

工学硕士学位论文

基于时间间隔的事件序列频繁模式挖掘算
法研究

**RESEARCH ON ALGORITHM OF MINING
FREQUENT PATTERNS BASED ON TEMPORAL
INTERVALS**

刘亚男

哈尔滨工业大学

2011 年 12 月

国内图书分类号：TP391.3

学校代码：10213

国际图书分类号：621.3

密级：公开

工学硕士学位论文

基于时间间隔的事件序列频繁模式挖掘算法研究

硕士研究生：刘亚男

导师：张春慨 副教授

申请学位：工学硕士

学科：计算机科学与技术

所在单位：深圳研究生院

答辩日期：2011年12月

授予学位单位：哈尔滨工业大学

Classified Index: TP391.3

U.D.C: 621.3

Thesis for the Master Degree of Engineering

**RESEARCH ON ALGORITHM OF MINING
FREQUENT PATTERNS BASED ON TEMPORAL
INTERVALS**

Candidate:	Yanan Liu
Supervisor:	Associate Prof. Chunkai Zhang
Academic Degree Applied for:	Master of Engineering
Specialty:	Computer Science & Technology
Affiliation:	Shenzhen Graduate School
Date of Defence:	Dec, 2011
Degree-Conferring-Institution:	Harbin Institute of Technology

摘 要

现存的序列模式挖掘算法多是基于瞬时事件的,然而在现实世界中很多事件都是发生在一段时间内,例如语言分析,网络检测等,时间间隔事件序列频繁模式挖掘在这些领域都有很重要的应用。本文的主要研究内容正是带有时间间隔的事件序列频繁模式挖掘。

与传统的序列模式挖掘不同的是时间间隔事件之间的关系是很复杂的,这也正是这种序列频繁模式挖掘的难点。到目前为止多数文章的关系定义都是基于 Allen 关系定义,本文中的事件关系定义也是基于 Allen 关系定义,并在此基础上进行了去噪声处理,使之更适应现实场景。另外本文简单的描述了影响频繁模式生成的各种兴趣度衡量,并且基于现实情况,本文采用支持度为兴趣度衡量。

而本文最主要的贡献是本文提出了一种基于“候选频繁模式生成—支持度计算”的高效算法,并且在两个阶段都提出了改进策略。首先在候选频繁模式生成阶段不同于传统的方法中利用 k 层频繁模式与 k 层频繁模式生成 $k+1$ 层候选频繁模式集,本文提出用 k 层频繁模式与 2 层频繁模式构成 $k+1$ 层候选频繁模式集,这样就能减少在合并两个频繁模式时的关于两个模式中间部分是否相等的比较次数,这种策略在提高算法效率的同时也能够减少冗余候选频繁模式的生成。其次,本文的算法维护一个 2-频繁模式集合,利用一定的策略来尽可能的减小用于合并生成候选集的 2-频繁模式集,使得产生尽量少的候选频繁模式,提高算法的效率。

在支持度计算阶段本文同样提出了两种改进策略。首先本文在构造候选频繁模式集的同时构造了索引,指向需要遍历的客户序列,这能够有效的减小算法的搜索空间,提高算法效率。其次不同于传统的挖掘算法中在计算支持度的时候多次遍历数据库,本文提出了一种算法,当计算具有相同长度的候选频繁模式的支持度时,只需遍历一次数据库。总之本文在支持度计算的过程中一方面减少遍历数据库的次数,另一方面减少遍历数据库时的搜索空间。

最后,本文提出了仿真数据的生成。在此基础上本文进行了两个方面的实验,一是本文提出了几个重要的参数对算法的影响,并且在实验中验证了提出的理论。二是为了证明本文提出的算法的有效性,本文就算法的效率以及正确率两个方面与枚举树算法和索引集算法进行了对比。

关键字: 候选频繁模式; 频繁模式; 时间间隔事件; 模式挖掘

Abstract

Existing temporal pattern mining assumes that events do not have any duration. However, events in many real world applications have durations, such as linguistic analysis and network detection areas. Mining frequent patterns among interval-based events has very important applications in these areas and it's the most important research content this thesis focuses on.

Different from traditional sequential pattern mining, relationships among interval-based events are very complex which is the difficulty in interval-based events sequential pattern mining area. So far, in most papers the relationship is defined based on Allen relationships definition. But this kind of relationship definition confine to the relationship between two events. The relationship definition in this thesis is also based on Allen relationships definition, and the thesis also proposes the appropriate improvement. Besides, this thesis deals with the noise and makes it suitable to the algorithm in this thesis. In addition to relationships definition, this thesis also describes several interestingness measures that influence the generating of frequent patterns and this thesis use minimum support as interestingness measures.

Based on the relationships definition described above, this thesis proposes an algorithm based on "Candidate Generation-Support Counting" model and gives improvement strategies in the two stages respectively. In "Candidate Generation" stage, different from traditional method that generating level- k candidates from two $k-1$ frequent temporal patterns, this thesis proposes a method that generating level- k candidates from one $k-1$ frequent temporal pattern and one 2 frequent temporal pattern. Doing than can reduce times of comparing between items that belong to two patterns. So it contributes to improving efficiency of the algorithm and also reducing the number of invalid patterns. At the same time, this thesis maintains a set of 2 frequent patterns which are used to combine candidates and use some strategy to reduce this set. So it contributes to reducing the number of invalid patterns.

This thesis also proposed two strategies to improve algorithm in "Support Counting" stage. Firstly, this thesis generates index which point to customer sequences need to be scanned when generating candidates set. This can effectively reduce the search space of algorithm. Secondly, different to traditional method, this thesis proposes a method to scan database only once to count support of all k -patterns. In short, in support counting procedure, the algorithm reduces times of scanning database and also reduces the search space.

At last, this thesis generates simulation dates, and then does experiments in two aspects. Firstly, the thesis proposes theory about the influence of several important

parameters on the algorithm. Secondly, in this thesis, the efficiency and the correct rate of the algorithm are compared to enumeration tree algorithm and indexing set algorithm.

