文章编号: 1001-0920 (2013) 03-0339-06

# 一种基于情节矩阵和频繁情节树的情节挖掘方法

林树宽, 乔建忠

(东北大学 信息科学与工程学院, 沈阳 110819)

**摘 要:** 针对现有的最小发生的频繁情节挖掘中存在的问题, 提出一种发现情节的不同最小发生并对其进行计数的方法. 在此基础上, 提出基于情节矩阵和频繁情节树的最小发生频繁情节挖掘方法, 基于直接扩展思想, 只需扫描数据一次, 不需生成候选情节, 提高了挖掘的时空效率. 提出了基于相同结点链和哈希链的优化方法, 通过省略相同结点的扩展过程, 进一步提高了挖掘性能. 最后, 在不同类型的真实数据集上进行实验, 实验结果验证了所提出的频繁情节挖掘方法的优势以及优化方法的有效性和高效性.

**关键词:** 频繁情节; 情节矩阵; 频繁情节树; 相同结点链

**中图分类号:** TP391 **文献标志码:** A

# An episode mining method based on episode matrix and frequent episode tree

*LIN Shu-kuan*, *QIAO Jian-zhong*

(College of Information Science and Engineering, Northeastern University, Shenyang 110819, China. Correspondent: LIN Shu-kuan, E-mail: linshukuan@ise.neu.edu.cn)

**Abstract:** For the problems existing in current frequent episode mining with minimal occurrences, the paper proposes a method of discovering distinct minimal occurrences and counting them. On this basis, an episode matrix and frequent episode tree based mining method is proposed, which only scans event sequences once based on the idea of direct extension without candidate generation and enhances space-time efficiency. Moreover, its optimization is presented based on same node chains and hash chains, which improves the mining performance further by omitting the extension process of same nodes. A series of experiments on different types of real data sets show the advantage of the proposed method, and validate the effectiveness and efficiency of the optimization method.

**Key words:** frequent episode; episode matrix; frequent episode tree; same node chain

## 0 引 言

随着传感器和射频识别 (RFID) 等电子数据采集设备 (EDGE) 在供应链管理[1]、环境监控[2]等诸多领域中的广泛使用, 有大量事件类型的数据产生, 这些事件类型的数据需要被及时地分析和处理, 以便得出管理者需要的有价值的信息. 因此, 复杂事件处理 (CEP) 成为新的研究热点[3-4]. 频繁情节挖掘是复杂事件处理的重要研究内容. 通过发现事件序列中的频繁情节, 可以建立相应的情节规则, 并对未来事件进行预测, 以便在事件发生前及早采取行动.

所谓情节是事件序列上发生事件的偏序集合. 事件序列上的频繁情节挖掘不同于事务数据库上的频繁模式挖掘[5], 也不同于序列数据库上的序列模式挖掘[6]. 为了建立情节规则, 本文对最小发生的频繁情节 (MinEpi) 进行挖掘.

**定义 1** 给定串行情节 $\mathrm{EP} = e_1e_2\cdots e_n$, 对于它的发生 $\mathrm{ep} = (e_1,t_1)(e_2,t_2)\cdots(e_n,t_n)$(其中 $t_1 < t_2 < \cdots < t_n$, 满足全序关系), 如果不存在任何 EP 的发生 $\mathrm{ep}' = (e_1,t_1')(e_2,t_2')\cdots(e_n,t_n')$ 满足 $t_1' \geqslant t_1$, $t_n \geqslant t_n'$, 且 $t_n' - t_1' < t_n - t_1$, 则称 ep 为情节 EP 的最小发生.

一个情节在同一个时间间隔内可能存在多个最小发生. 例如, 在序列"$a_1b_2b_3c_4$"(下标数字是事件发生的时间戳)中, 对于情节"$abc$", 在间隔 [1,4] 中有 $(a,1)(b,2)(c,4)$ 和 $(a,1)(b,3)(c,4)$ 两个最小发生. 为了避免情节规则建立中的冗余,需要从中确定该

**收稿日期:** 2011-12-20; **修回日期:** 2012-08-21.

**基金项目:** 国家自然科学基金项目(61272177); 中央高校基础科研基金项目(N110604001).

**作者简介:** 林树宽(1966–), 女, 教授, 从事时态数据挖掘、机器学习等研究; 乔建忠(1964–), 男, 教授, 博士生导师, 从事并行计算、人工智能等研究.

情节的唯一一次最小发生[7]. 为此, 本文提出“不同最小发生”的概念来解决这一问题.

