事件序列上的频繁情节挖掘算法①

丁 勇, 王 云, 李 丛

(南京理工大学 泰州科技学院, 泰州 225300)

摘 要: 事件序列上的频繁情节挖掘是时序数据挖掘领域的热点之一, 基于非重叠发生的支持度定义, 提出一个频繁情节挖掘算法NONEPI++, 该算法首先通过情节串接产生候选情节, 然后通过预剪枝和计算情节发生的时间戳来产生频繁情节. 算法只需扫描事件序列一次, 大大提高了情节挖掘的效率. 实验证明, NONEPI++算法能有效地挖掘频繁情节.

关键词: 事件序列; 频繁情节; 非重叠发生

Algorithms for Mining Frequent Episodes on the Event Sequences

DING Yong, WANG Yun, LI Cong

(Taizhou College of Science and Technology, NJUST, Taizhou 225300, China)

Abstract: Mining frequent episodes on the event sequences is one of the hot areas of data mining. In this paper, support based on non-overlapped occurrence is definited. We propose an algorithm called NONEPI++ for mining frequent episodes, which firstly generate candidate episodes by join episodes, then generate frequent episodes by pre-pruning and timestamp calculating. The algorithm can improve the efficiency of mining episodes. Experiments show that NONEPI++ algorithm can effectively mine frequent episodes.

Key words: event sequence; frequent episode; non-overlapped occurrence

近年来, 在许多应用中出现了大量的有序事件, 如网络监控、入侵检测、股票交易、网络日志、基因序列等. 对些事件序列进行挖掘和分析, 可以发现事件类型之间的紧随关系, 从而揭示用户或系统潜在的行为模式, 这种模式被定义为“情节”[1]. 基于事件序列上的频繁情节挖掘已经成为时序数据挖掘领域的热点之一.

为了解决频繁情节挖掘问题, Manilla 等人[1]首先引入了情节的概念, 并提出了两个经典算法 WINEPI[2]、MINEPI[3]. 其中, WINEPI 基于滑动窗口来定义支持度, 而 MINEPI 则是基于情节的最小发生定义支持度. 这些算法在计算支持度时都可能包含了情节的多次重叠的发生, 从而导致情节发生的“过计数”问题. 为此 Laxman 等人[4]引入了“非重叠发生”的概念来计算一个情节的支持度, 并且提出了一个高效的频繁情节挖掘算法 NONEPI[5], 该算法通过采用有限状态机来跟踪情节的状态转移, 从而实现情节支持度的计算, 不足之处, 该算法与 Apriori 算法类似, 需要多遍扫描事件序列, 并产生大量候选情节.

为了克服NONEPI算法的不足, 本文提出一个事件序列上的非重叠发生的频繁情节挖掘算法NONEPI++, 该算法通过计算情节发生的时间戳来产生所有的频繁情节. 提高了频繁情节挖掘的效率.

1 相关概念

1.1 事件、事件序列

给定事件类型集 ε={E1,E2,…,En}, 事件就是一个二元组(E,T), 其中, E 表示某一个事件, T 表示事件的发生时间. 事件序列 S 是由 n 个事件按其发生的时间先后顺序排列而成的序列, 表示为 S=<(E1,T1,),(E2,T2),..,(En,Tn)>.

① 收稿时间:2014-04-01;收到修改稿时间:2014-05-04

1.2 情节、子情节

一个情节 α 表示为 α=<E1E2…Ek>, Ei∈ε, 且对于所有的 i 和 j(1≤i<j≤n)满足 Ei 总是在 Ej 之前发生. 情节 α 中事件的个数称为 α 的长度, 记为|α|.

给定情节 α=<E1E2…Em>和 β=<E1'E2'…Ek'>, 若至少存在一个整数序列 1≤γ1<γ2<…≤γk, 满足 Ei'=Eγi(1≤i≤k), 则称 β 是 α 的一个子情节, 记为 α□β

1.3 发生、非重叠发生

给定事件序列 S 和情节 α=<E1E2…Ek>, 若至少存在一个序列 S'=<(E1,T1),(E2,T2,…(Ek,Tk)>, 满足 Ti<Ti+1(1≤i<k), 且 S'是 S 的一个子序列, 称情节 α 在 S 上发生. 区间[Ti,Tk]称为 α 在 S 上的一次发生, 其中 Ti 和 Tk 分别为起始时间和终止时间. α 的所有非重叠发生组成的时间戳集合记为 α.NO. 设[Ts,Te]和[Ts',Te']是情节 α 在 S 上的两次发生, 若 Te< Ts', 则称[Ts,Te]和[Ts',Te']是 α 在 S 上的非重叠发生.

1.4 支持度、频繁情节

情节 α 的所有非重叠发生组成的最大集合的基数称为 α 的支持度, 记为 α.sup. 给定支持度阈值 min_sup, 若 min_sup≤α.sup, 则称 α 是一个频繁情节, 支持度为 k 的频繁情节记为 F-k 情节.

