第39卷第11期 中国科学技术大学学报 Vol. 39, No. 11
2009年11月 JOURNAL OF UNIVERSITY OF SCIENCE AND TECHNOLOGY OF CHINA Nov. 2009
文章编号:0253-2778(2009)11-1194-08

基于 DSCFCI_tree 的带项目约束的数据流频繁闭合模式挖掘算法

胡为成[1],王本年[1],程转流[1,2]

(1. 铜陵学院信息技术与工程管理研究所,安徽铜陵 244000;2. 合肥工业大学计算机与信息学院,安徽合肥 230009)

摘要:根据数据流的特点,提出了一种挖掘约束频繁闭合项集的算法,该算法将数据流分段,用 DSCFCI_tree 动态存储潜在约束频繁闭合项集,对每一批到来的数据流,首先建立局部 DSCFCI_tree,进而对全局 DSCFCI_tree 进行有效更新并剪枝,从而有效地挖掘整个数据流中的约束频繁闭合模式. 实验表明,该算法具有很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
中图分类号:TP311 **文献标识码**:A

Algorithm for mining frequent closed patterns with item constraint over data streams based on DSCFCI_tree

HU Wei-cheng[1], WANG Ben-nian[1], CHENG Zhuan-liu[1,2]

(1. *Institute of Information Technology & Engineering Management, Tongling College, Tongling* 244000, *China*;
2. *College of Computer Science, Hefei Technology University, Hefei* 230009, *China*)

Abstract: According to the characteristics of data streams, a new algorithm was proposed for mining constrainted frequent closed patterns. The data stream was divided into a set of segments, and a DSCFCI_tree was used to store the potential constrainted frequent closed patterns dynamically. With the arrival of each batch of data, the algorithm first built a corresponding local DSCFCI_tree, then updated and pruned the global DSCFCI_tree effectively to mine the constrainted frequent closed patterns in the entire data stream. The experiments and analysis show that the algorithm has good performance.

Key words: data mining; data streams; association rule; frequent closed itemsets

0 引言

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,但数据流的无限性、快速性和流动性使传统的频繁模式挖掘算法难以适用. 近年来,研究者提出了不少关于数据流频繁模式挖掘算法[1~6],这些算法大致可以分为两大类:①基于概率误差区间的近似算法[2,5,6],即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. 该误差区间并非完全确定,只能以较高频率来保证. ②基于确定误差区间的近似算法[2]. 这类算法将数

收稿日期:2007-12-14;**修回日期**:2008-05-20
基金项目:安徽省高等学校青年教师科研资助计划项目(2008jq1143),安徽省自然科学基金(090416247),安徽省高等学校自然科学研究项目(KJ2007B236)资助.
作者简介:胡为成(通讯作者),男,硕士/副教授. 研究方向:数据挖掘. E-mail: huwc@tlu.edu.cn

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，这样不仅可以降低空间复杂度，还可以得到准确的误差区间.

大多数情况下，数据流中频繁模式的数量非常大，由全部的频繁模式所生成的关联规则数量也是巨大的，含有很多冗余、无用的规则，并且大量的规则也不便于理解和把握. Pasquier 提出了频繁闭合模式的概念[7]，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. 刘学军[8]提出一种基于滑动窗口的数据流频繁闭合模式的挖掘算法，该算法将数据流 DS 分段，每一个分段称为一个基本窗口，记作 BW，一个滑动窗口 SW 对应一个连续的基本窗口序列$\langle BW_1, BW_2, \cdots, BW_k\rangle$，它所容纳的基本窗口的数目是一个定值 k，滑动窗口内的潜在频繁闭合项集及其子集都被动态地压缩到一种新的树结构中，随着新数据的不断到来，滑动窗口以基本窗口为单位不断更新，树结构也随之更新. 因为该算法仅仅对滑动窗口内的所有数据流事务进行处理，在滑动窗口中，新数据不断进入，旧数据不断淘汰，淘汰数据的信息就完全丢失，这样就有可能丢失部分频繁闭合模式.

另外，在实际应用中，大部分用户关注的只是包含特定项集的关联规则，称为约束关联规则，这种约束称为项目约束[9]. 当项目约束被用于数据预处理或将其结合到数据挖掘算法中去时，不仅可以显著减少算法的执行时间和降低算法的空间复杂度，还可以将特定领域的问题结合到关联规则挖掘中，提高挖掘效率，有效地指导挖掘的方向[10,11].

