CN 43-1258/TP
ISSN 1007-130X

计算机工程与科学
Computer Engineering & Science

第 36 卷第 5 期 2014 年 5 月
Vol. 36, No. 5, May. 2014

文章编号:1007-130X(2014)05-0863-08

一种改进的数据流最大频繁项集挖掘算法*

胡　健,吴毛毛
(江西理工大学信息工程学院,江西 赣州 341000)

摘　要:提出了一种基于 DSM-MFI 算法的改进算法 DSMMFI-DS 算法,它首先将事务数据按一定的全序关系存入 DSFI-list 列表中;然后按排序后的顺序存储到类似概要数据结构的树中;接着删除树中和 DSFI-list 列表中的非频繁项,同时删除窗口衰退支持数大的事务项;最后采用自顶向下和自底向上的双向搜索策略来挖掘数据流的最大频繁项集。通过用例分析和实验表明,该算法比 DSM-MFI 算法具有更好的执行效率。

关键词:数据挖掘;数据流;界标窗口;最大频繁项集;窗口衰减支持数
中图分类号:TP274.2　　**文献标志码**:A
doi:10.3969/j.issn.1007-130X.2014.05.030

An improved algorithm for mining maximal frequent itemsets over data streams

HU Jian, WU Mao-mao
(Institute of Information Engineering, Jiangxi University of Science and Technology, Ganzhou 341000, China)

Abstract: Based on the algorithm of DSM-MFI, an improved algorithm, named DSMMFI-DS (Dictionary Sequence Mining Maximal Frequent Itemsets over Data Streams), is proposed. Firstly, it stores transaction data into DSFI-list in alphabetical order. Secondly, the data are stored sequentially into the tree similar to the summary data structure. Thirdly, non-frequent items in the tree and DSFI-list are removed, and the transaction items with the maximum count of window attenuation supports are deleted. Finally, the strategy (top-down and bottom-up two-way search) is used to mine maximal frequent itemsets over data streams, and case analysis and experiments prove that the algorithm DSMMFI-DS has better performance than the algorithm DSM-MFI.

Key words: data mining; data stream; landmark windows; maximal frequent itemsets; window attenuation support count

1　引言

频繁模式的挖掘是关联挖掘的核心和基础[1],是影响挖掘算法效率的一个决定性的因素,它是产生关联规则的基础[2]。因此,在频繁模式[3~5]挖掘方面取得的任何进展都将对关联挖掘以至于其它数据挖掘任务的效率产生重要的影响。

由于最大频繁项集[6~9]中隐含着全部的频繁项集,因此可以将计算频繁项集的问题转化为计算最大频繁项集。在某些应用中,只需要最大频繁项集而并不需要所有的频繁项集,这样,研究直接计算最大频繁项集的算法显示出重要意义。

最近,数据库和数据挖掘继续集中到一个新的数据模型中,数据到达是以数据流的形式。在很多应用中,实时产生了大量的数据流,比如从一个传感器网络到另一个传感器数据传输产生的数据;各个连锁店事务数据的流入;Web 记录和在 Web 上的点击流;在网络监控和交通管理的测量评估数据等[10]。本文就是基于这个背景提出了一个有效挖

* **收稿日期**:2012-12-03;**修回日期**:2013-04-03
通信地址:341000 江西省赣州市客家大道 156 号
Address: 156 Kejia Avenue, Ganzhou 341000, Jiangxi, P. R. China

掘数据流中最大频繁项集的算法。

2 相关工作

Giannella C 等人[11]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。马达等人[12]在一种基于压缩 FP-树的最大频繁项集挖掘算法中提出一种剪枝策略并结合 FP-树的结构,并根据 Patricia-树的原理设计出 PFP-树,对数据库进一步压缩降低内存。敖富江等人[13]提出一种新的基于文法顺序的 FP-Tree 的最大频繁项单遍挖掘算法 FPMFI-DS,该算法采用混合搜索空间顺序策略,利用子集等级剪枝策略来压缩搜索空间的大小;同时提出了能够在线更新挖掘滑动窗口内数据流的最大频繁项集 FPMFI-DS+算法。在界标窗口模型中,文献[14]提出了一种基于界标窗口模型的最大频繁项集的挖掘方法。S-FP-MFI 算法[15]是一种基于 FP 树的改进的数据流最大频繁项集挖掘算法,它是根据条件模式基的项目数对条件模式基进行动态排序,用来减少函数调用的次数。基于事务矩阵挖掘最大频繁项集的方法 AFMI[16]利用了迭代精简事务矩阵的方法来发现事务数据中的最大频繁项集,并提出了滑动窗口中的数据流最大频繁项集 AFMI+算法。频繁模式树的约束最大频繁项集快速挖掘算法[17]能够删除不满足约束条件的项集,由于不需要生成候选项目集,所以效率很高。Li Hua-fu 等人[18]首先提出的 DSM-FI 算法通过将事务数据存储到概要数据结构中,建立 IsFI-forest,将每个项的投影子集存储到 DHT 表中,采用自顶向下的搜索策略来发现频繁项集。但是,该算法在删除一个非频繁项集时需要遍历整个项集前缀频繁项目森林(IsFI-forest),当森林结构复杂时,需要删除很多非频繁项集,需要很多次遍历森林,消耗的时间很多。对于较长的非频繁项集也要建立 DHT 表,同时要将各个子集投影到 IsFI-forest 中去,这样不仅使存储空间加大,而且还造成删除耗时。