2 NONEPI++算法

2.1 算法描述

NONEPI++算法采用了与 Apriori 算法类似的广度优先搜索策略, 算法的基本思想是: 首先单遍扫描给定的事件序列 S, 从中发现所有的频繁 1 情节 F1, 并记录每个事件发生的时间戳. 然后, 由频繁 1 情节按照一定的规则串接生成候选 2 情节 C2, 通过 ComputerNO 过程计算 C2 中每个情节 α 的时间戳, 若时间戳集合的长度大于给定的阈值, 则该情节是频繁的, 所有满足 α.sup≥min_sup 的情节构成频繁 2 情节 F2. 以此类推, 由频繁 2 产生候选 3, 再计算时间戳和筛选, 产生频繁 3, 直到找不到任何频繁情节为止. 伪代码如下:

```
//定义时间戳及情节数据结构
Class Timestamp
{   long starttime //起始时间
    long endtime   //结束时间
}
Class Episode
{   String   name    //事件名称
    LinkedList<Timestamp>   NO //时间戳链表
}
算法: NONEPI++
输入: 事件序列 S=<(E1,T1,),(E2,T2),..,(En,Tn)>
最小支持度 min_sup
输出: 频繁情节 F
(1) 频繁情节集合 F=φ
(2) 扫描事件序列, 得到频繁 1 情节 F1 及时间戳
(3) F=F1
(4) While   F!= φ
(5)      F= getFrequent(F)
(6)      Output(F)

//由频繁 n 情节产生频繁 n+1 情节
Procedure   getFrequent(F)
输入: 频繁情节集合 Fn
输出: 频繁情节集合 Fn+1
(1) 频繁情节集合 F=φ
(2) For each α∈F
(3)     For each β∈F
(4)       If α.name 与 β.name 的前|α|-1 个字符相同
(5)           String name=α.name 的前|α|-1 个字符
                    +β.name 的最后一个字符
(6)           Episode γ=New Episode(name)
(7)           γ.NO=ComputerNO(α, β)
               If   |γ.NO|>=min_sup
(8)                 F.add(γ)
(9)   return F

//计算时间戳
Procedure   ComputeNO(α, β)
输入: 情节 α, 情节 β
输出: (α+β).NO
(1)   Timestamp i,j,k
(2)   For each i∈α.No
(3)       j=k
(4)       For each j∈β.No
(5)           If   i.Ts>(j-1).Te   and i.Te<j.TS
(6)                                                  and
(i+1).Ts>j.Ts
```

(7)　　add [i.Ts,j.TS] to (α+β).NO

(8)　　k=j+1

2.2 算法优化

根据情节的 Apriori 性质, 若一个情节是非频繁的, 则其任何超情节也是非频繁的. 在 getFrequent 过程的第(6)步后, 可预先测试给定情节 α 的子情节是否在频繁情节集合 F 中, 若不在则无需进行第(7)步操作, 从而避免了冗余的时间戳计算过程. 引入 HashMap<String,LinkedList>结构来存储所有的频繁情节, 其中 String 对应情节的串名, LinkedList 链表存放情节发生的时间戳.

```
//对候选情节剪枝
Procedure  Cut(Episode γ)
输入: 情节 γ, 频繁情节集合 F
输出: True/False
(1)  HashMap F
(2)  for each sub_γ∈γ
(3)    if  not F.contains(sub_γ.name)
(4)      return false
(5)  return true
```

2.3 算法复杂度分析

给定事件序列 S, 事件类型集合 ε, 频繁情节的集合|F|, 设每个情节至多有 max 个非重叠发生.

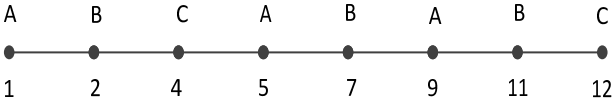
时间复杂度: 算法共产生|F|个频繁情节, 每次产生频繁情节需定位 HashMap, 上界为|ε|, 计算时间戳需单遍扫描 NO, 其上界为|S| , 所以时间复杂度为 O(|F|.|S|.|ε|).

空间复杂度: 共产生|F|个频繁情节, 每个情节至多有 max 个非重叠发生, 则空间复杂度为 O(|F|.max)

3 应用实例

给定事件序列 S1=

<(A,1),(B,2),(C,4),(A,5),(B,7),(A,9),(B,11)(C,12)>,



设最小支持度 min_sup=2, NONEPI++算法执行的步骤如下:

步骤一, 构造频繁情节 HashMap F =φ, 单遍扫描事件序列, 得到频繁 1 情节 F1={<A><B><C>}, 并将其添加到 F 中.

步骤二, 对 F1 中的两个情节按匹配规则进行串接, 形成候选 C2={<AA><AB><AC><BA><BB>

<BC><CA><CB><CC>}, 对每个候选情节 γ 分别查找其子情节 sub_γ 在 F 中是否存在? 存在, 所以计算 γ 的时间戳, 因 <AA><BB><CA><CB><CC> 的 |γ.NO|<min_sup, 将其从候选 C2 中剪枝, 最终形成频繁 F2={<AB><AC><BA><BC>}.

步骤三, 同理产生候选 C3={<ABC><BAC>}

因|<BAC>.NO|<min_sup, 所以从候选 C3 中剪枝, 形成 F3={<ABC>}. 当无任何候选产生时, 算法终止. 挖掘过程形成的频繁情节如表 1 所示.