本文提出一种新的算法来挖掘数据流中的约束频繁闭合模式，该算法采用 DSCFCI_tree 结构存贮数据流中的符合项目约束条件的频繁闭合模式，并将约束条件结合到挖掘算法过程中，随着数据流的流入不断更新 DSCFCI_tree 结构，从而有效地挖掘整个数据流中的约束频繁闭合模式.

1　定义和描述

定义 1.1　$I=\{x_1, x_2, \cdots, x_m\}$ 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 1.2　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 $X.\sup \geqslant (S-\varepsilon)N$，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 $X.\sup \leqslant \varepsilon N$，则称 X 为非频繁项集.

定义 1.3　对于频繁项集 X，若不存在同时满足下列条件的项集 Y：(ⅰ) $X \subset Y$；(ⅱ) $X.\sup = Y.\sup$，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 1.1　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，$X \subseteq Y$，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.

定义 1.4　项目约束 B 是项集 I 上的布尔表达式，以析取范式表示，形如 $B_1 \vee B_2 \vee \cdots \vee B_n$. 其中每个 B_i 形如 $i_{i1} \wedge i_{i2} \wedge \cdots \wedge i_{im}$（如果 B_i 中有 $\neg i_{ij}$，暂不考虑此项，最后再一次性删除不满足此类项的约束频繁闭合项集），$i_{ij} \in I$. 对于项集 X，若存在 $k(k=1,2,\cdots,n)$，使得 $B_k \subseteq X$，则称 X 满足约束条件 B. 满足约束条件 B 的频繁闭合项集称为约束频繁闭合项集. 显然，任一项集 X 都满足如下性质：

如果 X 不满足约束条件 B，那么 X 的任何子集均不满足约束条件 B；如果 X 满足约束条件 B，那么 X 的任何超集均满足约束条件 B.

定义 1.5　DSCFCI_tree 的结构

DSCFCI_ 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 6 个域组成(itemname, support, 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点到该节点所构成的项集的支持数，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头表中每个元组由三部分组成：项目名、支持数和指向树中有相同项目名的第 1 个节点.

例 1.1 某交易数据库 $\{t_1: ABC, t_2: BC, t_3: ABCD, t_4: A\}$，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DSCFCI_tree如图 1 所示.

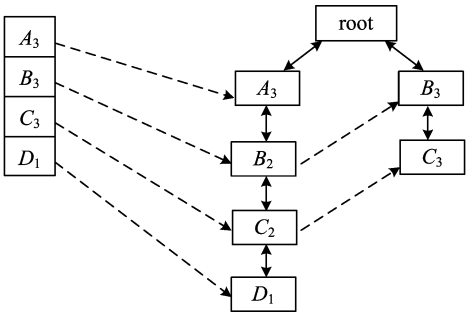

图 1 DSCFCI_tree 结构

Fig. 1 DSCFCI_tree structure

由定义 1.5 可知，对于 DSCFCI_tree 中任一节点 N 及其父节点 P，如果 N. support $=P$. support 则从根节点的直接子节点到节点 P 所构成的项集 X 不是闭合项集，如果 N_support $<P$_support 则 X 是闭合项集，叶子节点所对应的项集肯定是闭合项集. 图 1 中，项集 A、ABC、BC 就是闭合项集，而项集 AB 就不是闭合项集.

2 数据流约束频繁闭合项集挖掘算法 DSCFCI

2.1 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一段数据流的所有潜在频繁闭合项集，并用符合给定约束条件 B 的闭合项集来创建 DSCFCI_tree；对随后的第 $i(i\geqslant 2)$ 段数据流，同样先计算符合约束条件的闭合项集，将这些项集及其符合约束条件的子集压缩到第 i 段的局部 DSCFCI_tree 中，再将第 i 段的局部 DSCFCI_tree 和前 $i-1$ 段的全局 DSCFCI_tree 合并成前 i 段的全局 DSCFCI_tree. 随着每段数据的流入，不断增量更新 DSCFCI_tree，并在给定的误差内，对该树进行剪枝，这样就可以利用 DSCFCI_tree 有效地挖掘数据流中的所有约束频繁闭合项集.