Keywords: candidates, frequent pattern, interval-based event, pattern mining

目 录

摘要	I
Abstract.....	II
目录	IV
第 1 章 绪 论.....	1
1.1 研究背景及意义	1
1.2 课题相关的研究现状	2
1.2.1 传统事件序列挖掘算法	3
1.2.2 基于数据库转换算法	4
1.2.3 基于树状结构的算法	5
1.2.4 基于序列模式挖掘算法转换的算法	7
1.3 本文研究内容及结构安排	8
第 2 章 基于时间间隔事件频繁模式挖掘.....	10
2.1 引言	10
2.2 事件序列及模式定义	10
2.2.1 事件定义	10
2.2.2 事件序列定义	10
2.2.3 数据库定义	11
2.3 事件关系	11
2.3.1 事件关系定义	11
2.3.2 关系表达方法	13
2.3.3 事件关系的现实约束	14
2.4 带有时间间隔事件序列频繁模式挖掘	16
2.4.1 时间间隔事件序列频繁模式	16
2.4.2 时间间隔事件关联规则挖掘	17
2.5 本章小结	19
第 3 章 频繁模式候选集生成.....	20
3.1 引言	20
3.2 现有方法总结	20
3.3 改进的频繁模式候选集生成方法	21
3.3.1 二层频繁模式生成	22
3.3.2 改进的算法思想	23
3.3.3 剪枝策略	24

3.3.4 算法实现	25
3.4 本章小结	26
第 4 章 支持度统计	27
4.1 引言	27
4.2 现有方法总结	27
4.3 改进的支持度计算方法	28
4.3.1 数据库查找范围	28
4.3.2 支持度计算	29
4.4 本章小结	30
4.4.1 算法流程	31
4.4.2 算法实现	31
第 5 章 实验及结果分析	33
5.1 引言	33
5.2 数据生成	33
5.2.1 主要参数	33
5.2.2 数据生成过程	34
5.3 算法效率验证	35
5.3.1 参数对算法效率的影响	35
5.3.2 算法效率对比实验	37
5.4 算法正确性验证	38
5.4.1 支持度对正确率的影响	39
5.4.2 事件类型对正确率的影响	39
5.5 本章小结	40
结论	41
参考文献	43
哈尔滨工业大学学位论文原创性声明及使用授权说明	47
致谢	48

第1章 绪 论

1.1 研究背景及意义

数据挖掘在很多领域都得到了广泛的应用,例如市场分析,入侵检测和企业管理等。其中一个主要的组成部分就是序列模式挖掘,它起源很早,并且应用广泛。

序列模式挖掘的概念最早是由 Agrawal 和 Srikant^[1]提出的。他们假设了存在一个客户交易的数据库,每条交易记录包括客户 ID,交易时间和客户在交易中购买的商品的集合。目的是在数据库中挖掘出那些出现频繁的模式。自那以后,越来越多的人关注序列模式挖掘。挖掘的模式也越来越多样化,包括最大频繁模式挖掘^[1]、连续频繁模式挖掘^[2]、多维频繁模式挖掘^[3]等。

在过去的十年里,人们已经对序列模式挖掘有了较深的研究,即从一些瞬时事件序列中挖掘出符合用户限制的模式。虽然在这一领域中已经有了很多成熟的算法^[1,4,5,6],但是在现实生活中大多数事件并不是瞬时的,是发生在一段时间内的,即有开始时间和结束时间并且存入数据库中。此外,不同的事件可能是同时发生的,序列模式并不能很好的表达这种情形,因此基于间隔事件的时间模式挖掘是很有必要的。

在现实生活中,带有时间段的事件处处可见,这也正是挖掘带有时间间隔事件频繁模式的理论依据。例如在零售业、医疗、生物基因研究、网络检测等领域中,挖掘出这样的频繁模式都将具有指导作用。

在零售行业,例如超市的管理者知道在面包促销的开始和结束时间内,顾客会高频的购买黄油。这就会使管理者制定更好的营销策略,如在面包促销阶段将面包和黄油摆放在一起,并且适时的订货。另外通过这种挖掘还能使管理者了解到顾客的兴趣、习惯及需要,这将更有利于超市的管理。

在医疗行业,例如对一些糖尿病患者来说,利用带有时间间隔的事件序列频繁模式挖掘可以看出高血糖症状和糖尿症状是高频并发的,这对有效的测试糖尿病患者起到了积极作用。再如,对于登革热患者来说,知道患者开始发烧的第三天血小板数量开始减少,这对于患者病况的掌握很有意义。

在生物基因研究上^[7],例如, DNA 是一串核苷酸的序列,序列中的每一项是{A,C,G,T}中的一个,代表四种核苷酸。任意长度大于 4 的一串核苷酸被称作一个序列。高频出现的核苷酸和核苷酸序列称作 poly-regions,用它可以典型的代

表 DNA。而对于这些 poly-regions 互相高频覆盖的检测能够引导生物学家更好的研究不同的 DNA 以及这些具有高频覆盖 poly-regions 的 DNA 对生物进化的影响。在这里 poly-regions 可以被看作是一个带有时间间隔的事件，而起始时间和结束时间可以用表示核苷酸的字母表示。

在网络检测领域中依然可以用到这种时间间隔模式挖掘，可以分析包和路由日志记录。图 1-1 中是一个小型局域网的简图，图中灰色长方形代表主机，他们通过 Router1 与 Router2 连接。若将两个主机相连看作一个事件，那么源地址发出请求为事件的开始时间，连接建立为事件的结束时间，这就是一个完整的时间间隔事件。那么在这个局域网中会发生很多这样的事件，并且都会保存记录。利用这份记录我们可以通过带有时间间隔的事件频繁模式挖掘算法挖掘出这一网络中频繁发生事件的规律及网络的一般模式，为我们做预测及入侵检测提供依据。

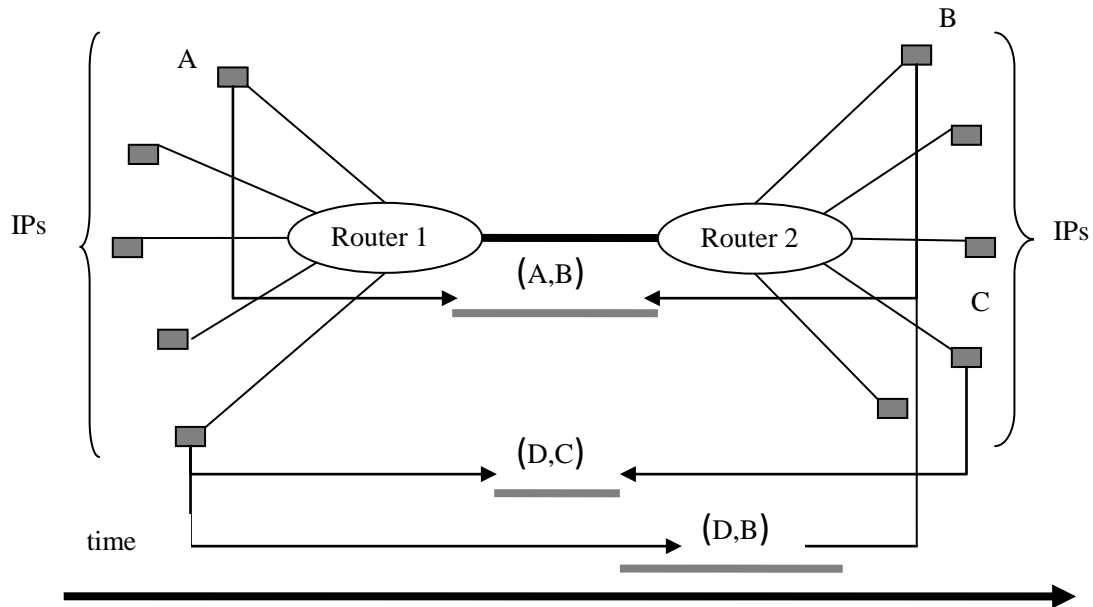


图 1-1 网络示例

1.2 课题相关的研究现状

带有时间间隔的事件序列频繁模式挖掘是个新兴领域。和传统的序列模式挖掘最大的不同是时间间隔事件挖掘复杂事件关系的处理上，即同样的事件间有不同的关系，从而构造成了不同的频繁模式。但是大多算法都是基于传统的序列频繁模式挖掘算法形成的。目前的成熟算法比较少。在现有的成熟算法中可以按照算法的基本思想将其分为三类，分别是基于数据库转换的算法、基于树的算法和基于模式前缀的算法。