表 1 频繁情节挖掘

序	频繁情	非重叠发生时间戳	支持
1	<A>	{[1,1][5,5][9,9]}	3
2	<B>	{[2,2][7,7][11,11]}	3
3	<C>	{[4,4][12,12]}	2
4	<AB>	{[1,2][5,7][9,11]}	3
5	<AC>	{[1,4][9,12]}	2
6	<BA>	{[2,5][7,9]}	2
7	<BC>	{[2,4][11,12]}	2
8	<ABC>	{[1,4][9,12]}	2

4 实验评估

实验采用的硬件环境为 Win7 操作系统、3.4GHZ Intel i3-4130 CPU、4GB 内存. 程序基于 Eclipse 平台和 Java 语言实现.

实验数据选择国内最具影响力的中国知网 CNKI 的数据, 下载南京理工大学泰州科技学院图书馆 Web 服务器上从 2014 年 1 月 1 日至 2014 年 1 月 31 日的日志数据, 该数据包括了读者对 8642 种不同参考文献的引用记录, 共 68413 条. 通过对不同的文献进行标号作为事件类型, 按照引用的时间顺序排列为一个事件序列.

图 1 显示随着支持度的增加, 挖掘的频繁情节个数减少. 但不同支持度阈值, NONEPI 和 NONEPI++挖掘的频繁情节个数是相同的.

图 2 显示 NONEPI++算法比 NONEPI 算法执行效率高. 当支持度较小时, 因为 NONEPI++算法需要进

行大量的时间戳计算, 所以相比 NONEPI 算法性能没有太大提升, 当支持度较大时, NONEPI++算法的执行效率有明显提升.

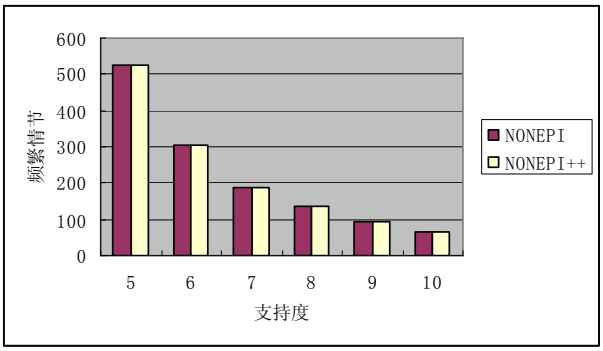


图 1 频繁情节个数和支持度阈值的关系

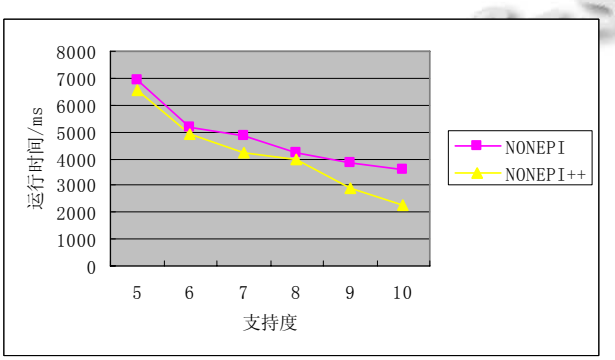


图 2 运行时间和支持度阈值的关系

5 结语

本文提出的 NONEPI++算法主要用于事件序列上非重叠发生的频繁情节挖掘, 算法通过情节的串接产生候选情节, 并根据情节的 Apriori 性质, 在计算情节发生的时间戳之前, 预先进行剪枝, 避免了冗余的计算过程, 提高了算法执行的效率. 理论和实验证明 NONEPI++算法能有效地挖掘事件序列上的非重叠发生的频繁情节.

参考文献

1 Manilla H, Toivonen H, Verkamo A. Discovering frequent episodes in sequences. Proc. of the First International Conference on Knowledge Discovery and Data Mining. 1995. 210−215.

2 Mannila H, Toivonen H, Verkamo AI. Discovery of frequent episodes in event sequences. Data Mining and Knowledge Discovery, 1997, 1(3): 259−289.

3 Mannila H, Toivonen H. Discovering generalized episodes using minimal occurrences. KDD, 1996, 96: 146−151.

4 Laxman S, Sastry PS, Unnikrishnan KP. Discovering frequent episodes and learning hidden markov models: A formal connection. IEEE Trans. on Knowledge and Data Engineering, 2005, 17(11): 1505−1517.

5 朱辉生,汪卫,施伯乐.基于频繁闭情节及其生成子的无冗余情节规则抽取.计算机学报,2012,1:53−63.

6 朱辉生,汪卫,施伯乐.基于最小且非重叠发生的频繁闭情节挖掘.计算机研究与发展,2013,50(4):852−860.

7 Zhu H, Wang P, He X, et al. Efficient episode mining with minimal and non-overlapping occurrences. 2010 IEEE 10th International Conference on Data Mining (ICDM). 2010. 1211−1216.

8 林树宽,乔建忠.一种基于情节矩阵和频繁情节树的情节挖掘方法.控制与决策,2013,28(3):339−344.