2.2 局部 DSCFCI_tree 的生成

对于第一批数据，局部 DSCFCI_tree 仅仅存贮符合约束条件的闭合项集，而对于随后的每批数据，需要将符合约束条件的闭合项集及其相应子集都压缩存贮到前缀树中，以便测试是否有新的闭合项集产生.

算法 2.1 Build_iDSCFCI_tree 算法

输入：N_i 段数据流、约束条件 B、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 DSCFCI_tree 和前 i 段的 new_f_list.

①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；

②把 f_list 按支持度非递增排序；

③按照 FP_tree 的生成算法生成该段数据流的 FP_tree；

④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；

⑤生成一个初始的 DSCFCI_tree，对每一个符合约束条件的潜在频繁闭合项集 X 及其子集，按 f_list中项的顺序依次插入 DSCFCI_tree，sup 置为 FP_tree 中该项集的支持数；

⑥将前 $i-1$ 段的 old_f_list 与 f_list 合并得前 i 段的 new_f_list.

假设有如表 1 所示的数据流，有 8 条事务，分成两段（每段 4 条事务），每段的最低支持数都设为 1，则可分别建立每段数据流的局部 DSCFCI_tree. 因为第 1 段的局部 DSCFCI_tree，其实就是前 1 段的全局 DSCFCI_tree，所以只要用已有算法求出前 4 条事务的频繁闭合项集 $\{ABCD: 1, ABC: 2, BC: 3, A: 3\}$，再将其插入树中即可，也即前述的图 1. 对于第二批数据流，先要用已有算法求出其闭合项集

表 1 数据流示例

Tab. 1 Example of data streams

tid	transaction
t_1	ABC
t_2	BC
t_3	$ABCD$
t_4	A
t_5	ABD
t_6	B
t_7	CD
t_8	ABD

$\{B:3, D:3, BDA:2, DC:1\}$，再将闭合项集及其子集依次插入第 2 段的局部 DSCFCI_tree 中，得出图 2 如下.

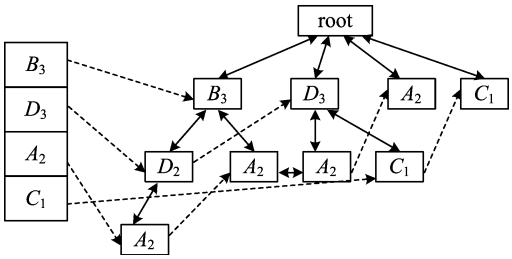

图 2　第二段数据流的局部 DSCFCI_tree

Fig. 2　DSCFCI_tree of the second batch of data stream

2.3　全局 DSCFCI_tree 的增量更新

定理 2.1　若项集 X：(Ⅰ)是某一段数据流的闭合项集；(Ⅱ)一直没有被剪枝掉；(Ⅲ)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(Ⅲ)，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.

在对两棵 DSCFCI_tree 进行合并更新时，需要对项集 X 考虑两种情况：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 在两棵树上均不是闭合项集，则需要分别在两棵树上求 X 的闭包 X_1, X_2，如果 $X_1 \cap X_2 \neq \varnothing$，则 $X_1 \cap X_2$ 是合并后的闭合项集.

算法 2.2　Update_DSCFCI_tree 算法

输入：前 $i-1$ 段的全局 oldDSCFCI_tree，约束条件 B，N_i 段的局部 DSCFCI_tree 和前 i 段的 f_list；

输出：前 i 段的全局 newDSCFCI_tree.

①if ($i=1$) N_i 段的局部 DSCFCI_tree 即为前 i 段的全局 newDSCFCI_tree；算法结束.