**定义 2** 对于情节EP $= e_1e_2\cdots e_n$的两个最小发生$\mathrm{mo} = (e_1,t_1)(e_2,t_2)\cdots(e_n,t_n)$和$\mathrm{mo}' = (e_1,t_1')(e_2,t_2')\cdots(e_n,t_n')$, 如果对任意的$i\in[1,n]$都有$t_i\neq t_i'$, 则称这两个最小发生为不同最小发生.

本文将在事件序列上挖掘不同最小发生的频繁情节(DMinEpi).

目前, 已有一些用来挖掘和处理MinEpi的方法, 文献[8]最早给出了MinEpi挖掘的框架; 文献[9]对情节中的事件进行了扩展, 除了包含事件类型和时间戳, 还包含了其他一些事件属性, 在情节描述中包含了对这些扩展属性的约束; 文献[7]侧重于最小发生的情节匹配、检测以及基于情节规则对未来事件的预测; 文献[10]对最小发生的闭频繁情节进行挖掘. 这些方法均采用频繁模式挖掘中Apriori算法[5]的挖掘思想(本文称为Apriori类方法), 需要经过由低阶频繁情节产生高阶候选情节, 再通过对候选情节进行评价产生高阶频繁情节的反复迭代过程(以下称为“候选-评价过程”), 因此挖掘的时间和空间代价较大. 不仅如此, 这些方法需扫描数据多次, 只能局限于处理静态的事件序列, 不能被扩展用来处理事件流数据. 除此之外, 这些方法均未考虑不同最小发生的问题.

针对上述问题, 本文提出了基于情节矩阵和频繁情节树的DMinEpi挖掘方法, 该方法采用与已有方法完全不同的思路, 通过对低阶频繁情节逐级扩展, 直接生成全部频繁情节, 只需扫描数据一次, 不需要生成候选情节集合.

## 1 基于情节矩阵和频繁情节树的情节挖掘

针对现有的MinEpi挖掘中存在的问题, 本文提出了基于情节矩阵和频繁情节树的DMinEpi挖掘方法. 其基本思想是: 通过建立情节矩阵生成所有的频繁2-情节, 在此基础上, 对2-情节进行基于时间队列的逐级扩展, 直接生成频繁$n$-情节$(n>2)$, 不需生成候选情节集合, 且只需对数据扫描一次.

### 1.1 挖掘频繁2-情节

为了生成任意长度的DMinEpi, 首先从频繁2-情节开始挖掘, 为此, 本文设计了一个情节矩阵识别2-情节的最小发生, 并记录所有2-情节的最小发生信息. 为了描述情节矩阵, 先给出时间队列的定义.

**定义 3** 给定一个事件序列中的情节EP $= e_1e_2\cdots e_n$, 其$m$个不同最小发生为$\mathrm{mo}_i = (e_1,t_1^i)(e_2,t_2^i)\cdots(e_n,t_n^i)$, $i=1,2,\cdots,m$, 则由所有发生的首事件的时间戳构成的队列$(t_1^1,t_1^2,\cdots,t_1^m)$称为该情节的时间队列,记为TQ.

给定一个包含$n$个事件类型$e_1,e_2,\cdots,e_n$的事件序列, 其情节矩阵是一个$n*n$矩阵,其行和列分别代表$n$个事件类型, 每个矩阵元素$[e_i,e_j]=(\mathrm{count},\mathrm{TQ})$. 其中: count是2-情节EP $= e_ie_j$的最小发生计数, TQ是情节EP的时间队列. 在此称$e_i$为矩阵元素$[e_i,e_j]$的左事件, $e_j$为元素$[e_i,e_j]$的右事件. 每个矩阵元素都被初始化为$[e_i,e_j]=(0,\mathrm{null})$, $i,j=1,2,\cdots,n$.