本文提出的 DSMMFI-DS(Dictionary Sequence Mining Maximal Frequent Itemsets over Data Streams)算法是基于 DSM-MF 算法[1]提出来的,DSMMFI-DS 算法先对流入的数据流按一定顺序排序(本文是按一种全序关系)存入 DSFI-list(Dictionary Sequence Frequent Item List)列表中,并且按这个顺序存储到 DSSEFI-tree 树中;接着删除 DSFI-list 列表中的非频繁项对应在 DSSEFI-tree 树中的项;然后删除 DSFI-list 列表中的非频繁项;最后在当前的概要数据结构中利用自顶向下和自底向上的双向搜索策略发现最大频繁项集。

3 问题的定义及相关知识

3.1 问题定义

从大的数据库中挖掘最大频繁项集是关联挖掘的一个重要问题。这个问题是 Bayardo 首先提出来的。问题的定义如下[1]:设 $\Psi=\{i_1,i_2,\cdots,i_n\}$ 是一系列的数据,i_n 称作项目。设数据库 DB 是一系列的交易事务,DB 的大小用 $|DB|$ 表示。一个事务 T 有 m 个项目,可以表示成 $T=\{i_1,i_2,\cdots,i_m\}$,k-项集就是包含有 k 个项目的集合,表示为 $\{x_1,x_2,\cdots,x_k\}$。一个项目集 X 的支持度就是指在这个数据库中所有包含 X 项的总项目数占这个数据库项目总数的百分比,X 项目的支持度用 $sup(X)$ 表示。如果 $sup(X)>minsup$,那么 X 就被称作频繁项集,$minsup$ 是用户自定义的最小支持度值,它的范围是[0,1]。所有频繁项集集合用 FI(Frequent Item)表示。如果一个频繁项集不是任何一个频繁项集的子集,那么就把它叫做最大频繁项集,一系列最大频繁项集的集合用 MFI(Maximal Frequent Itemsets)表示。本文目的就是找到所有支持度比用户给定的最小支持度大的最大频繁项集。

定义 1[19] 设数据流 $DS=\{w_1,w_2,\cdots,w_N\}$,其中 $w_i(\forall i=1,2,\cdots,N)$ 的 i 是每一个基本窗口的标识,N 是最后一个基本窗口的标识。它是一个连续的、无穷的基本窗口序列,每个基本窗口含有固定数目的事务,$\langle T_1,T_2,\cdots,T_k,\cdots\rangle$,$T_k=\{i_1,i_2,\cdots,i_m\}$,$T_k(K=1,2,\cdots)$ 称为事务。

定义 2 设用户给定的支持度 s 和允许偏差 $\varepsilon(0<\varepsilon<s)$,$|w|$ 表示基本窗口的长度,$SL=|w_1|+|w_2|+\cdots+|w_N|$,$N$ 为当前窗口标识,$x.SL=|w_j|+|w_{j+1}|+\cdots+|w_N|$,其中 w_j 为项目 x 在数据流中出现的第一个窗口。项目 x 被分为三种类型,如果 $sup(x)\geqslant s\cdot x.SL$,则称为频繁项集;如果 $s\cdot x.SL\geqslant sup(x)\geqslant\varepsilon\cdot x.SL$,则称潜在频繁项集;如果 $sup(x)<\varepsilon\cdot x.SL$,则称不频繁项集。

3.2 窗口衰退支持数

由于数据流的流动性与连续性,在一段界标窗

口内的事务数量可能很大甚至无限,数据流中蕴含的知识会随着窗口的推移而发生变化。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或者是最大频繁模式,而忽略那些历史事务的模式。

本文应用窗口衰减模型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窗口内的衰退比率为衰减因子 $cf(0<cf<1)$,记事务 T_i 到达时模式 P 的衰退支持数为 $cf(P, T_i)$,那么,当数据流上的第一个事务 T_1 到达时,$cf(P, T_1)=c$,当第二个到达时,有 $cf(P, T_2)=cf(P, T_1)\times cf+c$。依次类推,当第 i 个事务到达时,模式 P 的衰退支持数可以由下式得到[17]:

$$cf(P,T_i)=\begin{cases}c, i=1\\ cf(P,T_{i-1})\times cf+c, i\geqslant 2\end{cases}$$

$$c=\begin{cases}1, P\subseteq T_i\\ 0, \text{其它情况}\end{cases} \quad (1)$$

4　算法描述

DSMMFI-DS 算法先把每个窗口的事务数据存储到一个新的概要数据结构[19]—改进的前缀频繁项目树 *DSSEFI-tree* 中;然后根据 *DSFI-list* 列表中计算的每个项目的支持数,从 *DSSEFI-tree* 中删除不频繁项,接着删除 *DSFI-list* 列表中的非频繁项并将 *DSSEFI-tree* 树存储到 *DSSEFI-forest* 森林;最后根据用户需求或者需要遍历 *DSSEFI-forest* 森林挖掘相应项的最大频繁项集。

