第34卷 第16期 计 算 机 工 程 2008年8月
Vol.34 No.16 Computer Engineering August 2008
·软件技术与数据库· 文章编号：1000—3428(2008)16—0050—03 文献标识码：A 中图分类号：TP311

数据流中频繁闭合模式的挖掘

程转流[1,2]，胡学钢[1]
(1. 合肥工业大学计算机与信息学院，合肥 230009；2. 铜陵学院计算机系，铜陵 244000)

摘 要：频繁闭合模式集可唯一确定频繁模式完全集。根据数据流的特点，提出一种挖掘频繁闭合项集的算法，该算法将数据流分段，用DSFCI_tree动态存储潜在频繁闭合项集，对每一批到来的数据流，建立局部DSFCI_tree，进而对全局DSFCI_tree进行更新并剪枝，从而有效地挖掘整个数据流中的频繁闭合模式。实验表明，该算法具有良好的时间和空间效率。

关键词：数据挖掘；数据流；关联规则；频繁闭合项集

Frequent Closed Patterns Mining over Data Streams

CHENG Zhuan-liu[1,2], HU Xue-gang[1]
(1. School of Computer & Information, Hefei Technology University, Hefei 230009;
2. Department of Computer Science, Tongling College, Tongling 244000)

【Abstract】 The set of frequent closed patterns uniquely determines the complete set of all frequent patterns. According to the features of data streams, a new algorithm is proposed for mining the frequent closed patterns. The data streams are partitioned into a set of segments, and a DSFCI_tree is used to store the potential frequent closed patterns dynamically. With the arrival of each batch of data, the algorithm builds a corresponding local DSFCI_tree, then updates and prunes the global DSFCI_tree effectively to mine the frequent closed patterns in the entire data streams. The experiments and analysis show that the algorithm has good performance.

【Key words】 data mining; data streams; association rule; frequent closed itemsets

1 概述

数据流是一种潜在无限的、连续快速的、随时间不断变化的数据序列，挖掘数据流中的频繁模式已成为数据挖掘的热点之一。近年来，有不少数据流频繁模式挖掘算法[1-3]被陆续提出。这些算法大致可分为2大类：

(1)基于概率误差区间的近似算法[2-3]，以较高的置信度将频率估计的相对误差控制在一个较小范围内；

(2)基于确定误差区间的近似算法[2]，这类算法将数据流进行分段，在段的边界处，根据允许的误差 ε 舍弃不满足支持度要求的项集。

大多数情况下，数据流中的频繁模式的数量非常大，文献[4]提出了频繁闭合模式的概念，它可以唯一确定所有的频繁模式及其支持度且数量要小得多。随后，一些求解固定数据库的频繁闭合模式算法[5]被提出，但它们都不适用于数据流的动态环境。为此，本文提出一种新的算法来挖掘数据流中的频繁闭合模式。

2 问题定义和描述

定义1 $I=\{x_1,x_2,\cdots,x_m\}$ 是给定的全部项目集合。一个项集是全部项目集合的一个子集，也称为模式。数据流 DS 是一个依次到达的事务序列 $(t_1,t_2,\cdots,t_N,\cdots)$。其中，每个事务 t_i 也是一个项集。给定一个项集 X，DS 中所有包含 X 的事务数称为 X 的支持数，记为 $X.sup$。

定义2 设给定的支持度 S 和允许的误差 ε，N 表示到目前为止所见到的数据流 DS 的事务数，对于项集 X，如果有 $X.sup\geq(S-\varepsilon)N$，则称 X 为频繁项集；如果 $X.sup>\varepsilon N$，则称 X 为潜在频繁项集；如果 $X.sup\leq\varepsilon N$，则称 X 为非频繁项集。

定义3 对于频繁项集 X，若不存在同时满足下列条件的项集 Y：(1)$X\subset Y$；(2) $X.sup=Y.sup$，则称 X 为频繁闭合项集，类似可以定义潜在频繁闭合项集。