在扫描事件序列的同时, 构建情节矩阵. 后发生的事件将与先前发生的事件构成2-情节的发生. 如果该发生是最小发生且在时间窗口约束内, 则将在以先发生的事件为左事件, 后发生的事件为右事件的矩阵元素中记录其最小发生信息, 即将该元素的计数加一, 其左事件的时间戳被插入到该元素的时间队列尾. 这里的关键问题是如何确定两个事件构成的2-情节的发生是否为最小发生. 为了解决这个问题, 本文在每个矩阵元素上设置一个锁, 每个锁都被初始化为“加锁”状态. 基于“锁”的情节矩阵生成过程如下:

**算法 1** 情节矩阵生成算法(GenMat).

输入: 事件序列$S$, 事件序列包含的事件类型数量$n$, 时间窗口长度$l$;

输出: 情节矩阵.

```
index ← 0;        //扫描索引初始化
while (事件序列 S 未扫描完)
  根据索引 index 从 S 中读取当前事件 (e,t);
  for(i = 0; i < n; i++) do
   if(([e_i,e].lock == 0)&&(t − t_i < l)) then
     [e_i,e].count++;
     将时间戳 t_i 插入到元素 [e_i,e] 的时间队列尾;
     [e_i,e].lock ← 1;        // 加锁
   end if
  end for
  for(j = 0; j < n; j++) do
   if([e,e_j].lock == 1) then
     [e,e_j].lock ← 0;        // 解锁
     设置事件 (e,t) 为元素 [e,e_j] 的最后左事件;
   end if
  end for
  index++;
end while
```

可以证明, 算法1只对2-情节的最小发生进行计数, 即算法1是正确的; 算法1对所有2-情节的最小发生都进行了计数, 即算法1是完备的. 由于篇幅限制, 这里省略证明过程.

### 1.2 生成扩展情节的不同最小发生计数、建立频繁情节树

在情节矩阵中可以发现, 如果频繁2-情节中, $[e_i,e_j].\text{count} \geqslant \min\_\text{sup}$, 则2-情节$e_ie_j$将是频繁的. 进一步考虑以$e_i$为右事件的矩阵元素, 如果存在元素$[e_k,e_i]$满足$[e_k,e_i].\text{count} \geqslant \min_\text{sup}$, 则将2-情节$e_ke_i$与$e_ie_j$通过事件$e_i$连接起来, 得到3-情节$e_ke_ie_j$. 进一步考察情节$e_ke_ie_j$的频繁性, 这将依赖于它的不同最小发生计数. 类似地, 如果情节$e_ke_ie_j$是频繁的, 则满足$[e_m,e_k].\text{count} \geqslant \min_\text{sup}$的矩阵元素$[e_m,e_k]$将被考虑, 由情节$e_me_k$与$e_ke_ie_j$连接成的情节$e_me_ke_ie_j$将被考察是否频繁, 这也依赖于它的不同最小发生计数, 以此类推. 这样, 只需扫描事件序列一次, 就可以由频繁2-情节直接扩展生成任意长度的频繁情节, 不需生成候选情节集合. 这里的关键问题是如何确定经连接扩展生成的情节的不同最小发生计数. 本文提出了基于扩展时间戳的计数方法.

**定义 4**　给定$n$-情节$\text{EP}_1 = e_1e_2\cdots e_n (n \geqslant 2)$和2-情节$\text{EP}_2 = ee_1$, 它们的时间队列分别为$\text{TQ}_1 = (t_1^1,t_2^1,\cdots,t_{k1}^1)$和$\text{TQ}_2 = (t_1^2,t_2^2,\cdots,t_{k2}^2)$, 则将情节$\text{EP}_1$和$\text{EP}_2$通过事件$e_1$连接起来形成的$n+1$-情节$\text{EP} = ee_1e_2\cdots e_n$称为情节$\text{EP}_1$和$\text{EP}_2$的扩展情节, $\text{EP}_1$和$\text{EP}_2$称为情节EP的源情节.