4.1　概要数据结构

为了适应滑动窗口内最大频繁项集的挖掘,本节设计了一种前缀概要项目树 *DSSEFI-tree*,与 DSM-MFI 中的 *SFI-tree* 相比,除了存储项目名字 *item_name*、窗口的标识 *window_count*、项目的支持数 *node_count*、指向 *DSFI-list* 表中的相同节点的指针 *node_link* 和指向 *DSSEFI-tree* 树中具有相同项目名字的节点 *item_brother* 外,还增加了一个参数就是窗口衰退支持数 *parameter*,该参数主要记录着每个项随着窗口的移动所代表的权重,当 *parameter* 很大时说明该项不是最近窗口的项,我们可以尽早将该项从树结构中删除掉。

同时,为概要项目树 *DSSEFI-tree* 设计了一个头项列表 *DSFI-list*,该列表与 *DSM-MFI* 算法中的 *FI-list* 相比,除了具有存储项目名字 *item_name*、窗口的标识 *window_count*、项目的支持数 *node_count*、指向 *DSSEFI-tree* 树中的相同节点的指针 *node_link* 和指向 *DSSEFI-tree* 树中具有第一个相同项目名字的节点 *item_brother* 外,也增加了窗口衰退支持数 *parameter*,由这个参数我们就可以很快知道树中存储的哪些项是已经过期的,可以很方便地将此项删除。

4.2　增量更新 *DSSEFI-tree* 树

由于数据流不断地流入,内存的存储空间有限,需要实时地对树进行更新,及时删除不频繁项集和过期的项,同时要及时返回用户查询的结果。由于树本身在存储时是按照头项列表中的全序关系进行存储的,不受项到来的先后顺序影响,在对项进行操作时也比较方便。再根据用户设定的最小支持度和窗口衰退支持数阈值,在每一个窗口到来之后就对树进行一次更新,这样可以很快地将树中的不频繁项集和已经过期的项集剪枝掉,在节省内存空间的同时提高了查询的效率。假设新的窗口事务数据 $TDS(i)=(a_1,a_2,\cdots,a_N)$ 到来时,算法 1 是对树 *DSSEFI-tree* 的更新。

算法 1　增量更新树 *DSSEFI-tree*

输入:事务数据 $TDS(i)=(a_1,a_2,\cdots,a_N)$,用户设定的最小支持度 s,用户设定的窗口衰退支持数阈值 p,用户允许的最大误差率 $\varepsilon(0,s)$;

输出:更新后的树 *DSSEFI-tree*。

(1) if *DSFI-list*$=\varnothing$, then {*window_count*$=1$;
(2)　for each a_j ,将 a_j 的 *item_name*、*window_count* 和 *node_link* 插入到 *DSFI-list* 中;
(3)　end for
(4)　为树 *DSSEFI-tree* 构造根节点 *root*;
(5)　for each a_j, 将 a_j 的 *item_name*、*window_count*、*node_count*、*node_link*、*item_brother* 和 *parameter* 插入到 *DSSEFI-tree* 中;}
(6)　end for
(7) end if
(8) else {
　　window_count$=i$;
(9)　for each a_j 按全序插入到 *DSFI-list* 中;
(10)　if 有相同 *item_name* 的项 *item* then *item*. *parameter*$=a_j$. *parameter*;
(11)　end if
(12)　for each *DSFI-list* 中的每个项 a_i 对应的 *parameter* 项进行更新: *parameter*$=cf(P,T_{i-1})\times cf+c$;
(13)　　if a_i . *parameter* $\geqslant p$,将 a_i 从树 *DSSEFI-*

tree 中删除,接着从 *DSFI-list* 中删除;

(14) end if

(15) if a_i . *node_count*$<s \cdot a_i$. *SL* or a_i . *node_count*$<\varepsilon \cdot a_i$. *SL*,将 a_i 从树 *DSSEFI-tree* 中删除,接着从 *DSFI-list* 中删除;

(16) end if

(17) end for

(18) end for

(19)}

(20)}end else

4.3 最大频繁项集的挖掘算法

更新完树 *DSSEFI-tree* 后,我们根据需求来对数据流进行操作,查找相应项的最大频繁项集。本文设计一个改进的双向搜索策略 tb-btMFI(top-bottom and bottom-top selection of Maximal Frequent Items)[20],双向搜索从 *DSSEFI-tree* 树中发现最大频繁项集,其中设计一个监控参数 *monitor*,让自顶向下搜索和自底向上搜索同步进行,可以及早发现不频繁项,提高挖掘效率。

