在线挖掘数据流闭频繁项集的高效算法

毛伊敏[1] 陈志刚[2]
(江西理工大学应科院 赣州 341000)[1] (中南大学信息学院 长沙 410083)[2]

摘 要 数据流闭频繁项集挖掘算法得到了广泛的研究,其中一个典型的工作就是NewMoment算法。针对NewMoment算法存在搜索空间大而造成算法时间效率低的问题,提出了一种改进的数据流闭频繁项集挖掘算法A-NewMoment。它设计了一个二进制位表示项目与扩展的频繁项目列表相结合的数据结构,来记录数据流信息及闭频繁项集。在窗体初始阶段,首先挖掘频繁1-项集所产生的支持度为最大的最长闭频繁项集,接着提出新的"不需扩展策略"和"向下扩展策略"来避免生成大量中间结果,快速发现其余闭频繁项集,达到极大缩小搜索空间的目的。在窗体滑动阶段,提出"动态不频繁剪枝策略"来从已生成的闭频繁项集中快速删除非闭频繁项集,并提出"动态不搜索策略"来动态维护所有闭频繁项集的生成,以降低闭频繁项集的维护代价,提高算法的效率。理论分析与实验结果表明,A-NewMoment算法具有较好的性能。

关键词 数据挖掘,数据流,频繁项集,闭频繁项集
中图法分类号 TP182 **文献标识码** A

Efficient Algorithm for Online Mining Closed Frequent Itemsets over Data Streams

MAO Yi-min[1] CHEN Zhi-gang[2]
(Applied Science Institute of Jiangxi University of Science and Technology, Ganzhou 341000, China)[1]
(School of Information Science and Engineering, Central South University, Changsha 410083, China)[2]

Abstract Mining closed frequent itemsets from data streams has been extensively studied, in which NewMoment is regarded as a typical algorithm. However, there is the problem of its big search space causing bad performance in time in NewMoment. This paper presented an algorithm called A-NewMoment which ameliorates NewMoment to mine closed frequent itemsets. Firstly, it designed a combinative data structure which uses an effective bit-victor to represent items and an extended frequent item list to record the current closed frequent information in streams. Secondly, the new pruning strategies called WSS and CSS were proposed to avoid a large number of intermediate results generated, so the search space is reduced greatly. Finally, the pruning strategy called DNFIPS was also proposed to delete no closed frequent itemsets from HTC. At the same time, it also designed a novel strategy called DNSS to efficiently and dynamically maintain these operations that all closed frequent itemsedts are added and deleted. Theoretical analysis and experimental results show that the proposed method is efficient.

Keywords Data mining, Data streams, Frequent itemsets, Closed frequent item

1 引言

随着计算机技术与网络技术的发展,数据流得到了广泛的应用。相对于传统的静态数据,数据流具有连续性、无序性、无界性及实时性的特点[1],这使得数据流挖掘算法满足如下几个条件[2]:①分析数据流时,所有的数据元素最多只能访问一次;②尽管数据流中数据元素在连续不断产生,但是分析数据流所需要的内存空间必须是有限的;③新产生的流数据必须尽可能快地处理,要求算法的实时性高;④当用户提交查询时,须能及时反馈最新的数据流分析结果,要求算法的时间效率高。与传统的静态数据相比,挖掘数据流信息更具有挑战意义。

频繁项集的子集也是频繁项集,挖掘频繁项集在支持度低的情况下会产生组合爆炸问题,从而严重影响算法的时空效率。闭频繁项集比频繁项集规模小得多而且保存完全支持度信息。因此,当应用的时空效率要求较高,又需要完全的支持度信息时,研究挖掘数据流闭频繁项集更具意义。

目前挖掘数据流闭频繁项集得到了广泛的研究[3-8]。主要经典的算法包括Moment[3]、CFI-stream[4]、Stream-Close[5]和NewMomen[6]等。Chi等人首次提出挖掘数据流闭频繁项集Moment算法,该算法采用前缀项目树存储、维护结点信息,通过结点类型转换,对其进行不同的处理从而挖掘闭频繁项集。由于算法需耗费大量的时间维护结点的信息,因此严重影响其时间效率。NanJiang等提出了一种对Moment改进

到稿日期:2012-04-20 返修日期:2012-07-05 本文受国家自然基金项目(51164012基金项目号),江西教育厅科技项目(GJJ12347号)资助。
毛伊敏(1970—),女,博士,副教授,主要研究方向为数据挖掘,E-mail:mymlyc@163.com;陈志刚(1964—),男,博士,教授,主要研究方向为数据挖掘、网格计算。