**定义 5**　给定$n$-情节$\text{EP}_1 = e_1e_2\cdots e_n (n \geqslant 2)$和2-情节$\text{EP}_2 = ee_1$, 它们的时间队列分别为$\text{TQ}_1 = (t_1^1,t_2^1,\cdots,t_{k_1}^1)$和$\text{TQ}_2 = (t_1^2,t_2^2,\cdots,t_{k_2}^2)$, 对于$\text{TQ}_1$中任意相邻的时间戳对$(t_i^1,t_{i+1}^1)$, $k_1-1 \geqslant i \geqslant 0$, $t_0^1 = 0$, 如果在$\text{TQ}_2$中存在$k$个时间戳$t_{jl}^2 (k \geqslant l \geqslant 1, k \geqslant 1)$满足$t_{i+1}^1 > t_{jl}^2 > t_i^1$, 则称$t_{jm}^2 = \max(t_{jl}^2)(k \geqslant l \geqslant 1)$为扩展情节$\text{EP} = ee_1e_2\cdots e_n$在$t_i^1$和$t_{i+1}^1$之间的扩展时间戳. 对于$\text{TQ}_1$中每一时间戳对$(t_i^1,t_{i+1}^1)$, 扩展时间戳可以按升序形成一个队列, 称为扩展情节EP的扩展时间队列.

**定理 1**　给定$n$-情节$\text{EP}_1 = e_1e_2\cdots e_n (n \geqslant 2)$和2-情节$\text{EP}_2 = ee_1$, 其时间队列分别为$\text{TQ}_1 = (t_1^1,t_2^1,\cdots,t_{k1}^1)$, $\text{TQ}_2 = (t_1^2,t_2^2,\cdots,t_{k2}^2)$. 对于$\text{TQ}_1$中任意相邻的时间戳对$(t_i^1,t_{i+1}^1)$, $k_1-1 \geqslant i \geqslant 0$, 如果扩展情节$\text{EP} = ee_1e_2\cdots e_n$存在扩展时间戳, 则相应于时间戳对$(t_i^1,t_{i+1}^1)$, 有且只有一次扩展情节EP的不同最小发生计数产生.

设情节$\text{EP}_1$和$\text{EP}_2$在$t_i^1$和$t_{i+1}^1$之间的扩展时间戳为$t_{jm}^2$, $\text{EP}_1$相应于$t_{i+1}^1$的最小发生为$\text{mo}_1 = (e_1, t_{i+1}^1)(e_2,t_{i2}^1)\cdots(e_n,t_{in}^1)$, $\text{EP}_2$相应于$t_{jm}^2$的最小发生为$\text{mo}_2 = (e,t_{jm}^2)(e_1,t_j^2)$. 可以证明$\text{ep} = (e,t_{jm}^2)(e_1, t_{i+1}^1)(e_2,t_{i2}^1)\cdots(e_n,t_{in}^1)$一定是扩展情节EP的最小发生, 且只产生一次不同最小发生计数. 由于篇幅限制, 这里省略证明过程.

定理1为不同最小发生计数奠定了理论基础. 扩展计数和扩展时间队列的生成过程如下:

**算法 2**　生成扩展计数和扩展时间队列(GetExtendedCount).

输入: 情节$\text{EP}_1 = e_1e_2\cdots e_n$的时间队列$\text{TQ}_1 = (t_1^1,t_2^1,\cdots,t_{k1}^1)$, $\text{EP}_2 = ee_1$的时间队列$\text{TQ}_2 = (t_1^2,t_2^2,\cdots,t_{k2}^2)$;

输出: 扩展情节$\text{EP} = ee_1e_2\cdots e_n$的扩展计数extended_count和扩展时间队列ETQ.

```
extended_count ← 0; i ← 0; j ← 0; ETQ ←null;
repeat
  while (t_i^2 ⩾ t_j^1) j++;
  end while
  extended_count++;
  i++;
  while(t_i^2 < t_j^1) i++;
  end while
  将 t_{i-1}^2 插入扩展时间队列ETQ;
until ((i > k_2)||(j = k_1))
return extended_count.
```

在算法2的基础上, 算法3建立了频繁情节树, 即频繁情节挖掘的过程.