算法描述:自顶向下搜索,设置 *monitor* 的值为 0,对每一个项 x,将 x 所在的最长路径中所有项合并形成最大频繁候选项集存储在 M_1 中(假设为 k 项集),对这个最长的项集进行支持度计算,如果它满足用户给定的最小支持度,则它是最大频繁项集,并将这个最大频繁项集加入到 *MFI* 中,令 *monitor* 为 1,转到自底向上搜索,如果 M_2 中某项是 *MFI* 的子集,则从 M_2 删除此项;如果不满足,则对 x 和其他项进行任意组合形成 $k-1$ 项集存储在 M_1 中,令 *monitor* 为 1,转到自底向上搜索,如果 M_2 中某项是 *MFI* 的子集则从 M_2 删除此项。依次进行下去,直到 M_1 为空为止。自底向上搜索,如果 *monitor* 的值为 1,从树的叶子节点开始对 x 项进行相邻两短项集组合,存储在 M_2 中,搜索到 L 层时仍然保留 $L-1$ 层中的项,然后对组合的项集进行支持度计算,对第 L 层上的某个项目集 X,若 X 是 *MFS* 中某频繁项目集的子集,则不对 X 进行支持度计算,令 *monitor* 的值为 0,转到自顶向下搜索;否则,对 X 进行支持度计算,若 X 是第 L 层中的频繁项目集,则从 M_2 中删除 X 在 $L-1$ 层中的所有项目子集,否则删除 X,令 *monitor* 的值为 0,继续自顶向下搜索;若第 $L-1$ 层中的某个频繁项目集 y 在第 L 层上的所有超集均为非频繁项目集,则将 y 加到 *MFS* 中并从 M_2 中删除 y,令 *monitor* 值为 0,继续自顶向下搜索。如果 *monitor* 的值为 1,则处理完第 L 层上的每一个项目集,如果 *monitor* 的值为 1,则生成 $L+1$ 层上的候选频繁项目集,依次类推,直到 M_2 为空。如算法 2 所示。

算法 2 tb-btMFI 算法

输入:更新后的 *DSSEFI-tree* 树,当前窗口标识 N,最小支持度 s,用户允许的最大误差率 ε;

输出:最大频繁项集 *MFI*。

(1)令 $MFI=\varnothing$, $M_1=\varnothing$, $M_2=\varnothing$, *monitor*=0;

(2)for($k=n$;$k\geqslant 1$;$k--$)

(3){

(4) M_1 ={路径中包含 X_i 项目的组成最大候选频繁项集 X};

(5) 计算各候选频繁项集的支持度;

(6) if 候选项集 $\in$ *MFI* then $M_1=M_1-X$, *monitor*=1,转到(13);

(7) else 计算候选项集 X 的支持度;

(8) if X. *node_count*$\geqslant s \cdot X$. *SL* or X. *node_count*$<\varepsilon \cdot X$. *SL* then

(9) $MFI=MFI\cup\{X\}$, *monitor*=1,转到(13);

(10) else $M_1=M_1-\{X\}$, *monitor*=1,转到(13);

(11)}

(12)if $M_1=\{\varnothing\}$ then 退出;

(13)for ($k=0$; $k<=L$; $k++$){

(14) M_2 ={包含 x_i 项的相邻两项组成的候选频繁项集 X}

(15) if $X\in MFI$ then

(16) $M_2=M_2-X$, *monitor*=0,转到(2);

(17) if 在第 $k-1$ 层的项 X 是第 k 层的子集 then

(18) $M_2=M_2-X$, *monitor*=1,转到(2);

(19) 计算各项目 X 的支持度;

(20) if X. *node_count*$\geqslant s \cdot X$. *SL* or X. *node_count*$<\varepsilon \cdot X$. *SL* then

(21) $MFI=MFI\cup\{X\}$, *monitor*=1,转到(2);

(22) else $M_2=M_2-X$, *monitor*=1,转到(2);

(23)}

(24)if $M_2=\{\varnothing\}$ then 退出;

(25)end for

5 算法的复杂性分析与用例分析

5.1 算法的复杂性分析

(1)时间复杂度。

设当前窗口的标识为 N,基本窗口的事务数为 T,每个事务的平均项目数为 I,频繁 1-项目的个数为 k。在 DSM-MFI 算法中需要将 $T\times N(C_I^1+C_I^2+\cdots+C_I^I)$ 个节点存储到树中,那么在删除一项不频繁项时需要遍历 $\sum_{x=1}^{|T/2|}C_{|T/2|+x}^{x}$ 个节点;在挖

掘最大频繁项集时,对于 *FI-list* 中的每个项 i 都对应一个 $i.OFI\text{-}list$,包含的项目数用 L_i 表示,用 $C1$ 表示不频繁项目数,$C2$ 表示遍历一遍树所需要的时间,最后的挖掘完成时 DSM-MFI 算法的时间复杂度为[19]:

$$(I^2-I)/2\times T\times N+C1\times\sum_{x=1}^{|T/2|}C_{|T/2|+x}^{x}+C2\times\sum_{x=1}^{k}(C_{L_i}^{1}+C_{L_i}^{2}+\cdots+C_{L_i}^{L_x}) \tag{2}$$

对于 MMFI-DS 算法:(1)在 *SEFI-tree* 构造和维护算法中,算法需把 $T\times N$ 个项目插入到 *SEFI-tree* 中,删除一个不频繁项目,算法需删除此不频繁项在 *EIS-tree* 中节点的个数,设其数目为 $C3$。(2)在发现最大频繁项集的算法中,MMFI-DS 算法的时间复杂度为[19]:

$$T\times N+C1\times C3+C2\times\sum_{x=1}^{k}(C_{R_i}^{1}+C_{R_i}^{2}+\cdots+C_{R_i}^{L_x}) \tag{3}$$