②生成前 i 段的初始全局 newDSCFCI_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 oldDSCFCI_tree 中依次取下潜在频繁闭合项集 X；

④for each X {

⑤调用 FCI 算法求取 X 在 N_i 段的局部 DSCFCI_tree 的闭包 FCI(X)，进而得知 X 在 N_i 段的支持数 a；

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

⑦for N_i 段的局部 DSCFCI_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；

⑨if X 为闭合项集，then 调用 FCI 算法求 X 在 oldDSCFCI_tree 的闭包 FCI(X)，获支持数 b，将 X 按 f_list 中项的顺序插入到 newDSCFCI_tree 中，更新末节点的支持度为 $X.\sup + b$；

⑩else　调用 FCI 算法求 X 在 oldDSCFCI_tree 的闭包集 FCI(X)和 N_i 段的局部 DSCFCI_tree 的闭包集 FCI′(X)，获支持数分别为 c 和 d，记 FCI(X)$\cap$FCI′(X)$=X'$；将 X 插入到 newDSCFCI_tree 中，更新支持度为 $c+d$. }

⑪调用 prune_DSCFCI_tree 算法对 newDSCFCI_tree 进行剪枝，删除前 $i-1$ 段的全局 oldDSCFCI_tree 和 N_i 段的局部 DSCFCI_tree.

算法 2.3　FCI 算法

输入：项集 X，全局或局部 DSCFCI_tree.

输出：项集 X 在 DSCFCI_tree 上闭包项集.

将项集 X 按 DSCFCI_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 DSCFCI_tree 各分支查找节点 β，若从节点 β 到根节点的直接子节点所构成的项集 Y 是 X 的超集并且 Y 为闭合项集，最小的 Y 即为项集 X 在 DSCFCI_tree 上闭包项集.

算法 2.4　Prune_DSCFCI_tree 算法

输入：约束条件 B，前 i 段的全局 DSCFCI_tree；

输出：经过剪枝后的前 i 段的全局 DSCFCI_tree.

for 任一节点 $\beta \in$ DSCFCI_tree　{$\beta.\sup = \beta.\sup - 1$，if $\beta.\sup = 0$，则删除节点 β，并检查该分支上的项集是否满足约束条件，若不满足，则删除相应分支}.

在上述 prune_DSCFCI_tree 算法中，将所有节点的支持数减 1，其实就是减去 $|N_i| \times \varepsilon = 1$，如果每段长度 $|N_i|$ 为 $\lceil 1/\varepsilon \rceil$ 的 n 倍，就是减去每段数据流的长度 $|N_i| \times \varepsilon = n$.

同样，我们用表 1 的数据进行示例，也即将图 1 的全局 DSCFCI_tree 和图 2 的第 2 批数据流的局

部 DSCFCI_tree 进行合并. 根据前述的两棵 DSCFCI_tree 合并时应考虑的两种情况的第一种，第一批数据流中的闭合项集{$ABCD$:1, ABC:2, BC:3, A:3}和第二批数据流的闭合项集{B:3, D:3, ABD:2, CD:1}在合并之后仍然是闭合项集，只要准确计算其支持数即可. 对于第二种情况，如项集 AB 在两段数据流中均不是闭合项集，故先求出项集 AB 在两段数据流中的闭包. 利用 FCI 算法可求得项集 AB 在两棵 DSCFCI_tree 上闭包分别为 ABC 和 ABD，其交集为 AB，不为空集，故项集 AB 应是前两段数据流的全局 DSCFCI_tree 的闭合项集. 合并之后的前两段数据流的全局 DSCFCI_tree 如图 3 所示.

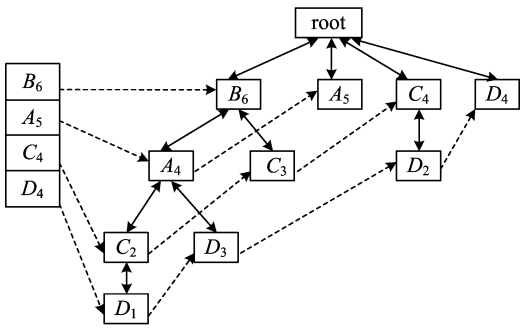

图 3 合并后的 DSFCI_tree

Fig. 3 DSCFCI_tree of the first 2 batches of data stream

合并之前的两棵 DSCFCI_tree 上的闭合项集{$ABCD$:1, ABC:2, BC:3, A:3}和{B:3, D:3, ABD:2, CD:1}在合并之后仍然是闭合的，并且增加了两个新的闭合项集{AB:4, C:4}.