定理1 包含项集 X 且与 X 有相同支持度的闭合项集有且仅有一个。

证明：对于任一个项集 X，总能找到一个包含 X 的最大项集 Y，$X\subset Y$，使得 $X.sup=Y.sup$，且不存在任何项集 Z，$Y\subset Z$；或对于任何项集 Z，$Y\subset Z$，$Z.sup<Y.sup$。根据定义可知，Y 是包含 X 的闭合项集。

设 Y 是一闭合项集，$X\subset Y$，且 $X.sup=Y.sup$；设存在另一闭合项集 Z，$X\subset Z$，且 $X.sup=Z.sup$。因为 $Y\subseteq Y\cup Z$，$(Y\cup Z).sup=X.sup=Y.sup$；$Z\subseteq Y\cup Z$，$(Y\cup Z).sup=X.sup=Z.sup$；又 Y 和 Z 都是闭合项集。所以，$Y\cup Z=Y$ 且 $Y\cup Z=Z$。因此，$Y=Z$，称 Y 为 X 的闭包。证毕。

定义4(DSFCI_tree的结构) DSFCI_tree是一个前缀压缩树，它的结构如下：

(1)由树根(记为null)、潜在频繁闭合项集构成的前缀子树和潜在频繁项头表组成。

(2)根节点之外的每一个节点由7个域组成(itemname, support, isfci, update, parent, child, nextnode)。其中，itemname为该节点所代表的项目名；support表示从根节点的直接子节

基金项目：安徽省高等学校自然科学基金资助项目(KJ2007B236)；安徽省高等学校青年教师科研基金资助项目(2008jq1143)
作者简介：程转流(1979－)，女，讲师、在职硕士研究生，主研方向：数据流挖掘，多Agent系统；胡学钢，教授
收稿日期：2007-09-27 **E-mail：**cz19612@tlu.edu.cn

点到该节点所构成的项集的支持数；isfci 表示从根节点的直接子节点到该节点所构成的项集是否为闭合项集，取值为 1 时表示是闭合项集；update 表示该节点所代表的项集的支持数是否被更新过；parent 指向其父节点；child 指向其孩子节点；nextnode 指向下一个同项目名节点，如果没有这样的一个节点，就为 null。

(3)头表中每个元组由 3 部分组成：项目名，支持数和指向树中有相同项目名的第 1 个节点。

例：某交易数据库{$t1$: ABC, $t2$: BC, $t3$: $ABCD$, $t4$: A}，假定最小支持数为 1，可求其频繁闭合项集{$ABCD$: 1, ABC: 2, BC: 3, A: 3}，则可生成 DSFCI_tree 如图 1 所示。

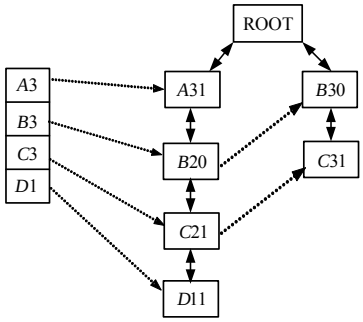

图 1　DSFCI_tree 结构

3　DSFCI 算法

3.1　算法的基本思想

首先将数据流分段为 $N_i(i=1,2,\cdots)$，每段长度 $|N_i|=\lceil 1/\varepsilon \rceil$ 或 $\lceil 1/\varepsilon \rceil$ 的整数倍，为便于算法的描述，取 $|N_i|=\lceil 1/\varepsilon \rceil$。以 ε 为最小支持度，调用已有的频繁闭合项集算法计算第 1 段数据流的所有潜在频繁闭合项集，并以这些闭合项集来创建 DSFCI_tree；对随后的第 $i(i\geq 2)$ 段数据流，同样先计算其闭合项集，将这些项集及其子集压缩到第 i 段的局部 DSFCI_tree 中，再将第 i 段的局部 DSFCI_tree 和前 $i-1$ 段的全局 DSFCI_tree 合并成前 i 段的全局 DSFCI_tree。随着每段数据的流入，不断增量更新 DSFCI_tree，并在给定的误差内，对该树进行剪枝，这样就可以利用 DSFCI_tree 有效地挖掘数据流中的所有频繁闭合项集。