其中,$C_{R_i}^{1},C_{R_i}^{2},\cdots,C_{R_i}^{L_x}$ 分别是对应的 $C_{L_i}^{1},C_{L_i}^{2},\cdots,C_{L_i}^{L_x}$ 删除不频繁项剩下的频繁项数目,显然 $\sum_{x=1}^{k}(C_{R_i}^{1}+C_{R_i}^{2}+\cdots+C_{R_i}^{L_x})\leqslant\sum_{x=1}^{k}(C_{L_i}^{1}+C_{L_i}^{2}+\cdots+C_{L_i}^{L_x})$。

对于 DSMMFI-DS 算法:(1)在 *DSSEFI-tree* 树的增量更新中,算法也要把 $T\times N$ 个项目插入到 *DSSEFI-tree* 中,删除一个不频繁项目,算法需删除此不频繁项在 *DSSEFI-tree* 中节点的个数,设其数目为 $C4$。(2)在发现最大频繁项集的算法中,DSMMFI-DS 算法的时间复杂度为:

$$T\times N+C1\times C4+C2\times\sum_{x=1}^{k}(C_{R_i}^{1}+C_{R_i}^{2}+\cdots+C_{R_i}^{L_x}) \tag{3}$$

从上面的三个时间复杂度来看,显然 $C3<\sum_{x=1}^{|T/2|}C_{|T/2|+x}^{x}$,$\sum_{x=1}^{k}(C_{R_i}^{1}+C_{R_i}^{2}+\cdots+C_{R_i}^{L_x})\leqslant\sum_{x=1}^{k}(C_{L_i}^{1}+C_{L_i}^{2}+\cdots+C_{L_i}^{L_x})$,因此式(3)≤式(2),即 MMFI-DS 算法的时间复杂度少于 DSM-MFI 算法的时间复杂度;而 DSMMFI-DS 算法增加了窗口衰退支持数,对于过期的或者用户不感兴趣的项将从树和列表中删除,$C4\leqslant C3$,那么式(4)≤式(3)<式(2),即 DSMMFI-DS 算法的时间复杂度要少于 MMFI-DS 算法的时间复杂度,DSMMFI-DS 算法的时间复杂度小于 DSM-MFI 算法的时间复杂度。

(2)空间复杂度。

DSM-MFI 算法在内存中需要保存 *FI-list* 表、*OFI-list* 表和 *SFI-tree* 树;MMFI-DS 算法在内存中需要保存 *FI-list* 表、*OFI-list* 表和 *EIS-tree* 树;DSMMFI-DS 算法在内存中保存 *DSFI-list* 表和 *DSSEFI-tree* 树。DSM-MFI 算法和 MMFI-DS 算法都用了投影子集的方法,将投影子集项存放在 *OFI-list* 表中,这样占用了一定大小的内存空间,DSMMFI-DS 算法用 *DSFI-list* 表来存放频繁一项集,一方面,不需要存储每个项目的投影子集,就节省了内存的空间,另一方面因用 *DSSEFI-tree* 树结构的存储经过全序排序后树的分支少,结构简单,同时在节点数据域中增加了监控参数 *monitor*,这样可以尽早地删减已经过期的或者对用户没有意义的频繁项,从而更节省了内存的开销。假设频繁 1-项目的树为 k,DSM-MFI 算法需要存储 2^k 个节点,MMFI-DS 算法需要存储 $C\times k$(C 为常数)个节点,DSMMFI-DS 算法则需要存储 $C\times r$ 个节点(r 为 k 个频繁项集删除过期或者用户不感兴趣的项后剩下的频繁项目树 $r\leqslant k$),显然,$C\times r\leqslant C\times k<2^k$,所以 DSMMFI-DS 算法比 DSM-MFI 算法和 MMFI-DS 算法在相同环境中处理同样数据流更节省内存空间。

5.2　用例分析

假设某一数据流 $W1=[c,f,a,d,e,p,m;f,c,d,a;m,a,b,c,d;b,c,a,m,p,i]$,从窗口中读入完数据存储到 *DSFI-list* 中的结构。

如图 1 所示,假定用户给定的最小支持度为 3,从 *DSFI-list* 中删除,然后根据 *DSFI-list* 列表中的顺序存储到 *DSSEI-tree* 中的结构,如图 2 所示。

同样地用 DSM-MFI 算法得到的 *SFI-tree* 树存储结构如图 3 和图 4 所示。

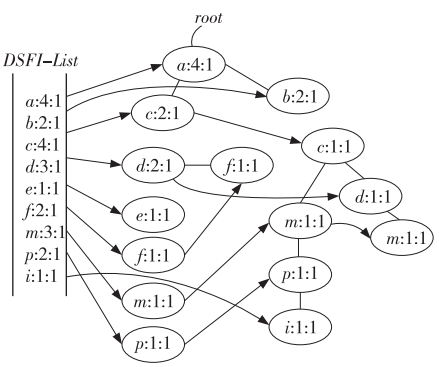

Figure 1　Tree storage structure of *DSSEFI-tree* after DSMMFI-DS algorithm reads the data streams