定理 2.2 按上述的 prune_DSCFCI_tree 算法对 DSCFCI_tree 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_DSCFCI_tree 算法可知，$k\leqslant i$，而 $i\leqslant\varepsilon\times i\times\lceil 1/\varepsilon\rceil=\varepsilon N$. 其中 N 表示到目前为止，所见到的数据流 DS 的事务数，则 $k\leqslant\varepsilon N$.

算法 2.5 Print_FCI 算法

输入：前 i 段的全局 DSCFCI_tree，支持度 S，允许误差 ε，约束条件 B，目前为止所见到的数据流 DS 的事务数 N；

输出：频繁闭合项集

遍历 DSCFCI_tree 中的任何一个节点 β，如果：①β 是叶子节点，或 β 的任一子节点 γ，都有 $\beta.\text{support}>\gamma.\text{support}$；②$\beta.\text{support}\geqslant(S-\varepsilon)N$；③从节点 β 到树根节点的直接子节点所构成的项集记为 X，X 完全满足约束条件 B 中先前未考虑的所有否定项，则输出项集 X 和它的支持数.

算法 2.6 完整的 DSCFCI 算法

输入：数据流 DS，约束条件 B，最小支持度 S 和允许误差 ε；

输出：所有的约束频繁闭合项集.

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 $|N_i|=\lceil 1/\varepsilon\rceil$；

②for 每一段数据流 {

③调用 build_iDSCFCI_tree 算法生成局部 DSCFCI_tree；

④调用 Update_DSCFCI_tree 算法生成全局 DSCFCI_tree；

⑤对全局 DSCFCI_tree 进行剪枝；

⑥如果需要的话，调用 print_FCI 算法输出所有的约束频繁闭合项集. }

定理 2.3 设 a 为任意约束闭合项集中的一个项目，$I(\text{DS},\varepsilon)=\{a\in I|a.\text{sup}>\varepsilon N\}$，则 $|I(\text{DS},\varepsilon)|\leqslant L\times\lceil 1/\varepsilon\rceil$. 其中，$L$ 为数据流中事务的平均长度，N 表示到目前为止所见到的数据流 DS 的事务数.

证明 用反证法，因为 $I(\text{DS},\varepsilon)$ 中数据项的支持数都大于 εN，所以 $I(\text{DS},\varepsilon)\times\varepsilon|\text{DS}|\leqslant L\times N$. 假设 $|I(\text{DS},\varepsilon)|>L\times\lceil 1/\varepsilon\rceil$ 成立，则

$$I(\text{DS},\varepsilon)\times\varepsilon N>L\times\lceil 1/\varepsilon\rceil\times\varepsilon N\geqslant L\times 1/\varepsilon\times\varepsilon N=L\times N$$

即 $I(\text{DS},\varepsilon)\times\varepsilon|\text{DS}|>L\times N$.

这与已得到的结论 $I(\text{DS},\varepsilon)\times\varepsilon N\leqslant L\times N$ 相矛盾，命题得证.

定理 2.4 DSCFCI_tree 的最大存储空间为 $O(L/\varepsilon)$. 其中，L 为数据流中事务的平均长度.

证明 根据定理 2.3 和剪枝算法 prune_DSCFCI_tree 可知，需要保存在 DSCFCI_tree 中的约束闭合项集的项目个数不超过 $L\times\lceil 1/\varepsilon\rceil$ 个，也就是说，当允许误差为 ε 时，按剪枝算法所得到的项目，至多仅需要空间 $O(L/\varepsilon)$.

定理 2.4 给出了 DSCFCI_tree 所存储的项目的最大耗费空间. DSCFCI_tree 中所存储的项目数与数据流的事务数和项目的数量没有任何关系，只与 ε 和事务的平均长度有关. 一般来说，事务的平均长度是固定的(通常在 30 以下)，因此，即使 ε 较小

(如 ε 取0.000 1),DSCFCI_tree 也不需要很大的存储空间.

3　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 DSCFCI 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/#assocSyndata)所产生的三套虚拟数据集 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 K 个不同的项,其他参数采用缺省值.闭合项集生成算法采用改进的 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 $\varepsilon=0.1\times S$.约束条件 $B=(1\wedge 2)\vee(3\wedge 4)$,1,2,3,4 是我们从刚开始出现的 30 个项中随机选择的 4 个项.

