模式与历史事务的模式.最后,大量仿真实验的结果表明,MSW 算法具有较高的效率与优良的可扩展性能,同时也优于其他同类算法.

References:

[1] Gaber MM, Zaslavsky A, Krishnaswamy S. Mining data streams: A review. ACM SIGMOD Record, 2005,34(2):18−26.

[2] Jiang N, Gruenwald L. Research issues in data stream association rule mining. ACM SIGMOD Record, 2006,35(1):14−19.

[3] Garofalakis MN, Gehrke J. Querying and mining data streams: You only get one look a tutorial. In: Franklin MJ, Moon B, Ailamaki A, eds. Proc. of the 2002 ACM SIGMOD Int'l Conf. on Management of Data. Madison: ACM Press, 2002. 635−635.

[4] Giannella C, Han J, Pei J, Yan X, Yu PS. Mining frequent patterns in data streams at multiple time granularities. In: Data Mining: Next Generation Challenges and Future Directions. 2004. 191−212.

[5] Chang JH, Lee WS. Finding recent frequent itemsets adaptively over online data streams. In: Lise G, Ted ES, Pedro D, Christos F, eds. Proc. of the 9th ACM SIGKDD Int'l Conf. on Knowledge Discovery and Data Mining. Washington: ACM Press, 2003. 487−492.

[6] Jiang N, Gruenwald L. CFI-Stream: Mining closed frequent itemsets in data streams. In: Roberto B, Kristin PB, Gautam D, Dimitrios G, Johannes G, eds. Proc. of the 12th ACM SIGKDD Int'l Conf. on Knowledge Discovery and Data Mining. Philadelphia: ACM Press, 2006. 592−597.

[7] Yu JX, Chong Z, Lu H, Zhang Z, Zhou A. A false negative approach to mining frequent itemsets from high speed transactional data streams. Information Sciences, 2006,176(4):1986−2015.

[8] Leung CKS, Khan QI. DStree: A tree structure for the mining of frequent sets from data streams. In: Clifton CW, Zhong N, Liu JM, Wah BW, Wu XD, eds. Proc. of the 6th Int'l Conf. on Data Mining. Hong Kong: IEEE Press, 2006. 928−932.

[9] Wong RCW, Fu AWC. Mining top-*k* frequent itemsets from data streams. Data Mining and Knowledge Discovery, 2006,13(2): 193−217.

[10] Papadimitriou A, Yu PS. Optimal multi-scale patterns in time series streams. In: Roberto B, Kristin PB, Gautam D, Dimitrios G, Johannes G, eds. Proc. of the 2006 ACM SIGMOD Int'l Conf. of Management of Data. Chicago: ACM Press, 2006. 647−658.

[11] Arasu A, Manku GS. Approximate counts and quantiles over sliding windows. In: Weikum G, Koenig AC, Dessloch S, eds. Proc. of the 23rd ACM SIGMOD-SIGACT-SIGART Symp. on Principles of Database Systems. Paris: ACM Press, 2004. 286−296.

[12] Ho CC, Li HF, Kuo FF, Lee SY. Incremental mining of sequential patterns over a stream sliding window. In: Tsumto S, Clifton CW, Zhong N, Wu XD, Liu JM, Wah BW, Cheung YM, eds. Proc. of the 6th Int'l Conf. on Data Mining Workshops. Hong Kong: IEEE Press, 2006. 677−681.

[13] Lee L, Ting H. A simpler and more efficient deterministic scheme for finding frequent items over sliding windows. In: Roberto B, Kristin PB, Gautam D, Dimitrios G, Johannes G, eds. Proc. of the 25th ACM SIGMOD-SIGACT-SIGART Symp. on Principles of Database Systems. Chicago: ACM Press, 2006. 290−297.

[14] Han J, Pei J, Yin Y. Mining frequent patterns without candidate generation. In: Chen W, Naughton JF, Bernstein PA, eds. Proc. of the 2000 ACM SIGMOD Int'l Conf. on Management of Data. Dallas: ACM Press, 2000. 1−12.

李国徽(1973－),男,湖南衡阳人,博士,教授,博士生导师,主要研究领域为现代数据工程,实时数据库系统.

陈辉(1976－),男,博士生,主要研究领域为实时数据库系统,数据挖掘.