图 1　DSMMFI-DS 算法读取完数据流后 *DSSEFI-tree* 树存储结构

从上面的结构图我们可以看到,MMFI-DS 算

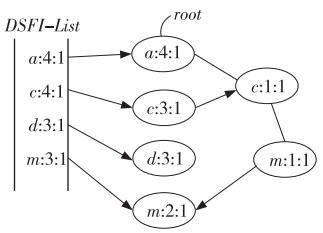

Figure 2 Tree storage structure of *DSSEFI-tree* after DSMMFI-DS algorithm removes not frequent itemsets

图 2 DSMMFI-DS 算法删除不频繁项集后的 *DSSEFI-tree* 树存储结构

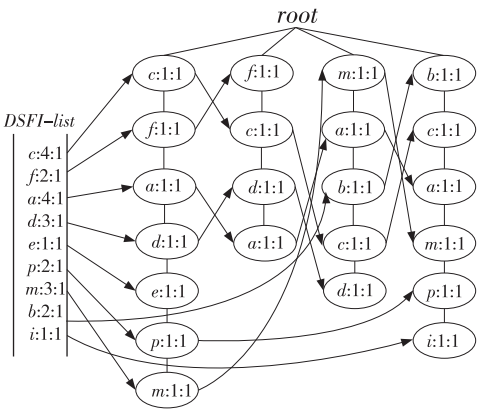

Figure 3 Tree storage structure of *SFI-tree* after DSM-MFI algorithm reads the data streams

图 3 DSM-MFI 算法读取完数据流后 *SFI-tree* 树存储结构

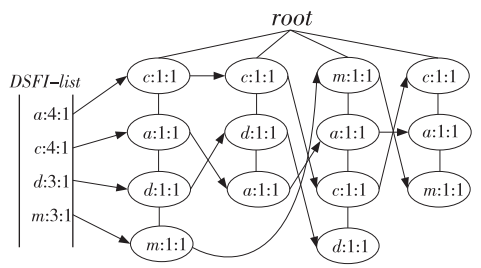

Figure 4 Tree storage structure of *SFI-tree* after DSM-MFI algorithm removes not frequent itemsets

图 4 DSM-MFI 算法删除非频繁项集后的 *SFI-tree* 树存储结构

法由于没有对读入的数据流进行排序,那么它的有些计数少的项目存储到 *EIS-tree* 树中可能成为根节点,这样我们在用自顶向下的算法搜索时发现最大频繁项集时会花费更多的时间来计算项目的支持度;另一方面,由于没有进行排序,那么频繁项目在存储时可能分布在树的各个分支中,这样在存储空间方面又会造成浪费。在 DSM-MFI 算法里,不但要存储 *IsFI-forest*,而且还要存储每个项的投影子集,在同样多的事务数据下需要更多的内存来存储。而 DSMMFI-DS 算法在读入时对项目按一定的顺序排序后,那么它们共享的前缀个数就多了,使树的存储结构得到了简化,节省了存储空间;另一方面,在挖掘最大频繁项集时也由于共享的前缀个数多,所以会在较短的时间里发现最大频繁项集。

6 实验结果和分析

实验在 CPU 为 Intel Pentium(R) Dual-Core 3.20 GHz、内存为 2.00 GB、操作系统为 Windows 7 Ultimate 的 PC 机上进行,所有的实验程序均采用 Visual C++ 6.0 实现。实验中的模拟数据由 IBM 数据生成器产生,为了符合数据流产生的特点,我们设定产生 10 000 k 条数据项,其中 1 k 表示 1 000条数据项,同时我们设定每个窗口的事务数据以及其他所有参数都使用默认值。用户设定的最小支持数为 0.1%,最大允许误差 ε 固定为 0.1Xs。对 DSM-MFI 和 DSMMFI-DS 两种算法执行时的时间和占内存空间进行比较,图 5 显示 DSMMFI-DS 算法比 DSM-MFI 算法的执行时间要少,一方面 DSMMFI-DS 算法不需要为每个项建立投影,另一方面在发现最大频繁项集时 DSMMFI-DS 采用双向搜索策略,所以 DSMMFI-DS 算法比 DSM-MFI 算法的执行效率要高。图 6 显示 DSMMFI-DS 算法比 DSM-MFI 算法在运行时所占的内存空间要小,特别是在随着窗口数目增多时,这种效果更明显:

(1)DSM-MFI 需要为每个项做投影,内存需要存储每个项的投影子集;

(2)每个项在建立相应的树时没有进行排序,那么树的分支也会比较多,存储时需要更多的内存空间;