3.1　算法性能分析

下面来分析算法的时间效率和所需的存贮空间.

图 4 的运行时间指的是:在不同的支持度下,每读入一段数据流,对该段数据流建立局部 DSCFCI_tree 及更新全局 DSCFCI_tree 和输出符合支持度要求的约束频繁闭合模式所花的时间;图 5 的存贮量主要是指在不同的支持度下,全局 DSCFCI_tree 所占的空间.从图 4,5 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 4 中,随着支持度的降低,符合支持度要求和约束条件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 DSCFCI_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 5 中,随着支持度的降低,符合支持度要求和约束条件的频繁闭合项集和潜在频繁闭合项集就会大量增加,所需的存贮空间也会增加,而且存贮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,符合约束条件的潜在频繁闭合项集的数量变化不大的原因.图 4 和图 5 都是对数据集 T15I7D1000k 运行的结果,对其他两个数据集运行的结果与此类似.

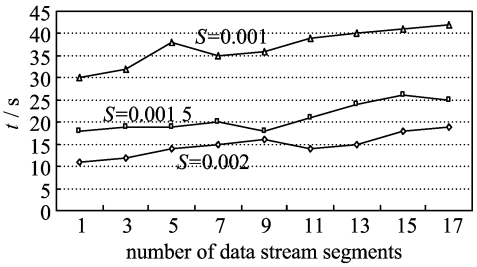

图 4　T15I7D1000k 的运行时间

Fig. 4　Execution time of T15I7D1000k

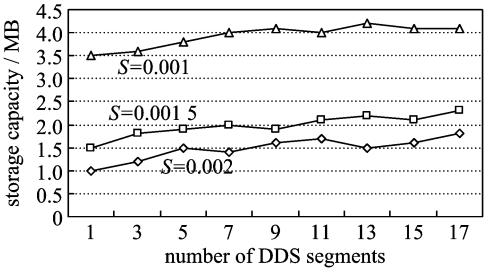

图 5　DSCFCI_tree 的存贮量

Fig. 5　Storage capacity of DSCFCI_tree

图 6 比较了 3 套不同的数据集在不同的支持度下、相同的约束条件 B 的情况下,全局 DSCFCI_tree 每增量更新一次所花的平均运行时间,从中可以看出,数据集中事务的平均长度,潜在频繁项集的平均长度对运行时间有很大的影响.

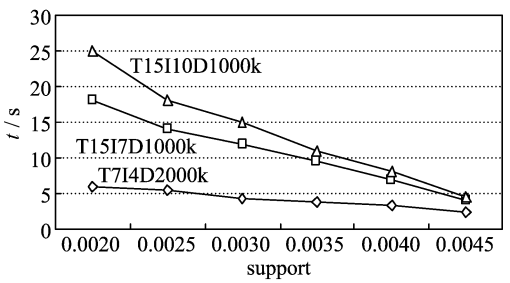

图 6　三组数据集的运行时间对比

Fig. 6　Comparison of execution time of different datasets

图 7 分析了数据集 T15I7D1000k 在三种不同的约束条件下(约束 0:没有约束;约束 1:$B=(1\wedge 2)\vee(3\wedge 4)$;约束 2:$B=1\wedge 2\wedge 3\wedge 4$)全局 DSCFCI_tree 每增量更新一次所花的平均运行时间,由图 7 可以看出,随着约束条件的加强,在相同的支持度($S=0.001\ 5$)和允许误差 $\varepsilon=0.1\times S$ 的情况下,符合约束条件的频繁闭合项集就越少,所以全局 DSCFCI_tree 的增量更新所需的时间就越少,而且全局 DSCFCI_tree 的所占的存储空间也会相应减

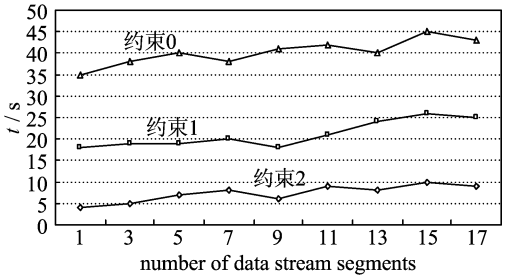

图 7 不同约束下的运行时间对比

Fig. 7 Comparison of execution time with different constraints

小. 由此可见,将项目约束条件嵌入到算法挖掘过程中,可减少算法的执行时间和降低算法的空间复杂度.