1.2.1 传统事件序列挖掘算法

序列模式挖掘,可看作瞬时事件序列频繁模式挖掘,这一领域中已经有了很成熟的理论基础,很多人将序列模式挖掘理论应用于带有时间间隔的事件序列挖掘领域中。鉴于此,本文对序列模式挖掘做了简单的介绍。

总体来说,序列模式挖掘算法可以分为两种模式,第一种是“生成—检测”模式,文献[1, 8, 9]介绍了几种此类型中比较经典的算法。本文以文献[9]中的 GSP (Generalized Sequential Pattern) 算法为例进行说明。另一种类型是“横向的频繁模式增长”模式,同样的也有文献[6,10,11]对这种模式进行了介绍,本文以文献[6]中介绍的 PrefixSpan (Prefix-projected Sequential pattern mining) 为例进行介绍。

(1) **GSP 算法** GSP 算法需要多次遍历数据库,第一次遍历时得到所有的 1-频繁模式,将这些 1-频繁模式与自身相结合构成 2-频繁模式候选集,这时再遍历数据库统计候选集中的模式的支持度得到所有 2-频繁模式。再将 2-频繁模式与自身相结合得到 3-频繁模式候选集,如此迭代,直至挖掘出所有的频繁模式。GSP 算法的主要过程可总结如下:

a) 候选集生成:有长度为 $k-1$ 的频繁模式 F_{k-1} ,下一个循环过程的候选集由 F_{k-1} 与它自身结合而成。然后为了能够快速统计支持度将候选集存入到哈希树中,再利用支持度的单调性进行剪枝,将那些子模式不是频繁模式的模式剪掉。

b) 支持度计算:再遍历数据库,对数据库中的每个序列 ζ ,在哈希树中查找 ζ 的所有子序列,相应的支持度加 1。

GSP 虽然有很好的剪枝策略,但是显而易见的,该算法要多次查询数据库例如若所挖掘的频繁模式最长为 k ,那么该算法要至少查询数据库 k 次。因此 GSP 算法是不适合较长频繁模式挖掘的。

(2) **PrefixSpan 算法** PrefixSpan^[6]是基于模式增长模型的算法。这种算法的主要思想是通过序列前缀的概念将数据库划分,在每个子数据库中独立挖掘序列频繁模式,最后再将频繁模式组合。主要算法过程如下:

a) 遍历数据库一次的到所有的 1-频繁模式。

b) 分割搜索空间:以上一步得到的 1-频繁模式作为序列前缀。若一个序列数据库具有 k 个 1-频繁模式,那么数据库就被分解成 k 个小的搜索空间。

c) 挖掘频繁模式:因为上一步中形成的搜索空间是互不干扰的,在这一步中在每个搜索空间中递归的增长频繁模式长度,最后得到所有的频繁模式集合。

文献[6]中同时还介绍了两种优化策略 bi-level projection 和 pseudo projection。

虽然 PrefixSpan 能够利用分割数据库再逐个空间挖掘的策略高效的挖掘频繁模式，但是由于要分割数据库，在硬盘输入输出上还是消耗很大的。

前面的两种算法都是基于横向数据库（即以客户序列为主体）的算法，此外还有算法^[12,13]将横向数据库转化成纵向（即以模式为主体）进行数据挖掘。在这样的算法中为每一模式分配一个 id-list，这个表中的内容为包含该模式客户序列号及相应的时间标签。比较典型的算法是文献[12]中介绍的 SPADE(Sequential Pattern Discovery using Equivalence classes)。

(3) **SPADE 算法** 文献[12]中介绍的 SPADE 算法主要有以下三个特点：

a) 用 id-list 来纵向表示数据库，对每个模式来说都有一个 id-list。而所有频繁模式都能通过 id-list 连接来得到。

b) 算法将原始的搜索空间分化成很多小的空间，每一个空间都能单独在存储器上进行运算。

c) 针对不同的空间可以应用不同的搜索策略，例如广度优先或深度优先。以便快速的得到所有的频繁模式。

SPADE 算法能够通过三次遍历数据库得到所有的频繁模式，但是现实中绝大多数数据库都是横向的，而将横向数据库转换成 SPADE 适用的以 id-list 表示的纵向数据库会带来额外的时间消耗。

1.2.2 基于数据库转换算法

基于数据库转换的算法主要方式是经过数据库转换，即将一个事件的开始时间和结束时间作为两个事件来处理，再配以特定的算法，挖掘出频繁模式，再还原到我们想要的带有时间段的事件频繁模式。

目前利用这种基本思想的算法比较少，文献[14]中就介绍了这样一种方法。首先将数据库中的时间序列转换成一个个按照发生时间排序的序列。例如数据库中有 A(2,17), B(3,7), C(7,14), B(13,19)，转换后就变成


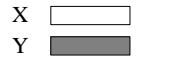




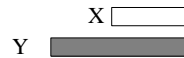
$$a^+ < b^+ < b^- = c^+ < b^{2+} < c^- < a^- < b^{2-}$$

并且将想要挖掘的事件关系转换成相应的序列，如表 1-1（表中所示的关系定义将在下一章进行详细的介绍）所示。

再将序列模式挖掘算法 PrefixSpan^[9]进行相应的转换，让它能够适应带有时间间隔事件频繁模式挖掘的特性。

这种方法的难点主要在于两个方面，一是要考虑事件关系的转换，即转换成瞬时事件序列的同时不能失去时间间隔事件序列的特性。二是衍生出的算法必须与前面所述的关系表达方式相结合，而这正是这一领域的难点。

表 1-1 数据库转换算法举例

XY关系	YX关系	关系图示	关系表达式
X before Y	Y after X		$X^+ < X^- < Y^+ < Y^-$
X equal Y	Y equal X		$X^+ = Y^+ < X^- = Y^-$
X meets Y	Y met-by X		$X^+ < Y^+ = X^- < Y^-$
X overlaps Y	Y overlapped-by X		$X^+ < Y^+ < X^- < Y^-$
X during Y	Y contains X		$Y^+ < X^+ < X^- < Y^-$
X starts Y	Y started-by X		$X^+ = Y^+ < X^- < Y^-$
X finishes Y	Y finished-by X		$Y^+ < X^+ < X^- = Y^-$