3.2　局部 DSFCI_tree 的生成

对于首批数据，局部 DSFCI_tree 仅仅存储闭合项集，而对于随后的每批数据，需要将闭合项集及其子集都压缩存储到前缀树中，以便测试是否有新的闭合项集产生。

Build_iDSFCItree 算法流程如下：

输入 N_i段数据流、允许误差 ε 和前i-1 段的old_f_list(如果i=1, old_f_list为空)

输出 N_i段的局部DSFCI_tree和前i段的new_f_list

Step1 依次读入每一个数据流元素，得到N_i段的潜在频繁项的集合f_list和每项的支持数；

Step2 把 f_list 按支持度非递增排序；

Step3 按照 FP_tree 的生成算法生成该段数据流的 FP_tree；

Step4 以 ε 为支持度，利用改进的 FP_growth 算法来计算该段的潜在频繁闭合项集；

Step5 生成一个初始的 DSFCI_tree，树中只有一个根节点，if (i=1) then 将每一个潜在频繁闭合项集按 f_list 中项的顺序依次插入 DSFCI_tree；else 将每一个潜在频繁闭合项集及其子集按 f_list 中项的顺序依次插入 DSFCI_tree，将闭合项集的末节点的 *isfci* 置为 1，sup 置为 FP_tree 中该项集的支持数；

Step6 将前 i-1 段的 old_f_list 与 f_list 合并得前 i 段的 new_f_list。算法结束。

3.3　全局 DSFCI_tree 的增量更新

定理 2 若项集 X：(1)是某一段数据流的闭合项集；(2)一直没有被剪枝掉；(3)在到目前为止的数据流中是频繁的，则 X 在到目前为止的数据流中是频繁闭合项集。

证明 :设项集 X 是第 i 段数据流的闭合项集(支持数为 a)，如果在其他段中没有出现 X 的超集，又因为条件(3)，显然项集 X 在到目前为止的数据流中是频繁闭合项集；若在其他段中出现了 X 的超集 Y(支持数为 b)，则 X 的支持数也变成 $a+b$，一定大于其超集的支持数。所以 X 也是频繁闭合项集。证毕。

在对 2 棵 DSFCI_tree 进行合并更新时，需要对项集 X 考虑 2 种情况：(1)项集 X 在其中一棵树上是闭合项集，则合并后也一定是闭合项集，只要合并其支持数即可；(2)项集 X 在 2 棵树上均不是闭合项集，则需要分别在 2 棵树上求 X 的闭包 $X1$, $X2$，如果 $X1\cap X2\neq\phi$，则 $X1\cap X2$ 是合并后的闭合项集。

UPDATE_DSFCItree 算法流程如下：

输入 前 i-1 段的全局 oldDSFCI_tree，N_i 段的局部 DSFCI_tree和前 i 段的f_list

输出 前 i 段的全局 newDSFCI_tree

(1)if(i=1) N_i 段的局部 DSFCI_tree 即为前 i 段的全局 newDSFCI_tree；算法结束。

(2)生成前 i 段的初始全局 newDSFCI_tree，包括头表和一个根节点，头表根据前 i 段的 f_list 生成。

(3)从 oldDSFCI_tree 中依次取下潜在频繁闭合项集 X。

(4)for each X {

(5)调用FCI算法求取X在N_i段的局部DSFCI_tree的闭包 FCI(X)，进而得知X在N_i段的支持数a，将项集X在局部 DSFCI_tree上的末节点的update置为 1。

(6)将 X 按 f_list 中项的顺序重新插入到 newDSFCI_tree 中，更新 X 的支持数 $X.sup$=$X.sup$+a; }。

(7)for N_i 段的局部 DSFCI_tree 中每个节点 α，if $\alpha.update$ =0 then {

(8)将从根节点的直接子节点到节点 α 的项集记为 X。

(9)if $\alpha.isfci$=1，then 调用 FCI 算法求取 X 在前 $i-1$ 段的全局 oldDSFCI_tree 的闭包 FCI(X)，获得 FCI(X)的支持度 b，若 newDSFCI_tree 中不存在闭合项集 X，将 X 按 f_list 中项的顺序插入到 newDSFCI_tree 中，更新末节点的支持度为 $X.sup$+b，并将末节点的 isfci 值置为 1。