(3)DSMMFI-DS 算法会根据每个项的窗口衰减支持数,及时将支持数小的项删除。因此,DSMMFI-DS 算法所占的内存空间比 DSM-MFI 要小。

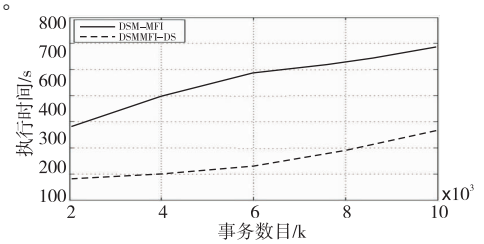

Figure 5 Execution time of DSM-MFI algorithm and DSMMFI-DS algorithm

图 5 DSM-MFI 算法和 DSMMFI-DS 算法执行时间

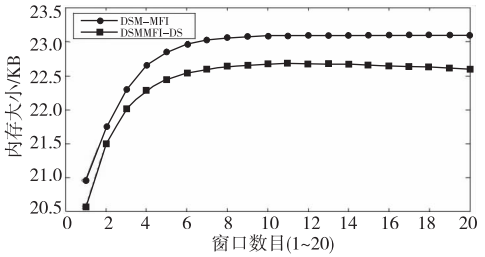

Figure 6　Memory of DSM-MFI algorithm and DSMMFI-DS algorithm

图 6　DSM-MFI 算法和 DSMMFI-DS 算法存储内存

MMFI-DS 算法也是对 DSM-MFI 算法的改进,为了验证 MMFI-DS 算法和 DSMMFI-DS 算法的优越性,也对 MMFI-DS 算法进行了实验对比。如图 7 和图 8 所示。

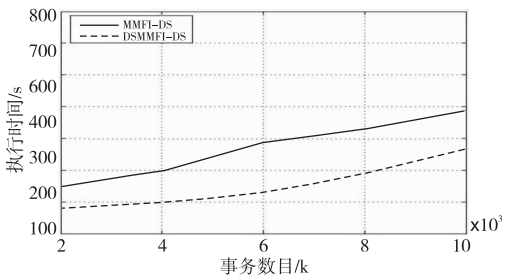

Figure 7　Execution time of MMFI-DS algorithm and DSMMFI-DS algorithm

图 7　MMFI-DS 算法和 DSMMFI-DS 算法执行时间

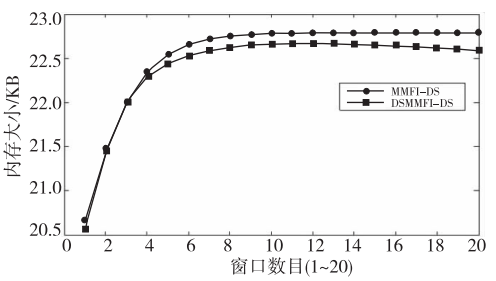

Figure 8　Memory of MMFI-DS algorithm and DSMMFI-DS algorithm

图 8　MMFI-DS 算法和 DSMMFI-DS 算法存储内存

通过实验可以看出,DSMMFI-DS 算法比 MMFI-DS 算法的执行效率更高。在图 7 中,由于 DSMMFI-DS 算法采用的是全序关系存储数据,与数据到来的时间顺序没有关系,在存储上更节省时间;而 MMFI-DS 算法没有按一定的顺序存储,在进行挖掘操作时需要花费很多时间来搜索。在图 8 中,在窗口数目比较小时,MMFI-DS 算法和 DSMMFI-DS 算法的存储空间相差不大,但是随着窗口数目的不断增多,DSMMFI-DS 算法更具有优越性,占用的内存空间小,主要是因为 DSMMFI-DS 算法能够及时地将不频繁项集从树中删除。

7　结束语

本文提出了改进的界标窗口内数据流最大频繁项集挖掘算法 DSMMFI-DS,该算法采用了按全序排序存储,不需要对每个项进行投影,并存储投影子集;该算法在发现最大频繁项集时采用自顶向下和自底向上的双向搜索策略,能快速发现最大频繁项集,最后该算法存储每个项的窗口衰退支持数,根据设定的窗口衰退支持数阈值能够尽早地删除那些支持数小的项,节省了内存空间。通过实验得出 DSMMFI-DS 算法比 DSM-MFI 算法和 MMF-DS 算法具有更好的执行效率。

参考文献:

[1] Li H, Lee S, Shan M. Online mining (recently) maximal frequent itemsets over data streams[C]// Proc of the 15th International Workshops on Research Issues in Data Engineering: Stream Data Mining and Applications, 2005: 11-18.

[2] Han J, Kamber M. Data mining: Concept and techniques[M]. 2nd edition. San Fransisco: Higher Education Press, 2001.

[3] Pan Yun-he, Wang Jin-long, Xu Cong-fu. State of the art on frequent pattern mining in data streams[J]. Acta Automatica Sinica, 2006, 32(4): 594-602. (in Chinese)

[4] Meng Cai-xia. Research on mining frequent itemsets in data streams[J]. Computer Engineering and Applications, 2010, 46(24): 138-140. (in Chinese)

[5] Zhuang Bo, Liu Xi-yu, Long Kun. TWCT-stream: Algorithm for mining frequent patterns in data streams[J]. Computer Engineering and Applications, 2009, 45(20): 147-150. (in Chinese)

[6] Yan Yue-jin. Research on mining algorithms of maximal frequent item sets[D]. Changsha: National University of Defense Technology, 2005. (in Chinese)

[7] Yan Yue-jin, Li Zhou-jun, Chen Huo-wang. A depth-first search algorithm for mining maximal frequent itemsets[J]. Journal of Computer Research and Development, 2005, 42(3): 462-467. (in Chinese)

[8] Huang Shu-cheng, Qu Ya-hui. Survey on data stream classification technologies[J]. Application Research of Computers, 2009, 26(10): 3604-3609. (in Chinese)

[9] Ji Gen-lin, Yang Ming, Song Yu-qing, et al. Fast updating maximum frequent itemsets[J]. Chinese Journal of Computer, 2005, 28(1): 128-135. (in Chinese)

[10] Li Hua Fu, Lee Suh Yin. Mining frequent itemsets over data streams using efficient window sliding techniques[J]. Expert Systems with Applications, 2009, 36(2, Part 1): 1466-1477.