另外需要说明的是，因为是将一件时间间隔事件当作两件瞬时事件处理，所以无论进行怎样的转换，转换后的数据库都将增大一倍。例如对于一个有 k 个事件的序列来说，每个时间间隔时间经转换后都变成了 2 个瞬时事件，序列的长度变为 $2k$ 。结果将产生多达 2^{2k} 个不同的序列模式，这样就导致了无论用什么样的算法都极大的增加了算法的搜索空间，这是算法复杂度的主要来源。

1.2.3 基于树状结构的算法

基于树状结构的算法是这一新兴领域中比较常用的算法，在文献[12,15,16]中详细描述枚举树算法正是这一类型中的经典算法。以这种思想为基础的算法主要有广度优先遍历算法 BFS，深度优先遍历算法 DFS，以及两种相结合的算法 Hybrid DFS。下面就对这种思想进行简单的介绍。

枚举树的主要思想就是将以树的数据结构存储所有形式的模式作为候选频繁模式，上述中所谓的不同方法即是这个树的不同遍历顺序。算法按照遍历顺序计算该候选频繁模式的支持度，将不满足支持度的剪枝，满足的留下。最后留在树上的频繁模式即是我们想要的频繁模式。

无论是哪种遍历方式都采用如图 1-2 所示的数据结构。树的根节点为空，第 k 层列出了长度为 $k-1$ 的时间间隔事件序列完全排列。例如数据库中共有 3 个事件 A, B, C，在树的第三层就是这三种事件的所有排列。如图中所示，从第三层开始，每层衍生出两层，蓝色的一层是以父亲为前缀产生的候选频繁模式，黄色的一层是父亲的可能关系。

枚举树算法中事件关系沿用 Allen 在文献[17]中提出的 12 种关系定义，可简化为七种：b (Before)，m (Meet)，o (Overlap)，c (Contain)，e (Equal)，f (Finish)

和 s(Start)。本文也将沿用这种关系定义，将在第二章详细说明。树中的事件关系是通过指针表达的，使三种以上事件序列的关系指向 2-频繁模式。例如图中的 $A||B^*A \rightarrow C^*BC$ 表示 ABC 这 3-频繁模式的关系为 A (Equals) B, A (Before) C, B (Meets) C，这就是黄色一层的意义。

每种遍历方法都有它的优缺点。在广度优先算法中，采用的是自顶向下的搜索策略，对于树结构的一层来说，算法首先遍历这层的所有节点，在遍历完全后才会遍历下一层。

因为广度优先遍历的特性，算法并不能够快速的得到长度较长的频繁模式。而深度优先遍历算法能够弥补这一缺点。深度优先遍历能够较快的得到长度较长得频繁模式。最重要的是筛选频繁模式的“支持度”具有向下闭合特性，即具有半单调性，根据 Apriori 原理，一个模式是频繁的，那么它的所有子集都是频繁的。那么在深度优先遍历 (DFS) 中，当发现一个候选模式不满足频繁模式的条件，可以将这一枝都剪掉，避免不必要的遍历，提高算法的效率。这是广度优先遍历办不到的。

即使如此，DFS 依然有它的不足。最明显的是如前面所述，枚举树算法利用指针来定义关系。如果采用 DFS 算法，当遍历到一个长度大于等于三的候选频繁模式时，关系无法通过指针来辨别。因此算法将重新遍历数据库得到相应的 2-频繁模式，这样将大大增加算法时间开销。因此为了克服这一缺点又产生了 Hybrid DFS 算法。

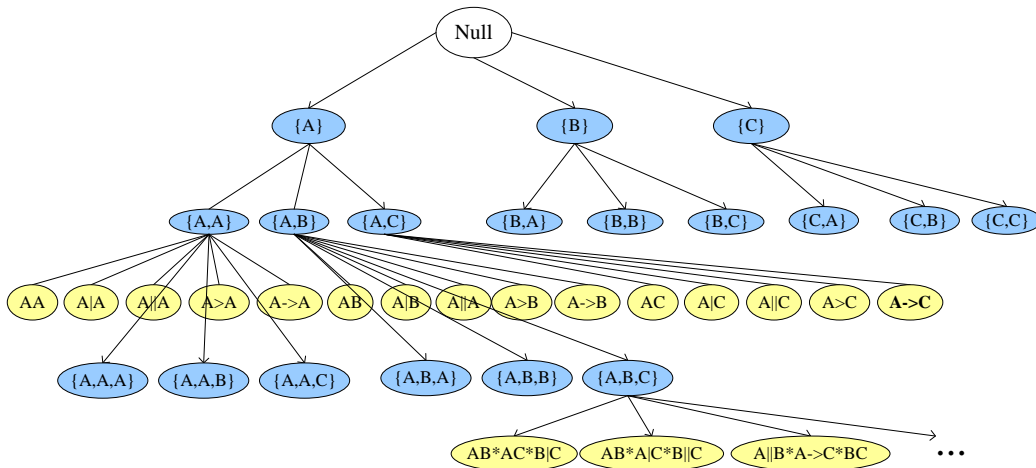


图 1-2 枚举树举例

Hybrid DFS 算法融合了 BFS 与 DFS。首先遍历广度优先遍历得出所有的 2-频繁模式，将这些 2-频繁模式存储，然后再进行深度优先遍历，这样的做法有效的避免了多次扫描数据库。

基于枚举树的算法是当今这一领域中比较盛行的算法，但是它的存储结构复

杂，若是没有合适的剪枝策略复杂度也是较高的。

1.2.4 基于序列模式挖掘算法转换的算法

在带有时间间隔的事件序列频繁模式挖掘领域中，现存的算法中大多数都是从序列模式挖掘算法转换过来的，其中一个比较高效的算法是基于索引集的算法^[18]。这种算法的前身是序列模式挖掘算法 MEMISP (Memory Indexing for Sequential Pattern mining) 算法^[19]。

同样的，这种算法也有自己运用的关系表达方法—矩阵。这是在这个领域中较为常见的一种表达方式，例如前文中提到的频繁模式 $\{A, B, C\}$ 可以表示为如式 1-1 形式，矩阵中的小写字母代表前文叙述的几种关系。

$$\begin{pmatrix} A & B & C \\ = & e & b \\ * & = & m \\ * & * & = \end{pmatrix} \quad (1-1)$$

算法的主要过程为构造索引集→挖掘频繁模式→构造索引集…，算法的主要数据结构如图 1-4 所示。在这个例子中右侧为数据库，每一条都是一个客户的记录。左侧表格为单项频繁模式“A”的索引集。索引集结构包括三个属性，第一列为该项在一条客户记录中出现的位置；第二列为该时间间隔；第三列为指向出现这一频繁模式的客户序列指针。