(10)else 分别调用 FCI 算法求取 X 在前 i-1 段的全局 oldDSFCI_tree 的闭包集 FCI(X)和 N_i 段的局部 DSFCI_tree 的闭包集 FCI'(X)，设它们的支持度分别为 c 和 d，记 FCI(X)∩FCI'(X)=X'；if $X'\neq\phi$ 且闭合项集 X'不存在于 newDSFCI_tree 中，then 将 X 按 f_list 中项的顺序插入到 newDSFCI_tree 中，更新末节点的支持度为 c+d；并将末节点的 *isfci* 值置为 1。}

(11)调用 prune_DSFCItree 算法对 newDSFCI_tree 进行剪枝，删除前 $i-1$ 段的全局 oldDSFCI_tree 和 N_i 段的局部 DSFCI_tree；算法结束。

FCI 算法流程如下：

输入 项集 X，全局或局部 DSFCI_tree

输出 项集 X 在 DSFCI_tree 上闭包项集

将项集 X 按 DSFCI_tree 对应的 f_list 进行排序，设首项

为 x，然后在频繁项头表中，从项 x 的指针开始，依次在 DSFCI_tree 各分支查找节点 β，$\beta.isfci$=1 且从节点 β 到根节点的直接子节点所构成的项集 Y 是 X 的超集，最小的 Y 即为项集 X 在 DSFCI_tree 上的闭包项集，算法结束。

prune_DSFCItree 算法流程如下：

输入 前 i 段的全局 DSFCI_tree

输出 经过剪枝后的前 i 段的全局 DSFCI_tree

for 任一节点 $\beta \in$DSFCI_tree {$\beta.sup=\beta.sup$-1, if $\beta.sup$=0，则删除节点 β}，算法结束。

定理 3 按上述的 prune_DSFCItree 算法对 DSFCItree 进行剪枝，所有闭合项集的支持度与真实支持度误差小于或等于 ε。

证明：设任何闭合项集 X，设到目前为止数据流的段数为 i，X 的支持度减少了 k，根据 prune_DSFCItree 算法可知，$k \le i$，而 $i \le \varepsilon \times i \times \lceil 1/\varepsilon \rceil = \varepsilon N$。其中，$N$ 表示到目前为止所见到的数据流 DS 的事务数，则 $k \le \varepsilon N$，证毕。

Print_FCI 算法流程如下：

输入 前 i 段的全局 DSFCI_tree，支持度 S(设对应的支持数为 SF)

输出 频繁闭合项集

遍历 DSFCI_tree 中的任何一个节点 β，if ($\beta.isfci$=1∧$\beta.sup$≥SF)，输出从节点 β 到树根节点的直接子节点所构成的项集和它的支持数。算法结束。

3.4 完整的 DSFCI 算法

完整的 DSFCI 算法描述如下：

输入 数据流 DS,最小支持度 S 和允许误差 ε

输出 所有的频繁闭合项集

(1)将数据流分段 $N_i(i=1,2,\cdots)$，每段长度 $|N_i|=\lceil 1/\varepsilon \rceil$ 或 $\lceil 1/\varepsilon \rceil$ 的整数倍，为便于算法的描述，取 $|N_i|=\lceil 1/\varepsilon \rceil$；

(2)for 每一段数据流 {

(3)调用 Build_iDSFCItree 算法生成局部 DSFCItree；

(4)调用 UPDATE_DSFCItree 算法生成全局 DSFCItree；

(5)对全局 DSFCItree 进行剪枝；

(6)如果需要的话，调用 Print_FCI 算法输出所有的频繁闭合项集。}

4 实验结果和分析

为进一步测试算法的性能，用 VC++6.0 在内存 384 MB，CPU 为 P4-1.7 GHz，操作系统为 Windows XP 的 PC 机上实现 DSFCI 算法。采用的数据流是由 IBM 合成数据发生器(下载于 www.almaden.ibm.com/cs/quest/syndatd.html/#assocSyn data)所产生的 3 套虚拟数据集：T15I10D1000k, T15I7D1000k, T7I4D2000k。其中，$|T|$表示数据集中事务的平均长度；$|I|$表示潜在频繁项集的平均长度；$|D|$表示事务总的数目，每个数据集都有 1 000 个项，其他参数采用缺省值。闭合项集生成算法采用改进 FP_growth 算法。数据流分段，每段包含 50 000 条事务，支持度为 S，允许误差 ε=0.1×S。