**算法 3**　频繁情节树的建立(BuiltTree).

输入: 情节矩阵;

输出: 频繁情节树.

```
procedure GenChild(fe)
对于矩阵中所有的事件类型 e
 if ([e, fe].count ⩾ min_sup) then
  call GetExtendedCount(fe.tq, [e, fe].tq);
  if (extended_count ⩾ min_sup) then
    建立 fe 的一个子结点 ce;
    ce.name ← e;
    ce.count ← extended_count;
    ce.tq ← extended_queue;
    fe ← ce;
    call GenChild(fe);
  end if
 end if
procedure BuiltTree
对于每个频繁的事件实例 fe
      将 fe 插入频繁情节树中作为根的子结点;
对于情节矩阵中所有的事件类型 e
      if ([e, fe].count ⩾ min_sup) then
```

建立 fe 的子结点 ce;
ce.name ← $e$;
ce.count ← $[e, \text{fe}]$.count;
ce.tq ← $[e, \text{fe}]$.queue;
fe ← ce;
call GenChild(fe); //扩展生成 fe 的子结点
end if

由算法 2 和算法 3 可以看出, 扩展情节的扩展计数不大于其源情节的发生计数, 因此, 若一个扩展情节是频繁的, 则它的两个源情节一定是频繁的. 故算法 3 的情节树扩展过程可挖掘出所有的频繁情节. 在这一过程中, 只扫描数据一次, 无需生成候选情节集合, 也无需经过反复的候选-评价过程.

**例 1** 对于图 1 给出的事件序列, 如果支持数阈值 $\min\_\sup = 2$, 则相应的频繁情节树如图 2 所示.

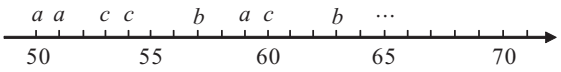


**图 1 事件序列**

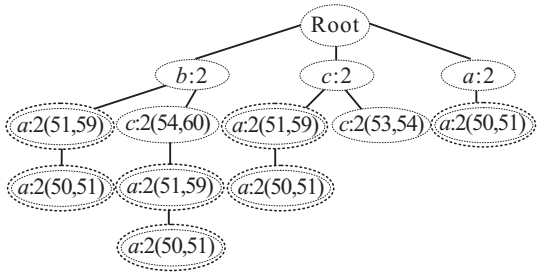


**图 2 频繁情节树**

## 2 基于相同结点链和哈希链的优化

为了进一步提高频繁情节挖掘的效率, 节省内存空间, 本文提出了上述挖掘过程的优化方法.

**定义 6** 设 $\text{node}_1, \text{node}_2, \cdots, \text{node}_n (n > 1)$ 是频繁情节树的结点, 它们的事件类型和时间队列分别为 $e_1, e_2, \cdots, e_n$ 和 $\text{TQ}_1, \text{TQ}_2, \cdots, \text{TQ}_n$. 如果对于 $\forall i, j \in [1, n]$, 均有 $e_i = e_j$ 且 $\text{TQ}_i = \text{TQ}_j$, 则称 $\text{node}_1, \text{node}_2, \cdots, \text{node}_n$ 为相同结点.

在图 2 所示的频繁情节树中, 双线椭圆表示的结点为相同结点.

**观察 1** 频繁情节树中的相同结点具有相同的子结点.

按照观察 1, 相同结点无需重复扩展, 它们的子结点只需被扩展生成一次, 且子结点信息只需在树中存储一次, 即可被所有的相同结点共享. 为此, 需将所有的相同结点用链连接起来 (在此称为相同结点链). 本文将所有结点中的时间队列进行了编码, 每个时间队列编码为其所有时间戳之和, 并在其上作用一个哈希函数

$$\text{hash}(\text{tq}) = \left(\sum_{i=1}^{n} t_i\right) \bmod 100, \tag{1}$$

其中时间队列 $\text{tq} = (t_1, t_2, \cdots, t_n)$. 具有相同哈希值的结点信息连在同一个哈希链中, 分别作为相应的相同结点链的链首. 这样, 在频繁情节树中建立两种类型的链, 一种是相同结点链, 另一种是哈希链, 如图 3 所示. 树中的结点只存储事件类型、发生计数和编码, 时间队列不再存储于树中, 而被存储在哈希链中, 且相同结点的时间队列只在哈希链中存储一次. 通常, 频繁情节树中会有很多相同结点 (图 6 表示的相关实验说明了这一点), 因此, 在树中建立相同结点链和哈希链将节省大量的挖掘时间和空间.