的算法 CFI-Stream，算法以基于内存的 DIU 树为存储结构，维护滑动窗口内的频繁闭合项集。通过对进入或离开滑动窗口的每个事务进行闭包检查实施闭频繁项集的挖掘；StreamClose 算法使用 LDIUtree 存储当前闭频繁项集，设计“最大项目匹配”和“相同前缀”剪枝策略缩小搜索空间，在挖掘过程中采用“向前看”方法进行闭包检测从而获得当前窗口的闭频繁项集。文献[4,5]采用基于前缀树存储结构捕捉数据流信息，但其存在结点维护代价大、算法的实时性不够高的问题。由于当前的计算机一次操作的位数越来越多，使得位图操作的速度越来越快[9]。NewMoment 算法采用二进制位序列表示每个事务的项目，采用二进制位与操作发现窗体初始阶段的闭频繁项集。当窗口滑动时，使用位移动技术移去过时的事务、新增最新的事务，实时在线地把所有闭频繁项集呈现给用户。该算法采用了二进制位滑动技术表示数据流信息与二进制位与操作计算项集的支持度，有效地提高了算法的时间效率，是目前性能最好的数据流闭频繁项集挖掘算法。但是 NewMoment 算法存在一个主要问题：没有采用剪枝策略，造成具有极大的搜索空间及保存大量中间结果用于剪枝，影响了算法的时间效率；针对这些问题，本文提出了数据流闭频繁项集挖掘算法——A-NewMoment（an ameliorated NewMoment algorithm），该算法设计了一种新的 BV-DFIlist（dictionary frequent itemsets list with bit-vectors）存储结构来记录数据流信息及闭频繁项集。在窗体初始阶段，首先对频繁 1-项集按其支持度由低到高进行排序，发现各频繁 1-项集所产生的支持度最大的最长闭频繁项集；接着使用提出的新的剪枝策略——“不需扩展策略”与“向下扩展策略”，来避免生成大量中间结果，从而快速挖掘出其余闭频繁项集。在窗体滑动阶段，提出“动态不频繁剪枝策略”来从存储闭项集的 HTC 表中快速删除非闭频繁项集，并提出“动态不搜索策略”来维护所有发生变化的闭频繁项集，以降低闭频繁项集的维护代价，提高算法的时间效率。

2 问题定义

数据流 $DS=\{T_1,T_2,\cdots,T_i,\cdots\}$ 为一个大小无限的事务数据序列，其中 T_i 为第 ith 个产生的事务，$T_i=\{I_1,I_2,\cdots,I_m,\cdots\}$，$I_i(i=1,2,\cdots,p)$ 称为项目。滑动窗口 SW（Sliding Window）包含数据流 DS 上最近的 N 个事务数据，$|N|$ 表示 SW 的大小。

定义 1 项目集 $x\subseteq I$，事务集 $T\subseteq DS$，定义映射关系：$g:2^I\to 2^T$，$g(x)=\{t\in T|$事务 t 支持项目 $x\}$，称 $g(x)$ 为项集 $x\subseteq I$ 的支持集。$f:2^T\to 2^I$，$f(T)=\{c\in I|T$ 中的任何事务均支持项目 $c\}$，称 $f(T)$ 是 $T\subseteq DS$ 的公共项集。

定义 2 $x\subseteq I$ 的绝对支持度为 $spt(x)=|g(x)|$，相对支持度为 $|g(x)|$ 除以 $|T|$ 的值。

定义 3 项目集 $x\subseteq I$，2^I 上闭包算子 $h(x)=f(g(x))$，$CFI\subseteq I$ 是闭频繁项集，当且仅当 $h(CFI)=CFI$。

定义 4 项目集 $x\subseteq I$，项目集 x 的包含索引是：$subsume(x_i)=\{j\in I|j\neq x_i\cap g(x_i)\subseteq g(j)\}$。

定义 5 项集 p 是闭频繁项集 CFI 的生成子，若 $h(p)=CFI$ 且不存在 $A\subset p$，则 $h(A)=CFI$。

定义 6 给定生成子 $gen=Y\cup i$，其中 Y 是一个闭频繁 1-项集，$i\notin Y$，定义生成子的支持度后序集：$post\_set_s(gen)=\{j\in FI\text{-}list|j\notin gen\cap spt(i)<spt(j)\}$。

定义 7 给定生成子 $gen=Y\cup i$，其中 Y 是一个闭频繁 1-项集，$i\notin Y$，定义生成子的支持度前序集：$pre\_set_s(gen)=\{j\in FI\text{-}list|j\notin gen\cap spt(i)>spt(j)\}$