算法首先遍历数据库得到 1-频繁模式，在构造 1-频繁模式索引集，即图 1-3 中的左侧表格。接下来根据索引集遍历索引集中指针指向的客户序列，从索引集中记录的位置后开始查询，再得到 2-频繁模式。如此迭代，知道得到所有的频繁模式。

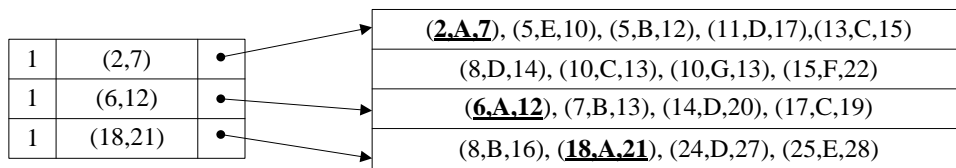


图 1-3 索引集算法举例

索引集算法沿用瞬时事件序列频繁模式挖掘算法，突出的有点是记录了指针，这样在遍历数据库的时候可以直接去掉那些对频繁模式没有贡献的序列，能够很好的缩小算法搜索空间，弥补了上述基于枚举树算法的不足。从效率上来说索引集算法是比枚举树算法优的。但是，这种算法的缺点也是比较明显的。首先，这种算法存在一个严重的缺陷，将导致得不到正确的结果。这种算法在一条记录中存在相同的频繁模式的情况下容易遗失频繁模式。例如：一个数据库中有两条记

录，分别是 $\{A(2,9), A(13,25), B(20,28)\}$ 和 $\{A(2,9), B(8,13)\}$ ，如图 1-4 所示

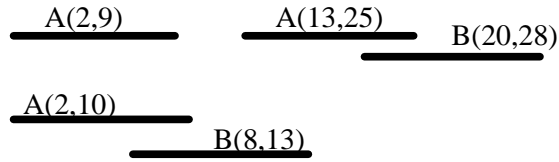


图 1-4 索引集算法缺点举例

假设最小支持度为 100%。在这种情况下，A-索引集中只有 A(2,9)（因为索引集中只记录第一次出现的），在挖掘候选频繁模式的时候从第一条记录中只会挖掘出模式 A(Before)B，而漏掉 A(Overlap)B，使得它不能满足最小支持度，被抛弃。而实际上 A(Overlap)B 是频繁模式。

第二，在构造索引集和形成频繁模式的时候，两次遍历索引表，这样大大增加了时间复杂度。所以一个效率较高且能得到完全的频繁模式的算法是迫切需要的。

1.3 本文研究内容及结构安排

本文的主要内容是提出一个高效且高正确率的带有时间间隔事件序列频繁模式挖掘算法，该算法是基于“候选频繁模式生成—支持度计算”模型的，本文分别针对这两个部分进行了研究：

(1) 带有时间间隔的事件间的关系。前文中论述了 Allen 关系定义的局限性。本文结合本文算法的特性选择了适应本身的关系定义方法。并且对影响频繁模式以及关联规则的兴趣度量做了一定研究以及说明了本文采用支持度为兴趣度量的理由。

(2) 生成频繁模式候选集。这是本文算法的一个重要部分，本文采用“候选频繁模式生成—支持度计算 2”模型算法，在候选频繁模式生成部分不同于传统的算法的结合方式，本文提出了新的候选频繁模式生成方法，并且在此基础上提出了适合本文算法的剪枝策略。

(3) 支持度计算。这是本文算法第二个重要的组成部分。在频繁模式候选集已经生成的基础上，需要计算候选频繁模式的支持度以确定频繁模式。不同于传统的需要多次遍历数据库的情况，在本文中提出了一种对于具有相同的长度的候选频繁模式只需遍历一次数据库的算法。

(4) 系统生成。在上述内容的基础上，生成一个系统来实现上述的算法。本文首先产生仿真数据集，并基于该数据集运行本文提出的算法及另外两种传统算法，在算法效率及正确率两个方面针对不同的参数影响进行了对比。

在前文中本文已经详细介绍了课题的研究背景，及课题的国内外研究现状。

接下来的几章主要对课题的理论部分进行详细的论述。本文结构安排如下：

第二章：这一章是本文的理论基础，在这一章中本文首先介绍了时间间隔事件序列频繁模式挖掘的相关概念。然后对这种特殊模式中的多个事件间的关系作了研究并且确定了与本文算法相合的关系定义方式。在这一章的最后介绍了各种兴趣度量对频繁模式以及关联规则生成的影响。

第三章：在这一章中介绍了改进算法的第一部分，即频繁模式候选集的生成。在这一章中首先介绍了几个比较典型的候选频繁模式生成的方法，然后本章描述了本文提出的改进的频繁模式候选集的生成算法并且给出了剪枝策略。

第四章：在这一章中介绍了改进算法的第二部分，即支持度计算方法。同第三章相同的是本章首先介绍了几种比较经典支持度计算方法。然后提出了改进的支持度计算方法。

第五章：在这一章中实现了本文的实验系统，首先介绍了仿真数据集的生成方法，然后在此基础上验证了各种参数对算法的影响，并且针对改进的算法在算法效率以及正确率两个方面的优越性进行了实验证明。

第2章 基于时间间隔事件频繁模式挖掘

2.1 引言

这一章主要介绍带有时间间隔事件序列频繁模式挖掘的相关概念，作为后续理论的理论基础。

首先本章将对事件以及带有时间间隔的事件序列进行定义，其次在 Allen 事件关系定义的基础上论述多个事件间关系，然后本章将论述不同的兴趣度对频繁模式挖掘的影响及对频繁模式进行定义，最后将在上述的内容基础上简单的论述关联规则的产生。

2.2 事件序列及模式定义

这一节中将对本文研究的基本内容时间间隔事件序列及时间间隔事件频繁模式进行定义。

2.2.1 事件定义

一个事件定义为 E ，它是一个三元组 (e, t_{start}, t_{end}) 。第一个元素为事件的标签， t_{start} 为事件的开始时间， t_{end} 为事件的结束时间。例如一个病人开始发烧是 11 日，病愈为 14 日，可以记为 $(fever, 11, 14)$ 。值得一提的是当 $t_{start} = t_{end}$ 时，事件就变为瞬时事件，本文不考虑这种情况。

2.2.2 事件序列定义

一个基于时间段的事件序列或事件序列记作 $\zeta = \{E_1, E_2, \dots, E_m\}$ ，其中 E_i 为上述时间间隔事件。 ζ 是一系列事件的有序集合，这些事件按照开始时间从早到晚排列，对于开始时间相等的事件，按照事件标签的字母顺序排序。