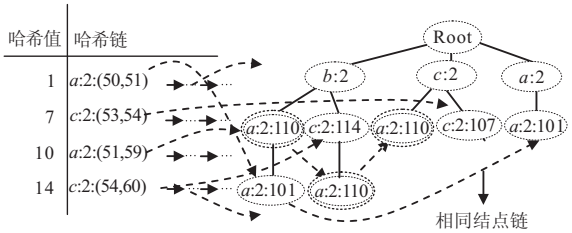


**图 3 哈希链和相同结点链**

## 3 实验及性能评价

为了说明所提出的 DMinEpi 挖掘方法的性能以及优化效果, 本文在真实数据集上进行了一系列实验, 将所提出的基于情节矩阵和频繁情节树的挖掘方法 (基本方法) 与 Apriori 类挖掘方法[8]在时间和空间成本上进行了比较, 同时也验证了基于相同结点链和哈希链对挖掘过程进一步优化 (优化方法) 的效果.

所有实验都在 2.66 GHz CPU, 3.5 GB 内存的 PC 机上完成, 采用两个不同类型的真实数据集. 第 1 个数据集来自人类基因组数据[11], 包括 5 个事件类型, 是一个稠密数据集; 第 2 个数据集来自 Intel 伯克利实验室布置的 54 个无线传感器[12], 每个事件包含传感器编号、时间戳、温度等属性. 本文选择了其中的 27 个传感器, 并将温度属性值分为 4 个区间, 形成了 108 个事件类型. 由于包含较多的事件类型, 在给定的时间窗口内, 情节的支持数较少, 因此这是一个稀疏数据集.

图 4 给出了两个数据集上 3 种方法随不同的频繁度阈值变化的挖掘时间, 两个实验中事件序列长度均取 10 K. 由图 4 可以看出, 稠密和稀疏数据集上的实验结果有所不同. 在稠密数据中, 基本方法的挖掘时间较 Apriori 类方法具有明显的优势, 而在稀疏数据中, 基本方法的挖掘时间多于 Apriori 类方法. 两个数据集上优化方法挖掘的时间效率较基本方法均有较大幅度的提高, 且均高于 Apriori 类方法. 这是由于

优化方法继承了基本方法中无需生成候选情节的优点, 并通过建立相同结点链和哈希链省略了很多相同结点的扩展过程, 相同结点数量越多, 其优化效果就越明显. 同时还可以发现, 无论在稠密数据还是在稀疏数据上, 优化方法的挖掘时间对于频繁度阈值的变化都是不敏感的, 始终花费较少时间, 尤其当频繁度阈值较小时, 挖掘时间变化不大, 这种结果在目前的频繁情节挖掘中尚未见到(一般地, 当频繁度阈值较小时, 挖掘时间会显著增加). 究其原因, 是建树过程中建立的相同结点链产生了这种效果. 当设置较小的频繁度阈值时, 虽然产生的频繁情节较多, 但相同结点的数量也明显增多, 它们被链在各自的相同结点链中, 只被扩展一次. 这样, 通过建立相同结点链, 大量相同结点将共享相同的子结点, 这使得大量相同结点的扩展过程被省略, 相同结点数量越多, 节省的时间就越多. 因此, 无论取多小的频繁度阈值、产生多少频繁情节, 需要进一步扩展生成子结点的结点数量基本上是稳定的, 优化方法的挖掘时间受频繁度阈值的影响较小.

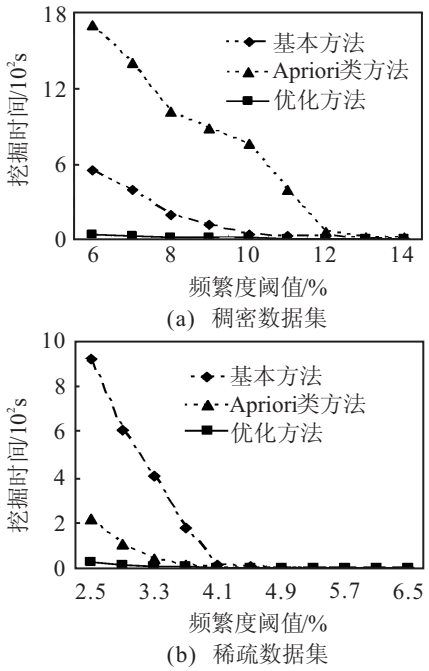


**图 4　两个数据集上的挖掘时间比较**

优化的效果还可以从图 5 中看到. 在该实验中, 尽管设置了一个较大的频繁度阈值(12%), 还是可以看出优化的效果, 且随着事件序列长度的增加, 优化的效果越来越好. 如果设置一个较小的频繁度阈值, 优化的效果会更加明显, 图 4 中当频繁度阈值较小时, 可以看出这种效果.