3 A-NewMoment 算法

本节介绍一个有效的挖掘数据流闭频繁项集挖掘算法 A-NewMoment，该算法由两个阶段构成：第一阶段是窗体初始化阶段，第二阶段是窗体滑动阶段。

3.1 数据结构

3.1.1 项目集的表示

文献[6]首次提出采用二进制位滑动技术移走过时的事务，新增当前的事务，随后许多学者都采用此方法表示最近的数据流信息。但是，这种方法在窗体滑动时，每个项目先进行二进制位左移操作，然后将新增事务的值添加到最右边的位置，极大地耗费了维护代价，降低了时间效率。本文提出用项目字典二进制位置换技术置换过时的事务，新增当前的事务，这样，每个项目无需左移操作，直接设置新增事务的值，从而减小维护代价，提高时间效率。设项目 x_i，如果 x_i 在当前事务中出现则用 1 表示，否则用 0 表示。当窗口移动时，用最近事务所包含项目的二进制位表示置换过时事务所包含项目的二进制位。例如，设滑动窗口尺寸为 10，事务$\langle T_1,(bea)\rangle$、$\langle T_2,(dfegach)\rangle$、$\langle T_3,(ce)\rangle$和$\langle T_4,(dbeac)\rangle$等为窗口 1 的事务，窗口 1 各事务用项目字典二进制位表示如表 1 所列，当窗体滑动时，新增事务$\langle T_{11},(edach)\rangle$，窗口 2 各事务用项目字典二进制位表示如表 2 所列。

表 1 窗口 1 事务项目字典二进制位表示

Transaction	Items	Dictionary	Bit-dictionary
T_1	bea	abe	11001000
T_2	dfegach	acdefgh	10111111
T_3	ce	ce	00101000
T_4	dbeac	abcde	11111000
T_5	dfagch	acdfgh	10110111
T_6	ace	ace	10101000
T_7	gaech	acegh	10101011
T_8	gach	acgh	10100011
T_9	agh	agh	10000011
T_{10}	ec	ce	00101000

表 2 窗口 2 事务项目字典二进制位表示

Transaction	Items	Dictionary	Bit-dictionary
T_1	edach	acdeh	10111001
T_2	dfegach	acdefgh	10111111
T_3	ce	ce	00101000
T_4	dbeac	abcde	11111000
T_5	dfagch	acdfgh	10110111
T_6	ace	ace	10101000
T_7	gaech	acegh	10101011
T_8	gach	acgh	10100011
T_9	agh	agh	10000011
T_{10}	ec	ce	00101000

3.1.2 BV-DFIlist 结构

基于上述的项目字典二进制位置换技术的思想，本文提出一个新的数据结构——BV-DFIlist，其特点如下所示：

(1)它由一个 BV-array（a array of bit-vectors of array）、DFIlist（dictionary frequent item list）和 HTC（a hash table of closed itemsets）组成。

(2) DFIlist 的每个项包含 4 个域：itemnane、rownum、count 和 tid。itemname 代表项目名；count 代表项目的频繁支持度计数；rownum 记录项目在 BV-array 中列的位置；tid 记录最近的包含该结点的那个事务的标识。

(3) HTC 是三级哈希表，第一级是按从低到高排序的项集的支持度，第二级是按字典顺序排列的项目，第三级是用来存储所有的频繁闭项集的项目名称。

3.2 窗体初始化阶段

当初始窗口中的数据流流进时，BV-array 存储了数据流的所有信息，对 DFIlist 中的项目按照支持度由小到大进行排序，然后使用二进制位图技术挖掘 DFIlist 中各项目生成子的支持度最大的最长闭频繁项集，最后使用各种剪枝策略快速挖掘其余的闭频繁项集。

定理 1 对于 $\forall x_1, x_2 \in I$，若 $spt(x_1) < spt(x_2)$，则 x_1 比 x_2 所生成的频繁项集少。

证明：$\because spt(x_1) = |g(x_1)|, spt(x_2) = |g(x_2)|$

又 $\because spt(x_1) < spt(x_2)$

根据定理 2 知：

$\therefore g(x_1) < g(x_2)$

即：支持 x_1 的事务数少于支持 x_2 的事务数，

则：则 x_1 比 x_2 所生成的频繁项集少。

由定理 1 与定义 4 知，对频繁 1-项集按支持度由低到高进行排序，每个 1-项集所产生的包含索引含有的频繁项集多于不按支持度排序所产生的包含索引含有的频繁项集。

引理 1[10] 若 1-项集 x 的 $subsume(x) \in \emptyset$，则 x 为闭频繁项集。

定理 2 在 A-NewMoment 算法流程中，若 1-项集 x 的 $subsume(x) \notin \emptyset$，则 $spt(x) = spt(x \cup subsum(x))$ 且 $x \cup subsume(x)$ 为闭频繁项集。