3.2 实验对比分析

目前直接在动态数据流上挖掘约束频繁闭合项集的算法不多,文献[9]提出了一种基于滑动窗口的数据流频繁闭合项集挖掘算法,滑动窗口以基本窗口为单位不断更新. 但在动态数据流上挖掘频繁项集的算法比较多,比较典型的是刘学军等在借鉴 FP_growth 算法的基础提出的 FP-DS 算法[12],该算法将数据流中的所有潜在频繁项集压缩在一棵 FP_tree 上,然后再在 FP_tree 上输出所有符合条件的频繁项集. 下面我们分别采用与文献[9,12]相同的实验环境和实验数据集对算法进行测试,对运行结果进行比较.

设 algorithm Ⅰ,algorithm Ⅱ 和 algorithm Ⅲ 分别表示本文算法、FP-DS 算法和文献[9]提出的算法. 首先将 algorithm Ⅰ 和 algorithm Ⅱ 进行比较,实验数据集是 T7I4D2000k,两算法都是对数据流分段,algorithm Ⅰ 在每段的分界处,如果需要的话,调用 print_FCI 算法输出所有的约束频繁闭合项集,而 algorithm Ⅱ 只能在需要的时候得到相应的潜在频繁 FP_tree,还要在此基础上调用已有的算法求取频繁闭合项集. 在支持度从 0.004～0.01 之间变化时,algorithm Ⅱ 的最大存储空间在 13.8～17.7 MB 之间变动(含辅助内存和临时内存),而 algorithm Ⅰ 由于只存储约束频繁闭合项集和潜在频繁项集,最大存储空间只在 2.1～4.5 MB 变化; algorithm Ⅰ 的运行时间主要花在局部 DSCFCI_tree 创建和全局 DSCFCI_tree 的更新上,而 algorithm Ⅱ 的主要时间开销是每段调用 FP_growth 算法和 reconstruct 算法(重构算法)的时间开销. 此外还有 FP_tree 求取频繁闭合项集的时间,图 8 是两者运行时间的比较. 从图 8 可以看出,algorithm Ⅰ 在时间性能上也优于 algorithm Ⅱ.

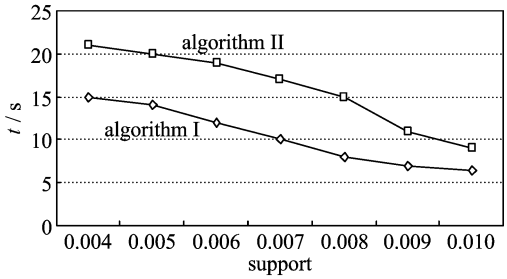

图 8 运行时间的比较

Fig. 8 Comparison of execution time

Algorithm Ⅲ将数据流 DS 分段,每一个分段称为一个基本窗口,一个滑动窗口对应一个连续的基本窗口序列,它所容纳的基本窗口的数目是一个定值 k. 因为该算法仅仅对滑动窗口内的所有数据流事务进行处理,在滑动窗口中,新数据不断进入,旧数据不断淘汰,淘汰数据的信息就完全丢失,这样就有可能丢失部分频繁闭合模式. 实验数据集仍然是 T7I4D2000k,将该算法和本文算法比较,从时间和空间来看,两种算法结果相近,有时甚至本文算法还稍微多一点,但在输出符合条件的频繁闭合项集时,algorithm Ⅰ 输出的频繁闭合项集要多,并且随着 k 值的减少,多得越多. k 值为 1 时,algorithm Ⅲ 仅仅求某一分段的符合条件的频繁闭合项集. 当 k 值维持在一个较大的水平时($k>30$),algorithm Ⅰ 输出的频繁闭合项集比 algorithm Ⅲ 输出的频繁闭合项集大约多 10%左右,并且比较稳定.

4 结论

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:(Ⅰ)提出一种专门存储闭合项集的 DSCFCI_tree 结构,并设计了一种快速更新 DSCFCI_tree 的算法,不存在模式延迟现象;(Ⅱ)能在很小的误差允许范围内,对 DSCFCI_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;(Ⅲ)将项目约束条件嵌入到算法挖掘过程中,进一步减少算法的执行时间和降低算法的空间复杂度. 实验表明,该算法具有很好的时间和空间效率.