[11] Giannella C, Han Jia-wei, Pei Jian, et al. Mining frequent patterns in data streams at multiple time granularities[M]// Data Mining: Next Generation Challenges and Future Directions, Cambridge: MIT, 2003.

[12] Ma Da,Wang Jia-qiang. A compressed FP-tree based on algorithm for mining maximal frequent itemsets[J]. Journal of Changchun University of Science and Technology(Natural Science Edition), 2009,32(3):457-461. (in Chinese)

[13] Ao Fu-jiang,Yan Yue-jin,Liu Bao-hong, et al. Online mining maximal frequent itemsets in sliding window over data streams[J]. Journal of Sytem Simulation, 2009,21(4):1134-1139. (in Chinese)

[14] Li Hai-feng,Zhang Ning. Maximal frequent itemsets mining method over data stream[J]. Computer Engineering, 2012,38(21):45-48. (in Chinese)

[15] Hu De-min, Zhao Rui-ke. An improved algorithm for mining maximum frequent itemsets[J]. Computer Applications and Software, 2012,29(12):186-188. (in Chinese)

[16] Zhang Yue-qin, Chen Dong. Mining method of data stream maximum frequent itemsets[J]. Computer Engineering, 2010,36(22):86-90. (in Chinese)

[17] Hua Hong-juan, Zhang Jian, Chen Shao-hua. Mining algorithm for constrained maximum frequent itemsets based on frequent pattern tree[J]. Computer Engineering, 2011,37(9):78-80. (in Chinese)

[18] Li H, Lee S, Shan M. An efficient algorithm for mining frequent itemsets over the entire history of data streams[C]// Proc of the 1st International Workshop on Knowledge Discovery in Data Streams, held in conjunction with the 15th European Conference on Machine Learning (ECML 2004) and the 8th European Conference on the Principles and Practice of Knowledge Discovery in Databases (PKDD 2004), 2004:1.

[19] Mao Yi-min,Yang Lu-ming,Li Hong, et al. An effective algorithm of mining maximal frequent patterns on data stream[J]. Chinese High Technology Letters, 2010,3(3):100-107. (in Chinese)

[20] Song Yu-qing,Zhu Yu-quan,Sun Zhi-hui, 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. (in Chinese)

附中文参考文献:

[3] 潘云鹤,王金龙,徐从富.数据流频繁模式挖掘研究进展[J].自动化学报,2006,32(4):594-602.

[4] 孟彩霞.面向数据流的频繁项集挖掘研究[J].计算机工程与应用,2010,46(24):138-140.

[5] 庄波,刘希玉,隆坤.TWCT-Stream:数据流上的频繁模式挖掘算法[J].计算机工程与应用,2009,45(20):147-150.

[6] 颜跃进.最大频繁项集挖掘算法的研究[D].长沙:国防科学技术大学,2005.

[7] 颜跃进,李舟军,陈火旺.一种挖掘最大频繁项集的深度优先算法[J].计算机研究与发展,2005,42(3):462-467.

[8] 黄树成,曲亚辉.数据流分类技术研究综述[J].计算机应用研究,2009,26(10):3604-3609.

[9] 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.

[12] 马达,王佳强.一种基于压缩 FP-树的最大频繁项集挖掘算法[J].长春理工大学学报(自然科学版),2009,32(3):457-461.

[13] 敖富江,颜跃进,刘宝宏,等.在线挖掘数据流滑动窗口中最大频繁项集[J].系统仿真学报,2009,21(4):1134-1139.

[14] 李海峰,章宁.数据流上的最大频繁项集挖掘方法[J].计算机工程,2012,38(21):45-48.

[15] 胡德敏,赵瑞可.一种改进的最大频繁项集挖掘算法[J].计算机应用与软件,2012,29(12):186-188.

[16] 张月琴,陈东.数据流最大频繁项集挖掘方法[J].计算机工程,2010,36(22):86-90.

[17] 花红娟,张健,陈少华.基于频繁模式树的约束最大频繁项集挖掘算法[J].计算机工程,2011,37(9):78-80.

[19] 毛伊敏,杨路明,李宏,等.一种有效的数据流最大频繁模式挖掘算法[J].高技术通讯,2010,3(3):100-107.

[20] 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.

作者简介:

胡健(1967-),男,江西赣州人,博士,教授,研究方向为知识工程与知识发现、软件工程。**E-mail**:1050023437@qq.com

HU Jian, born in 1967, PhD, professor, his research interests include knowledge engineering and knowledge discovery, and software engineering.

吴毛毛(1987-),男,江西鹰潭人,硕士生,研究方向为数据挖掘。**E-mail**:wumaomao2010@163.com

WU Mao-mao, born in 1987, MS candidate, his research interest includes data mining.