证明：由定义 4 及 $subsum(x) \notin \emptyset$ 知，$g(subsume(x)) = g(x)$。

即：支持 1-项集 x 的 $subsume(x)$ 的事务与支持 1-项集 x 的事务是一样的。

$\therefore$ 支持 1-项集 x 的事务与支持 $x \cup subsume(x)$ 的事务也是一样的。

即：$g(x) = g(x \cup subsum(x))$

$\because spt(x) = |g(x)|, spt(x \cup subsum(x)) = |g(x \cup subsume(x))|$

$\therefore sprt(x) = spt(x \cup subsum(x))$ (1)

根据算法 A-NewMoment 知，不存在项集 Y，使得 $Y \supset subsume(x)$，显然，也不存在项集 Y，使得

$Y \supset x \cup subsume(x)$ (2)。

由式(1)与式(2)知，问题得证。

文献[11]通过判断每个生成子的支持度与其前序集和后序集中每个元素的支持集的包含关系确定该生成子是否保序，从而实现频繁项集构成格的跳跃式搜索。本文除了采用此搜索策略外，还设计了“不需扩展策略”(WSS, without search strategy)与“向下扩展策略”(CSS, continuative search strategy)两个搜索策略，详见定理 3 与定理 4 的分析。

定理 3 在 A-NewMoment 算法流程中，对于 $\forall x_1, x_2 \in$ 1-项集，若 $spt(x_1) = spt(x_2)$ $(x_1 < x_2)$，且 $x_2 \subset subsume(x_1)$，则 $g(x_1) = g(x_2)$。

证明：$\forall x_i \in$ 1-项集，$(i = 1, 2, \cdots, n)$，设 $x_1 < x_2 < \cdots < x_n$，由 A-NewMoment 算法知：

$spt(x_1) \leqslant spt(x_2) \leqslant \cdots \leqslant spt(x_n)$

若 $spt(x_1) = spt(x_2)$ 且 $x_1 < x_2$，则 x_2 处于以下两种情况：

(1) $x_2 \notin subsume(x_1)$，由定义 1 知，$g(x_1) \neq g(x_2)$。

(2) $x_2 \subset subsume(x_1)$，由定义 1 知，$g(x_1) = g(x_2)$，问题得证。

由定理 3 的分析知，1-项集 x_2 不需进行扩展操作，本文称“不需扩展策略”。

定理 4 在 A-NewMoment 算法流程中，对 $\forall i \in post\_set_s(c)$，若 c 为闭频繁项集，则 $\exists (c \cup i)$ 为闭频繁项集。

证明：$\because c \subseteq c \cup i, g(c \cup i) \subseteq g(c)$

由定义 1 和定义 2 得：$spt(c \cup i) \leqslant spt(c)$

$\because spt(c) > s, \therefore \exists (c \cup i)$ 为闭频繁项集。

由定理 4 的分析知，闭频繁项集被挖掘出来，还需对其进行扩展操作，本文称之为“向下扩展策略”。

通过以上分析，在 BV-DFIlist 的数据结构基础上，进行窗体初始化阶段闭频繁项集搜索的过程如算法 1 所示。

算法 1 MingCFI-WinInit

```
输入：一个存储最初窗口事务的 BV-array；
输出：HTC；
For each x_i∈DFIlist,i=1,2,…,n
    If CheckSubsume(x_i, pre_set_s)==.f. and spt(x_i)>s then
  subsume= ∩_{x_j∈g(x_i)} x_j  /*计算所有包含项目 x_i 的事务的交
    If subsume=∅ then
      把 x_i 和它的支持度插入到 HTC 中；
    Else
      把 subsume 和它的支持度插入到 HTC 中；
    Endif
    Search(x_i, post_set_s);
    Endif
  Endfor
  Procedure Search(closed_set, post_set_s)
      While post_set_s≠∅ do
  i=min<(post_set_s);
  post_set_s=post_set_s\i;
  new_gen=closed_set∪i;/*生成一个生成子
  If spt(new_gen)>s
    If new_gen⊆subsume then
      closed_set_new=new_gen;
      post_set_new=∅;
      For all j∈post_set_s do
        If (closed_set_new∪j)⊆subsume then
          closed_set_new=closed_set_new∪j
        Else
          post_set_new=post_set_new∪j
        Endif
      endfor
    Else
      If is_dup(new_gen,pre_set_s)==.f. then
        closed_set_new=new_gen
```

```
post_set_new=∅
For all j∈post_set_s do
  If g(new_gen)⊆g(j) then
    closed_set_new=closed_set_new∪j
  Else
    post_set_new=post_set_new∪j
  Endif
Endfor
把 closed_set_new插入到 HTC 中;
search(closed_set_new,post_new)
Endif
Endif
Endif
Enddo
```

3.3 窗体滑动阶段

当窗体滑动时,需进行过时事务的删除及最新事务的增加操作,这些操作会引起当前窗口闭频繁项集发生变化,因此应设计不同的剪枝与搜索策略来动态维护所有发生变化的闭频繁项集。