一个事件序列中所包含的时间间隔事件的数量称作该序列的长度，若 ζ 中有 k 个事件，那么这个序列就称作 k -序列。如图 2-1 所示，是一个客户序列，这个序列可以表示为： $\zeta = \{(A, 1, 7), (B, 3, 19), (D, 4, 30), (C, 7, 15), (C, 23, 42)\}$ 。

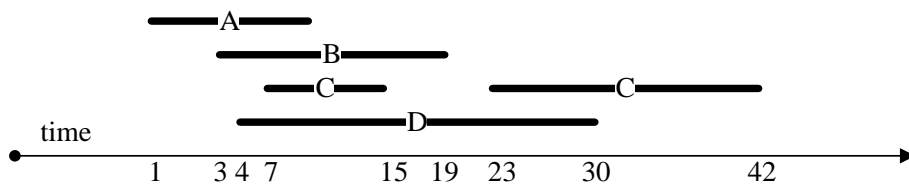


图 2-1 时间间隔事件序列示例

2.2.3 数据库定义

前文中给出了事件序列的概念，那么多个事件序列组成一个数据库： $D = \{\zeta_1, \zeta_2, \dots, \zeta_n\}$ ，其中每个事件序列就是一个客户序列。在本文中产生的数据库有三列，第一列为客户标志，每个客户对应一个客户标志，而具有同一个客户标志的元组同属一个事件序列；第二个为事件标签，每一个标签对应于一种事件，例如在医院病历中，事件标签可以对应发烧、打喷嚏、咳嗽等等；第三个就是事件的发生时间段。下面是一个小的医院病人病历数据库，如表 2-1 所示，其中同属一个 Patient 的病状为一个序列。本文实验部分的输入正是这样的数据库形式。

表 2-1 数据库示例

病人	病症	发病时间段	病人	病症	发病时间段
01	a	(5,10)	02	c	(16,18)
01	b	(8,12)	02	c	(19,21)
01	c	(14,18)	02	d	(16,22)
01	d	(14,20)	03	a	(12,20)
01	b	(17,22)	03	b	(22,25)
02	a	(8,14)	03	c	(29,31)
02	b	(9,15)	03	d	(28,32)

2.3 事件关系

带有时间间隔事件频繁模式挖掘与传统的序列频繁模式挖掘主要区别就在于事件关系的处理上。传统的序列模式挖掘中的事件是瞬时事件，这样事件的发生关系很简单。若是两件事在一个序列中发生，并且两件事情总是前后发生的，那么就称是这两件事是串行关系，反之若是没有前后约束，两个件事就是并行关系。而对于带有时间间隔事件来说，因为每个事件都有两个时间点，事件间的关系就要复杂得多，所以才需要适合处理这种关系的算法。这也正是本文的主要研究内容，下面就对时间间隔事件间关系表达做详细的介绍。

2.3.1 事件关系定义

从目前的国内外研究现状来看，对于 2.2 节中描述的事件序列，人们大都采用 Allen 在 1994 年提出的关系定义模式^[17]。文献[17]中定义了 13 种关系模式：“before”，“after”，“during”，“contain”，“meet”，“met by”，“overlap”，“overlapped by”，“start”，“start by”，“finish”，“finished by”，和“equal”如图 2-2 所示。

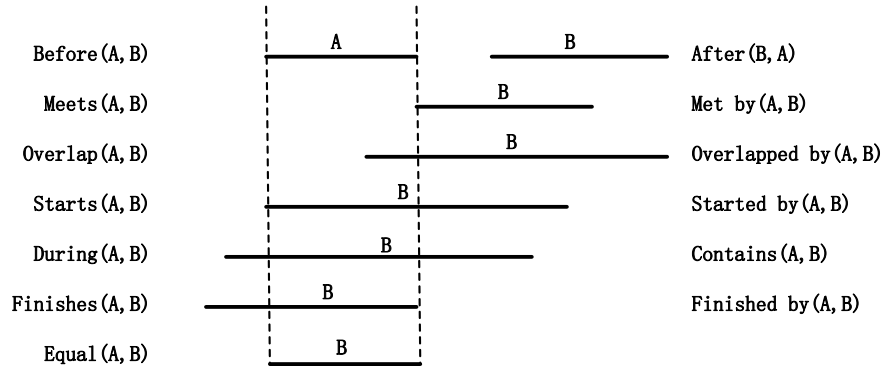


图 2-2 Allen 关系定义

对于两个事件关系来说，Allen 关系模式已经很完整。前面已经提到本文采用的数据库中的序列是有序的，即事件按照开始时间排序，因此可以将上述的 13 中关系简化为 7 种，他们的数学定义如下（在这里沿用 2.2 节中的事件定义，以后不再赘述），在定义中默认为 A 在 B 之前，即 $t_{start}(A) \leq t_{start}(B)$ ：

(1) **Before(A,B)**: B 在 A 结束之后发生，其关系记为符号“b”，也有文章记为“ $A \rightarrow B$ ”。当且仅当 $t_{end}(A) < t_{start}(B)$ 时，我们称 A before B。

(2) **Meet(A,B)**: 事件 B 在事件 A 结束时发生，记为符号“m”，也有文章记为“ $A \sim B$ ”。当且仅当 $t_{end}(A) = t_{start}(B)$ 时，我们称 A meet B。

(3) **Contain (A,B)**: 事件 A 包含 B，记为符号“c”，也有文章记为“ $A > B$ ”。当且仅当 $t_{start}(A) < t_{start}(B)$ 且 $t_{end}(A) > t_{end}(B)$ 时我们称 A contain B。

(4) **Start(A,B)**: 事件 A, B 同时发生，记为符号“s”，也有文章记为“ $A \mid > B$ ”。当且仅当 $t_{start}(A) = t_{start}(B)$ 且 $t_{end}(A) \neq t_{end}(B)$ 时，我们称 A start B。

(5) **Overlap(A,B)**: 事件 A 与 B 重叠，记为符号“o”，也有文章记为“ $A \mid B$ ”。当且仅当 $t_{start}(A) < t_{start}(B)$ 且 $t_{end}(A) < t_{end}(B)$ 时，我们称 A overlap B。

(6) **Finish(A,B)**: 事件 A 与 B 同时结束，记为符号“f”，也有文章记为“ $A > \mid B$ ”。当且仅当 $t_{start}(A) < t_{start}(B)$ 且 $t_{end}(A) = t_{end}(B)$ 时，我们称 A finish B。

(7) **Equal (A,B)**: 事件 A 与 B 同时发生并且同时结束，记为符号“e”，也有文章记为“ $A \parallel B$ ”。当且仅当 $t_{start}(A) = t_{start}(B)$ 且 $t_{end}(A) = t_{end}(B)$ 时，我们正 A equal B。