参考文献(References)

[1] Giannella C, Han J, Pei J, et al. Mining frequent patterns in data streams at multiple time granularities

[C]//Next Generation Data Mining. Cambridge, Mass: MIT Press, 2003: 191-212.
[2] Manku G S, Motwani R. Approximate frequency counts over data streams[C]// Proceedings of the 28th International Conference on Very Large Data Bases. Hong Kong: VLDB Endowment, 2002: 346-357.
[3] Chang J H, Lee W S. Finding recent frequent itemsets adaptively over online data streams[C]// Proceedings of the 9th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. Washington: ACM Press, 2003: 487-492.
[4] Cormode G, Muthukrishnan S. What's hot and what's not: tracking most frequent items dynamically[J]. ACM Transactions on Database Systems, 2003, 30(1): 249-278.
[5] Arasu A, Manku G S. Approximate counts and quantiles over sliding windows[C]// Proceedings of the 23rd ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems. Paris, France: ACM Press, 2004: 286-296.
[6] Datar M, Gionis A, Indyk P, et al. Maintaining stream statistics over sliding windows[J]. SIAM Journal on Computing, 2002, 31(6): 1 794-1 813.
[7] Pasquier N, Bastide Y, Taouil R, et al. Discovering frequent closed itemsets for association rules[C]// Proceedings of the 17th International Conference on Database Theory. Berlin: Springer-Verlag, 1999, 1 540: 398-416.
[8] 刘学军,徐宏炳,董逸生, 等. 基于滑动窗口的数据流闭合频繁模式的挖掘[J]. 计算机研究与发展,2006,43(10): 1 738-1 743.
[9] Han J,Kamber M. 数据挖掘概念和技术[M]. 范明译,北京:机械工业出版社, 2001.
[10] 陈慧萍,朱峰,王健东,等. 一种基于划分的带项目约束的频繁项集挖掘算法[J]. 系统工程与电子技术,2006,28(7):1 082-1 086.
[11] 宋余庆,朱玉全,孙志挥,等. 一种基于频繁模式树的约束最大频繁项目集挖掘及其更新算法[J]. 计算机研究与发展, 2005, 42(5): 777-783.
[12] 刘学军,徐宏炳,董逸生等. 挖掘数据流中的频繁模式[J]. 计算机研究与发展,2005,42(12):2 192-2 198.

(上接第 1 193 页)

[10] Chan Y T, Ho K C. A simple and efficient estimator for hyperbolic location[J]. IEEE Transactions on Signal Processing, 1994, 42(8): 1 905-1 915.
[11] Ward A, Jones A, Hopper A. A new location technique for the active office[J]. IEEE Personal Communications, 1997, 4(5): 42-47.
[12] Priyantha N B, ChakrabortyA, Balakrishnan H. The cricket location-support system[C]// Proceedings of the 6th Annual International Conference on Mobile Computing and Networking. Boston, MA: ACM Press, 2000: 32-43.
[13] Chen J C, Yip L, Elson J, et al. Coherent acoustic array processing and localization on wireless sensor networks[C]//Proceedings of IEEE, 2003, 91(8): 1 154-1 162.
[14] Elson J, Girod L, Estrin D. Fine-grained network time synchronization using reference broadcasts[C]// Proceedings of the 5th Symposium Operating Systems Design and Implementation. Boston, MA: ACM Press, 2002, 36(S1): 147-163.
[15] Li T, Ekpenyong A, Huang Y F. A location system using asynchronous distributed sensors[C]// Proceedings of the 23rd Annual Joint Conference of the IEEE Computer and Communications Societies. Hong Kong: IEEE Press, 2004, 1: 620-628.
[16] 赵保华,张炜,李婧,等. 传感器网络中的多重贪心路由算法[J]. 北京邮电大学学报, 29(Sup):11-15,2006
[17] Vig J R. Introduction to quartz frequency standards[R]. SLCETTR-92-1, Army Research Laboratory, Electronic and Power Sources Directorate, Fort Monmouth, NJ, 1992.