3.3.1 删除过期事务

删除过期事务是窗体滑动阶段的第一阶段,当事务超出当前窗口时,最老的、过时的事务的每个项目置 0。因项目 x_i 的流失,其支持度也相应发生变化,因此,已挖掘出来的闭频繁项集也会发生变化。

定理 5 在 A-NewMoment 算法流程中,$\forall x_i \in DFIlist$, $c=\{x_i \cup post\_set_s(x_i)\} \subset CFI$,若 $x_i.count<s$,则 $\forall c_i \in c, c_i.count<s$。

证明:由算法 1 知,项集 c_i 存在两种情况:

(1)当 $subsume(x_i)\in\emptyset$ 时

由定义 2 知:项集 c_i 为 x_i,其 $spt(c_i)=spt(x_i)$,若 $spt(x_i)<s$,则 $spt(c_i)<s$。

(2)$subsume(x_i)\notin\emptyset$ 时

由定理 3 知:$x_i \cup subsume(x_i) \subset c$,

$$spt(x_i)=spt(x_i \cup subsume(x_i)) \tag{3}$$

$$\forall c_i \in c, spt(c_i) \leqslant spt(x_i \cup subsume(x_i)) \tag{4}$$

若 $spt(x_i)<s$,由式(3)、式(4)得:$spt(c_i)<s$。

由定理 5 的分析知,随着时间的流失,若项目 x_i 的支持度从大于用户规定的最小支持度变化到小于用户规定的最小支持度,则在 HTC 中,由 x_i 所生成的闭频繁项集的支持度也会小于用户规定的最小支持度,因此,这些闭频繁项集都要被剪枝,本文称之为"动态不频繁项剪枝策略"(DNFIPS, dynamic non-frequent items pruned strategy)。

定理 6 在 A-NewMoment 算法流程中,$\forall x_i \notin T_d$(T_d 为将要删除的事务),$c=\{x_i \cup post\_set_s(x_i)\} \subset CFI$,若删除 T_d,则 $c \subset CFI$。

证明:$\because \forall x_i \notin T_d$,由 3.1.1 节的项目集表示知,在 T_d 的事务中,x_i 设置为 0。

$$\therefore x_i \cap \underset{x_i \notin T_d}{post\_set_s}(x_i)=0 \tag{5}$$

$\because c=\{x_i \cup post\_set_s(x_i)\} \subset CFI$

$\therefore T_d$ 被删除,由式(5)得:$c \subset CFI$。

由定理 6 的分析知,删除过时事务 T_d,若 $x_i \notin T_d$ 中的项目不对 x_i 生成的所有闭频繁项集进行重新搜索,则不会影响整个闭频繁项目集的正确输出,本文称之为"动态不搜索策略"(DNSS, dynamic non-search strategy)。通过上述分析,删除过期事务的过程如算法 2 所示。

算法 2 DeleTrans

```
输入:BV-DFIlist;
输出:HTC;
第 (tid mod N)位全置 0;
Foreach x_i∈T_d,i=1,2,…,p/*T_d 为过时事务
  Updata spt(x_i);
  If spt(x_i)≥s then
    从 HTC 中删除 x_i 的所有生成子;
    Search(x_i,post_set_s);
  Else
    从 HTC 中删除 s=spt(x_i)和 itemname=x_i 的所有生成子;
  Endif
Endfor
```

3.3.2 添加新事务

添加新事务是窗体滑动阶段的第二阶段,当事务流进当前窗口时,最新事务的每个项目置 1。新增的每个项目其支持度发生变化,已挖掘出来的闭频繁项集也会发生变化。添加新事务的过程如算法 3 所示。

算法 3 InseTrans

```
输入:BV-DFIlist;
输出:HTC;
Foreach x_i∈T_new,i=1,2,…,p/*T_new为新增事务
  第 (tid mod N)位置 1;
  Update spt(x_i);
Endfor
  Resort(DFIlist);
Foreach x_i∈T_new,i=1,2,…,p/*T_new为新增事务
  If spt(x_i)≥s then
    从 HTC 中删除 x_i 的所有生成子;
    Search(x_i,post_set_s);
  Endif
Endfor
```