虽然 Allen 关系定义很完整，但是仅局限于两事件间关系。一方面，若是按照 Allen 关系定义，相同的事件间关系模式能用不同表达式表达，图 2-3 a)说明的就是这种情况，在图中一个模式中两两事件的关系是相同的，但是表达式却不同，第一个是 $((a \text{ overlap } b) \text{ overlap } (c \text{ overlap } d))$ ，而第二个则是 $((a \text{ overlap } b)$

contain c) overlap d)。

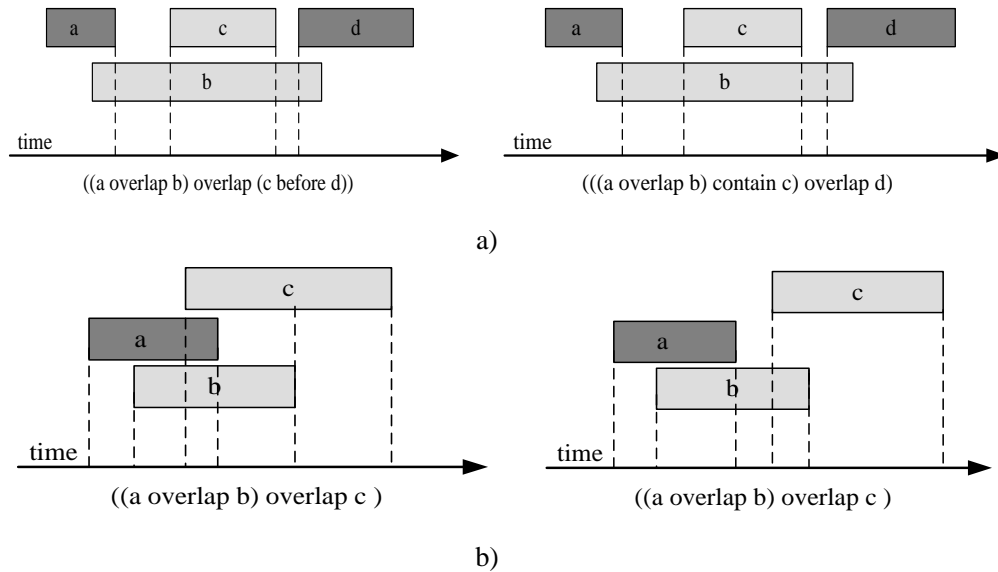


图 2-3 Allen 关系局限性说明示例

另一方面，不同的模式却可以用相同的表达式表达，如图 2-3 b) 中所示，两者为不同模式，第一个图中 $a \text{ overlap } c$ ，第二个图中 a 与 c 的关系是 before 。但是表达式中并没有体现出这一点。

综上所述 Allen 关系定义可以完美的表达两个事件间关系，却对多个事件间关系无能为力，因此衍生出了多种基于 Allen 关系定义的多事件间关系定义。下面我们将针对三个以上事件间关系进行讨论。

2.3.2 关系表达方法

多个时间间隔事件间的关系是基于 Allen 关系定义的。现在已经有很多文章中给出了多个事件关系定义，主要分为两类，一类是列举出任意两个事件关系。比较经典的有文献[19]中列举的矩阵表示，如图 2-4 所示。图中矩阵中的符号沿用 2.2 节定义。为了减少存储空间，在具体的算法实现中可以用如图所示的上三角矩阵。这种表示方式简单明了，运用比较普遍。

同时在文献[15]中运用了指针来表达模式中的事件关系，如图 2-5 所示。这种方法为一个模式中的任意两个事件建立一个指针，指向已经挖掘出的 2-频繁模式。这种方法的局限在于必须在挖掘长度大于等于 3 的频繁模式之前要挖掘出所有的 2-频繁模式。然而，也正是因为这样，这一条件能够很好的限制后续频繁模式的挖掘，即有利于剪枝策略。

多个事件关系定义的另一类是根据具体算法而衍生出的自定义关系定义方式。例如在文献[20]中自定义了一种关系表达方式，如图 2-6 所示。在图中右侧

的向量代表 contain, finish, overlap, meet, start 五种关系，数字代表向量右侧事件和向量左侧的所有事件间属于相应关系的数量。在图 2-6 中，以 C 前的 (0,0,1,1,0) 为例，他与 A 是 meet 关系，与 B 是 overlap 关系，因此向量为 (0,0,1,1,0)。并且文中已经证明了这种表达方式的正确性。这种方式并不常见，需配合相应的算法共同作用，不然是没有意义的。

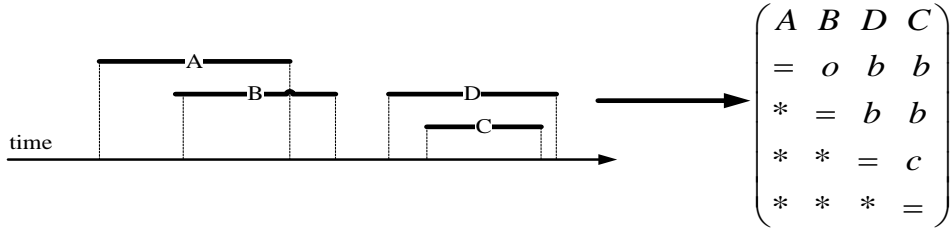


图 2-4 多个事件间关系示例 1

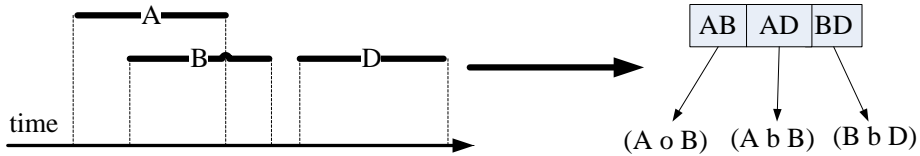


图 2-5 多个事件间关系示例 2

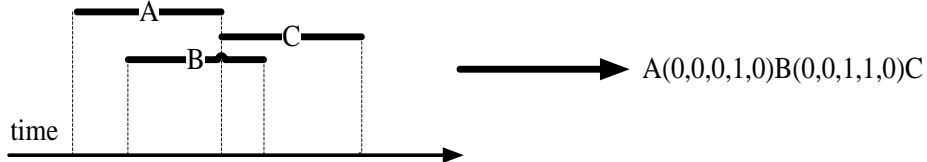


图 2-6 多个事件间关系示例 3

2.3.3 事件关系的现实约束

2.3.3.1 关系的重定义

在理论上前文的论述已经能够详细明确的表达带有时间间隔事件间的关系，但是关系定义在实际中的运用是有极大弹性的。前文所叙述的 7 种关系在本文中是互斥的，但是在实际运用中用户未必需要所有的关系模式。例如用户只需要“before”，“overlap”，“contain”三种关系，这时我们可以将“finish”和“equal”关系并入到“contain”关系中，而“start”根据第二个事件的结束时间归入到“contain”或“overlap”关系中，最后“meet”关系并入到“before”关系中。具体关系定义情况视具体情况而定，就有很大的弹性。另外需要说明的是，本文中是按照 7 种关系定义挖掘频繁模式的，忽略用户实际应用中重新定义关系模式的情况。