从图 4 还可以发现, 当频繁度阈值取值很大时(对于稠密数据, min_fre ⩾ 13%, 对于稀疏数据, min_fre ⩾ 4.9%), 3 种方法的挖掘时间比较接近, 且挖掘时间接近于零. 此时, 几乎没有多少频繁情节生成, 因此 3 种方法的差距并不明显, 但在实际挖掘过程中, 取过大的频繁度阈值是没有意义的.

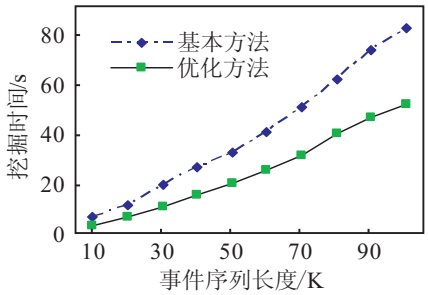


**图 5　频繁度阈值为12%时的优化效果(稠密数据集)**

图 6 给出了两个数据集上相同结点数量随频繁度阈值的变化情况. 可以看出, 随着频繁度阈值的降低, 相同结点的数量迅速增加. 通过建立相同结点链, 将节省大量的挖掘时间. 因此, 如前文所述, 即使取较小的频繁度阈值, 挖掘时间也不会明显增加, 相同结点数量越多, 优化效果越好.

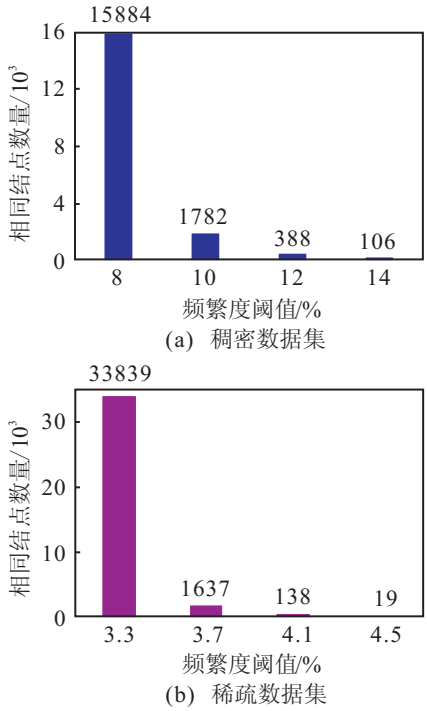


**图 6　两个数据集上不同频繁度阈值下的相同结点数量**

图 7 分别在两个数据集上对 3 种挖掘方法所占用的内存空间进行了比较. 由图 7 可以看出, 无论是在稠密集还是稀疏集中, 优化方法的空间占用较基本方法明显减少, 且都少于 Apriori 类方法. 这是因为在优化方法中, 对时间队列进行了编码, 树中的结点只存储事件类型、计数和编码这样的简单信息, 使得空间占用大大减少; 此外, 优化方法通过建立相同结点链和哈希链, 使同一相同结点链上所有相同结点的信息(包括时间队列)在相应的哈希链中只存储一次, 节省了大量的内存空间, 且频繁度阈值越小, 相同结点数量越多, 节省的空间就越多. 因此, 优化方法的空间

成本并不随频繁度阈值的减小而明显增加, 其空间性能优于基本方法和 Apriori 类方法.