4 算法比较

NewMoment 算法与 A-NewMoment 算法都由两部分构成:窗体初始阶段及窗体滑动阶段。假设频繁项目的总数为 $INum$,在窗体初始阶段,NewMoment 算法需要 $C_{INum}^{1}+C_{INum}^{2}+\cdots+C_{INum}^{INum}$ 个项目进行闭项集子集检测,设闭项集子集检测所需的平均时间为 $TCFI$,则 NewMoment 算法在窗体初始阶段的时间复杂度为:

$$\sum_{i=1}^{INum}(C_{INum}^{i}) \times TCFI \tag{6}$$

对于 A-NewMoment 算法,算法生成项目的包含索引时,需每个项目与其支持度后序进行二进制位与运算,其所耗费的时间为 $Tsubsume$;设算法产生的生成子为 $gen_1, gen_2, \cdots, gen_n$,这些生成子需进行保序判断的时间为 $TOgen_1, TOgen_2, \cdots, TOgen_p$,需进行包含索引子集判断的时间为 $TSgen_1, TSgen_2, \cdots, TSgen_m$,则生成子进行判断的时间为:

$$\sum_{i=1}^{p} gen_i \times Tgen_i + \sum_{j=1}^{m} gen_j \times Tgen_j + Tsubsume$$

由 A-NewMoment 算法知,$Tsunsume$ 远小于生成子进行

判断所耗费的时间，因此，$Tsunsume$ 可以忽略不计。则 A-NewMoment 算法的时间复杂度为：

$$\sum_{i=1}^{p} gen_i \times Tgen_i + \sum_{j=1}^{m} gen_j \times Tgen_j \quad (7)$$

由于 A-NewMoment 算法采用了剪枝策略，需进行生成子保序判断及包含索引子集判断的项目总数小于 NewMoment 算法中需进行闭项集子集检测的项目总数，且项目进行保序判断及包含索引子集判断所需的平均时间小于 TCFI，项目进行包含索引子集判断所需的时间远小于 TCFI。通过上述的分析，式(7)＜式(6)，即，在窗口初始阶段，A-NewMoment 算法的时间复杂度低于 NewMoment 算法的时间复杂度。

在窗体滑动阶段的第一阶段——过期事务删除过程，NewMoment 算法首先需要对每个删除项目进行枚举产生新的项目，然后，所产生的每个新项目进行闭项集子集检测，最后，根据闭项集子集检测判断结果对所产生的新项目在 HTC 表中做相应的更新、增加和删除闭频繁项集操作。同样，A-NewMoment 算法也要从 HTC 表中删除所有删除项目所产生的闭频繁项集及增加产生的新的闭频繁项集，由上述分析知，A-NewMoment 算法对每个删除项目所产生的生成子进行保序判断及包含索引子集判断所耗费的时间比 NewMoment 算法对每个删除项目所产生的新项目进行闭项集子集检测所耗费的时间少。因此，在窗体滑动阶段的过期事务删除过程，A-NewMoment 算法的时间复杂度低于 NewMoment 算法的时间复杂度。在窗体滑动阶段的第二阶段——新事务添加过程，NewMoment 算法与 A-NewMoment 算法对每个新增项目的处理与删除项目类似，不同点就是判断操作，根据过期事务删除操作过程分析知，在窗体滑动阶段的新事务添加过程中，A-NewMoment 算法的时间复杂度也低于 NewMoment 算法的时间复杂度。

5 实验结果及比较

为了评估算法的性能，所有的实验在 CPU 为 2.0GHZ 的 PentiumIV，内存为 1GMB，操作系统为 Windows XP 的 PC 机上进行，所有的实验程序均采用 C 语言编写，在 VC+6.0 环境中运行，实验中的模拟数据由 IBM 模拟数据产生器[12]产生，合成数据的主要参数为：T 表示事务的平均长度，I 表示频繁项集的平均长度，D 表示事务的数目，1k 代表 1000 条事务的记录，实验中的所有的项目数目固定在 1000。

算法性能的评估包括存储空间、数据流信息装载在初始窗口时算法的运行时间(以下简称算法的初始运行时间)和窗体滑动时算法的平均运行时间(算法的滑动运行时间)。算法在运行过程中，系统工具提供察看存储空间值的变化。窗体滑动时算法的平均运行时间是记录 100 个连续的窗体滑动时算法的平均运行时间。

5.1 算法的性能分析

实验分别使用短疏(T10I4D100K)、长疏(T20I5D100K)和长密(T30I20D100K) 3 套数据集来评估算法的性能，这 3 套数据集的最小支持度分别为 0.1%、0.4%和 0.6%。在实验中，窗口的尺寸大小从 10k 个事务变化到 100k 个事务，每变化一次窗口尺寸的大小，执行初始阶段数据流信息装载算法和滑动阶段事务删除、增加算法。图 1 说明，在多种数据类型环境中，随着事务数据流的增多，算法所消耗的存储空间及算法的初始运行时间几乎呈直线形状增大，但是，算法的滑动运行时间变化幅度不大。因此，从图 1 看出，A-NewMoment 算法的性能较稳定。

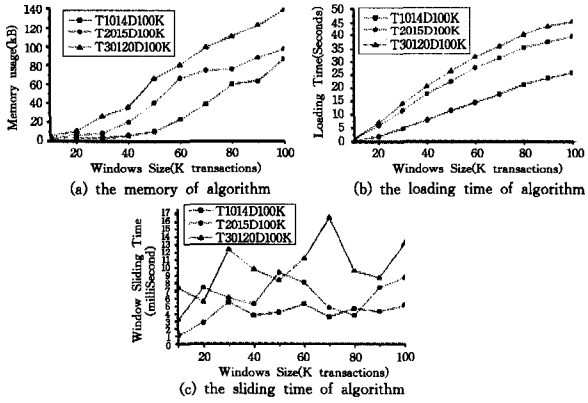

图 1 A-NewMoment 算法的性能.