下面来分析算法的时间效率和所需的存储空间。

图 2 的运行时间指的是每读入一段数据流，对该段数据流建立局部 DSFCI_tree 及更新全局 DSFCI_tree 和输出符合支持度要求的频繁闭合模式所花的时间；图 3 的存储量主要是指全局 DSFCI_tree 所占的空间。从 2 个图可以看出：在支持度一定的情况下，随着数据流的不断流入，算法所消耗的时间和空间虽略有增加，但趋于稳定，适合数据流无限性的特点。在图 2 中，随着支持度的降低，符合支持度要求的频繁项集和潜在频繁项集就会大量增加，相应地，频繁闭合项集和潜在频繁闭合项集也会增加。这样更新全局 DSFCI_tree 和输出符合支持度要求的频繁闭合模式所花的时间就会成倍增长；同样，在图 3 中，随着支持度的降低，符合支持度要求的频繁闭合项集和潜在频繁闭合项集就会大量增加，所需的存储空间也会增加，而且存储量在刚开始的几段数据流到来时增长较快，但随后逐渐趋于稳定。这主要是刚开始潜在频繁闭合项集增长较快，而后来通过剪枝，潜在频繁闭合项集的数量变化不大的原因。

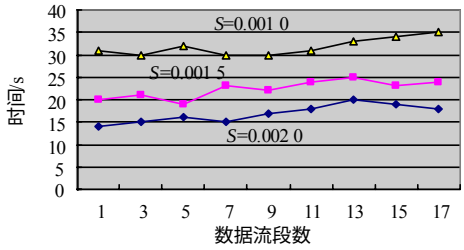

图 2 T15I7D100k 的运行时间

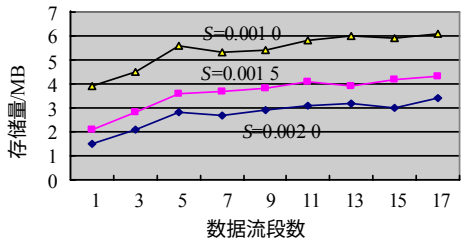

图 3 DSFI_tree 的存储量

图 2 和图 3 都是对数据集 T15I6D1000k 运行的结果数据，对其他两个数据集运行时能得到相类似的情况。

5 结束语

本文根据数据流的特点，提出了一种快速挖掘数据流中的频繁模式算法，其主要贡献在于：(1)提出一种专门存储闭合项集的 DSFCI_tree 结构，并设计了一种快速更新 DSFCI_tree 的算法，不存在模式延迟现象；(2)能在很小的误差允许范围内，对 DSFCI_tree 进行剪枝，大大减少了该树的存储量和遍历该树所花的时间。实验证明：该算法具有很好的时间和空间效率。

参考文献

[1] Giannella C, Han Jiawei, Pei Jian, et al. Mining Frequent Patterns in Data Streams at Multiple Time Granularities[C]//Proc. of the NSF Workshop on Next Generation Data Mining. Cambridge, Mass, USA: MIT Press, 2003.

[2] Manku G S, Motwani R. Approximate Frequency Counts over Streaming Data[C]//Proc. of the 28th Int'l Conference on Very Large Data Bases. Hong Kong, China: [s. n.], 2002.

[3] Arasu A, Manku G S. Approximate Counts and Quantiles over Sliding Windows[C]//Proc. of the 23rd ACM Symposium on Principles of Database Systems. Paris, France: ACM Press, 2004.

[4] Pasquier N, Bastide Y, Taouil R, et al. Discovering Frequent Closed Itemsets for Association Rules[C]//Proc. of the 17th Int'l Conf. on Database Theory. Berlin, German: Springer-Verlag, 1999.

[5] 刘君强, 孙晓莹, 庄越挺, 等. 挖掘闭合模式的高性能算法[J]. 软件学报, 2004, 15(1): 94-102.