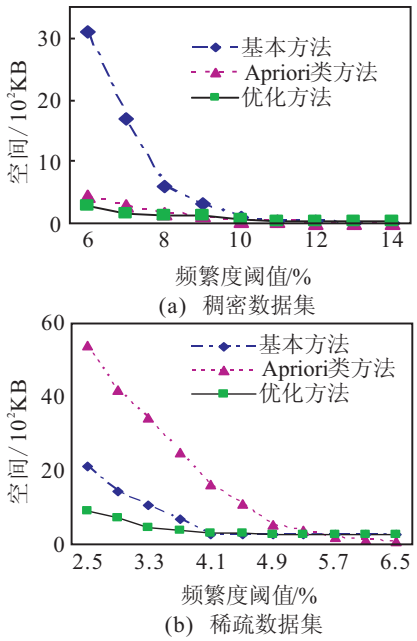


**图 7 两个数据集上空间代价的比较**

## 4 结 论

本文研究了不同最小发生频繁情节 (DMinEpi) 的挖掘方法. 现有的 MinEpi 挖掘并未考虑不同最小发生的情况, 且需多次扫描数据和反复进行候选-评价并生成大量候选集, 导致这些方法不能处理事件流数据, 且时间和空间代价较大. 针对这些问题, 本文提出了一种基于情节矩阵和频繁情节树的挖掘方法, 该方法与现有方法的挖掘思路完全不同, 通过对低阶频繁情节进行逐级的扩展直接生成高阶频繁情节, 在此过程中, 只需扫描数据一次, 不需生成候选情节集合, 也不需要反复进行候选-评价过程, 在稠密数据集和稀疏数据集上分别显示出较好的时间和空间性能. 为了进一步减少挖掘时间, 节省内存空间, 增强所提出方法在不同类型数据上的时间、空间性能, 本文又进一步提出了基于相同结点链和哈希链的优化方法. 无论在稠密集还是在稀疏集上, 相比于基本方法和 Apriori 类方法, 其时间和空间性能均具有明显的优势, 且具有时间和空间成本不随频繁度阈值明显变化的良好特性. 真实数据集上的实验表明了本文所提出的挖掘方法的优势以及优化方法的有效性和时间、空间上的高效性.

## 参考文献(References)

[1] Lee C H, Chung C W. Efficient storage scheme and query processing for supply chain management using RFID[C]. Proc of the ACM Sigmod Conf on Management of Data. Vancouver: ACM Press, 2008: 291-302.

[2] Chandy K M, Aydemir B E, Karpilovsky E M, et al. Event webs for crisis management[C]. Proc of the Int Association of Science and Technology for Development. Anaheim: Acta Press, 2003: 1020-1025.

[3] Wu E, Diao Y, Rizvi S. High-performance complex event processing over streams[C]. Proc of the ACM Sigmod Conf on Management of Data. Vancouver: ACM Press, 2006: 407-418.

[4] Demers A, Gehrke J, Panda B, et al. Cayuga: A general purpose event monitoring system[C]. Proc of the 3rd Biennial Conf on Innovative Data Systems Research. New York: ACM Press, 2007: 412-422.

[5] Agrawal R, Imielinski T, Swami A. Mining association rules between sets of items in large databases[C]. Proc of the ACM Sigmod Conf on Management of Data. New York: ACM Press, 1993: 207-216.

[6] Agrawal R, Srikant R. Mining sequential pattern[C]. Proc of the 11th Int Conf on Data Engineering. Dallas: IEEE Computer Society, 1995: 3-14.

[7] Cho C W, Zheng Y, Wu Y H, et al. A tree based approach for event prediction using episode rules over event streams[C]. Proc of the Database and Expert Systems Application. Heidelberg: Springer-Verlag, 2008: 225-240.

[8] Mannila H, Toivonen H, Verkamo I. Discovery of frequent episodes in event sequences[J]. Data Mining and Knowledge Discovery, 1997, 1(3): 259-289.

[9] Mannila H, Toivonen H. Discovering generalized episodes using minimal occurrences[C]. Proc of the 2nd Int Conf on Knowledge Discovery and Data Mining. Menlo Park: AAAI Press, 1996: 146-151.

[10] Zhou W Z, Liu H Y, Cheng H. Mining closed episodes from event sequences efficiently[C]. Proc of Pacific-Asia Conf on Knowledge Discovery and Data Mining. Heidelberg: Springer-Verlag, 2010: 310-318.

[11] The UCSC genome bioinformatics site[EB/OL]. (2010-1-1). http://hgdownload.cse.ucsc.edu/downloads.html.

[12] Intel Lab Data Site[EB/OL]. (2010-1-1). http://db.csail.mit.edu/ labdata/labdata.html.