5.2 算法的性能比较

不同支持度的比较如图 2 所示。实验是在滑动窗口大小尺寸为 100k 的条件下运行 A-NewMoment 与 NewMoment 算法，采用 T10I10D200K 数据集，评估 A-NewMoment 与 NewMoment 算法最小支持度从 0.1%变化到 1%的性能。

窗口尺寸大小不同的比较如图 3 所示。实验中，窗口尺寸大小从 10 个事务变化到 100k 个事务，每变化一次窗口尺寸的大小执行初始阶段数据流信息装载算法和滑动阶段事务删除、增加算法。实验采用 T10I10D200K 数据集，评估 A-NewMoment 与 NewMoment 算法最小支持度固定为 0.1%的性能。

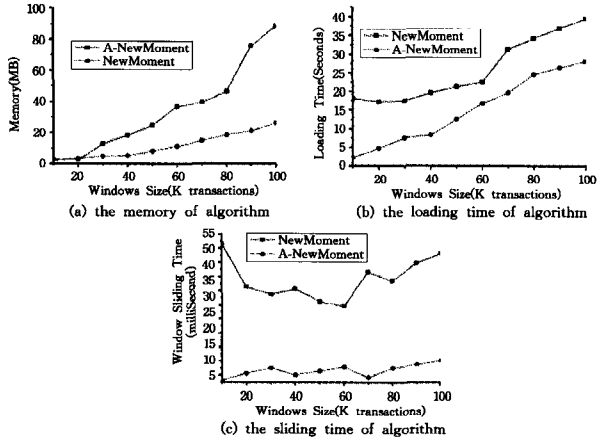

图 3 不同算法在不同窗口尺寸下的比较

图 2、图 3 说明，在同一数据集中不同支持度、不同窗口尺寸的情况下，运行 A-NewMoment 算法所需的存储空间比 NewMoment 算法所需的存储空间大，但是，A-NewMoment 算法的初始运行时间与滑动运行时间都比 NewMoment 算法的运行时间少。因此，从图 2、图 3 看出，A-NewMoment 算法与 NewMoment 算法相比，以增大存储空间为代价来换取较好的时间效率。

搜索空间大小的比较。实验中，初始窗口尺寸为 10k，连续滑动 10 个窗口运行 A-NewMoment 与 NewMoment 算法。实验采用长疏(T20I5D100K)和长密(T30I20D100K)2 套数

据集，最小支持度由 0.1%变化到 1%来评估 A-NewMoment 与 NewMoment 算法搜索空间的大小。表 3、表 4 说明，不管在窗体的初始状态还是滑动状态，A-NewMoment 算法的搜索空间都比 NewMoment 算法小，这是因为 A-NewMoment 算法采取了一些搜索策略，极大地减少了搜索空间。

表 3　T20I5D100K 数据集下的比较

algorithm	state of window	support threshold			
		0.1%	0.2%	0.5%	1.0%
A-NewMoment	initialization	1843267	937712	471628	283368
	sliding	286678	218432	106608	51438
NewMoment	initialization	9427678	76432328	3712378	1640276
	sliding	1123642	836608	432218	217402

表 4　T30I20D100K 数据集下的比较

algorithm	state of window	support threshold			
		0.1%	0.2%	0.5%	1.0%
A-NewMoment	initialization	11946487	9872431	4564872	1738237
	sliding	12768	10223	5624	2964
NewMoment	initialization	73755746	73210436	38659311	18433637
	sliding	81134	76236	37542	17464

结束语　由于数据流的高速、无限和不可预测的特点，传统的高效的闭频繁项集挖掘算法无法运用到数据流环境中，而目前经典的数据流闭频繁项集挖掘算法存在因搜索空间大而造成算法时间效率低的问题，针对此问题，本文提出了一种数据流中闭频繁项集挖掘算法 A-NewMoment。其在窗体初始阶段，按照频繁 1-项集的支持度从低到高进行排序，发现由频繁 1-项集所生成的支持度最大的最长闭频繁项集，采用“不需扩展策略”与“向下扩展策略”可避免生成大量中间结果，快速发现其余闭频繁项集。在窗体滑动阶段，提出“动态不频繁剪枝策略”从存储闭项集的 HTC 表中快速删除非闭频繁项集及“动态不搜索策略”动态地维护所有发生变化的闭频繁项集。

参 考 文 献

[1] Golab L, Ozsu M T. Issues in Data Stream Management[J]. ACM SIGMOD RECORD, 2003, 32(2): 5-14

[2] 陈辉. 数据流频繁模式挖掘及数据预测算法研究[D]. 武汉：华中科技大学，2008

[3] Chi, Y, Wang H, Yu P. MOMENT: maintaining closed frequent itemsets over a data stream sliding window[C]// Proc of the 2004 IEEE International Conference on Data Mining, 2004. LosAlamitos, USA: IEEE Computer Society, 2004: 59-66

[4] Jiang Nan, Gruenwald. CFI-stream: Mining closed frequent itemsets in data streams[C]// Proc of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2006. Philadelphia, USA: ACM publisher, 2006: 592-597

[5] Ranganath B N, Murty M N. Stream-Close: Fast Mining of Closed Frequent Itemsets in High Speed Data Streams[C]// Proc of the 2008 IEEE International Conference on Data Mining Workshops, 2008. Pisa, Italy: IEEE Computer Society, 2008: 516- 525

[6] Li Hua-fu, Ho Chi-chuan, Lee S-Y. Incremental updates of closed frequent itemsets over continuous data streams[J]. Exper Systems with Applications, 2009, 36(2): 2451-2458

[7] Yen S-J, Lee Y-S. An efficient algorithm for maintaining frequent closed itemsets over data stream[C]// Proc of the 22nd International Conference on Industrial, Engineering and Other Applications of Applied Intelligent Systems, 2009. Tainan, Taiwan: Springer-Verlag, 2009: 767-776

[8] Cheng J, Ke Yi-ping, Nq W. Maintaining frequent closed itemsets over a sliding window [J]. Journal of Intelligent Information Systems, 2008, 31(1): 191-215

[9] 熬富江. 数据流频繁模式挖掘关键算法及其仿真应用研究[D]. 长沙：国防科技大学，2008

[10] 宋威，杨炳儒，徐章艳. 一种改进的频繁闭项集挖掘算法[J]. 计算机研究与发展，2007，45(2)：278-286

[11] Lucchesc C, Orlando S, Perego R. Fast and memory efficient mining of frequent closed itemsets [J]. IEEE Trans on Knowledge and Data Engineering, 2006, 18(1): 21-36

[12] http://www.almaden.ibm.com

(上接第 199 页)

[5] Apache™. Hadoop: http://hadoop.apache.org/

[6] 陈勇. 基于 Hadoop 平台的通信数据分布式查询算法的设计与实现[M]. 北京：北京交通大学，2009

[7] Berliner H. The B* tree search algorithm: A best-first proof procedure[J]. Artificial Intelligence, 1979, 12: 23

[8] Wang Wei-yan, Wang Xiao-ling, Zhou Ao-ying. Hash-Search: An Efficient SLCA-Based Keyword Search Algorithm on XML Documents[J]. Lecture Notes in Computer Science, 2009, 5463: 496

[9] Byers J, Considine J. Simple Load Balancing for Distributed Hash Tables[J]. Lecture Notes in Computer Science, 2003, 2735: 80

[10] Litwin W, Neimat M A, Schneider D A. LH*-A Scalable Distributed Data Structure[J]. ACM Trans. on Database Systems, 1996, 21(4): 480

[11] Grover L K. Quantum Computers Can Search Arbitrarily Large Databases by a Single Query[J]. Phys. Rev. Lett., 1997, 79: 4709-4712

[12] Rowstron A, Druschel P. Pastry: Scalable, Decentralized Object Location, and Routing for Large-Scale Peer-to-Peer Systems[J]. Lecture Notes in Computer Science, 2001, 2218: 329

[13] 周可，王桦，李春花. 云存储技术及其应用[J]. 中兴通讯技术，2010，16(4)：24

[14] Sivathanu S, Liu Ling, Mei Yi-duo. Storage Management in Virtualized Cloud Environment[C]// IEEE 3rd International Conference on Cloud Computing. 2010: 204

[15] Chang F, Dean J, Ghemawat S, et al. Bigtable: A Distributed Storage System for Structured Data[J]. ACM Trans. Comp. Syst., 2008, 26(2): 4

[16] Dean J, Ghemawat S. MapReduce: Simplified Data Processing on Large Clusters[C]// Proc. 6th USENIX Symp. on Operating Syst. Design and Impl. 2004: 137-150

[17] Dean J. Experiences with MapReduce, an abstraction for large-scale computation[C]// Proc. 15th International Conference on Parallel Architectures and Compilation Techniques, 2006. Seattle, Washington, USA, ACM, 2006