2.3.3.2 时间限制

在现实生活中事件发生的时间并非非常精确。例如病人的病历信息中，一种病状的开始和结束并不能精确到分秒，这样就使事件界限的定义模糊不清，也使得事件间的关系定义模糊不清，如图 2-7 所示。很遗憾的前文叙述的 Allen 关系定义并没有好好的解决这一问题。

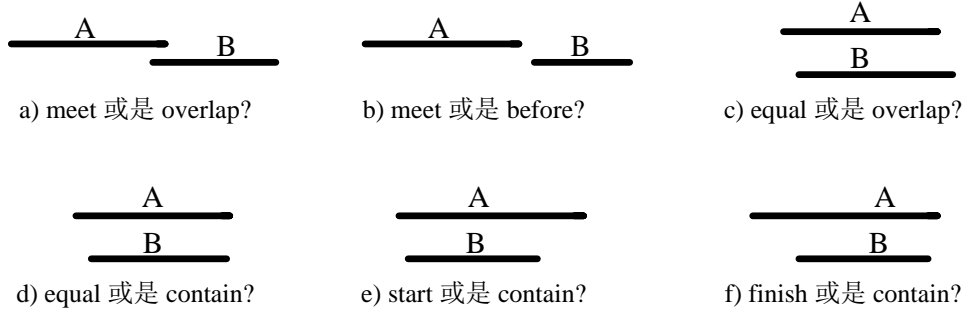


图 2-7 事件时间间隔界限模糊示例

本文提出了一种方法来解决这个问题，那就是加入时间限制。通过加入一个时间限制，可以将上述的 7 中关系转化成如图 2-9 所示的七种关系。在图中可以看出关系定义的基本概念是相同的，只是在边界上加入一个限制，当时间差超过这一限制时，两个事件之间的关系才成立。

给定一个阈值 ε ，阈值为正， ε 对于不同的关系定义是不同的。那么七种关系可以重新定义如下：

- (1) **Before(A, B):** $t_{end}(A) < (t_{start}(B) - \varepsilon)$;
- (2) **Meet(A, B):** $(t_{start}(B) - \varepsilon) \leq t_{end}(A) \leq (t_{start}(B) + \varepsilon)$;
- (3) **Contain (A, B):** $t_{start}(A) \leq (t_{start}(B) - \varepsilon)$ 且 $t_{end}(A) \geq (t_{end}(B) + \varepsilon)$;
- (4) **Start(A, B):** $(t_{start}(B) - \varepsilon) \leq t_{start}(A) \leq (t_{start}(B) + \varepsilon)$ 且 $t_{end}(A) < (t_{end}(B) - \varepsilon)$,
 $\leq t_{start}(A) \leq (t_{start}(B) + \varepsilon)$ 且 $t_{end}(A) > (t_{end}(B) + \varepsilon)$;
- (5) **Overlap (A, B):** $t_{start}(A) < (t_{start}(B) - \varepsilon)$ 且 $t_{end}(A) < (t_{end}(B) - \varepsilon)$;
- (6) **Finish (A, B):** $t_{start}(A) < (t_{start}(B) - \varepsilon)$ 且 $(t_{end}(B) - \varepsilon) \leq t_{end}(A) \leq (t_{end}(B) + \varepsilon)$;
- (7) **Equal (A, B):** $(t_{start}(B) - \varepsilon) \leq t_{start}(A) \leq (t_{start}(B) + \varepsilon)$ 且 $(t_{end}(B) - \varepsilon) \leq t_{end}(A) \leq (t_{end}(B) + \varepsilon)$ 。

特殊的，对于关系“before”来说，在一些情况下，两个事件的相隔时间太长，那么这种“before”关系是没有意义的。所谓两个事件 A 和 B 的事件差为 $t_{start}(B) - t_{end}(A)$ ，那么设置一个最大时间差 \max ，当 $t_{start}(B) - t_{end}(A) > \max$ 时，两个事件的“before”关系不成立，也就是两个事件没有任何关系，不纳入频繁模

式中。总结时间限制如图 2-8 所示。

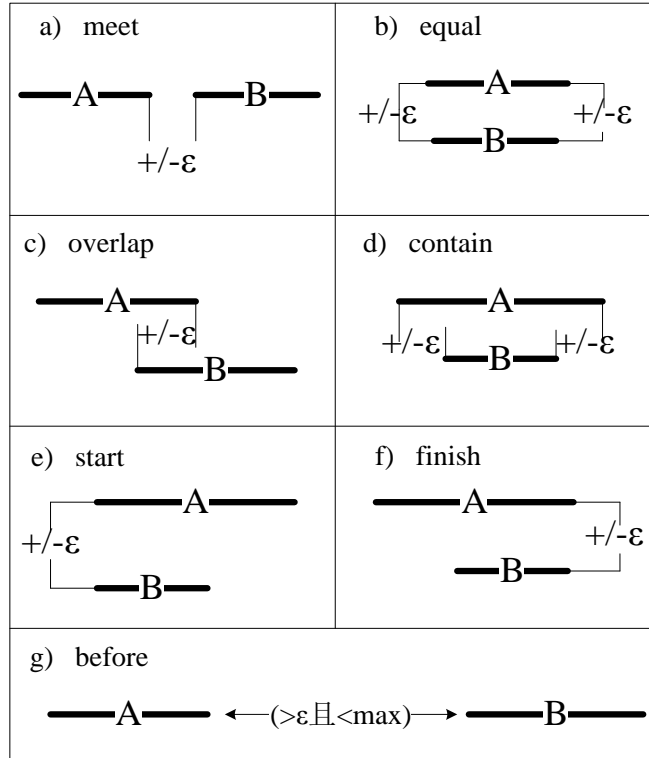


图 2-8 时间限制示例

2.4 带有时间间隔事件序列频繁模式挖掘

前面已经介绍了本课题的涉及到的一些主要知识点，下面这节介绍时间间隔事件序列频繁模式挖掘，也是本文研究的重点。本节的内容将以前面所叙述的为基础。本节将给出模式以及频繁模式的概念，介绍几种兴趣度以及简单的阐述一下关联规则。

2.4.1 时间间隔事件序列频繁模式

在 2.2 节中已经介绍了事件序列以及间隔事件关系概念，在这基础上下面给出间隔事件模式概念^{[21][22]}。前文中论述的加入时间限制的关系定义视具体环境的不同而波动很大，因此在本文中采用的是未加入时间限制的 7 种关系定义。定义一个关系集合 $\mathfrak{R} = \{o, s, m, f, e, b, c\}$ ，集合中的字母代表前面论述的七种关系。

(1) **模式** 一个包含 n 个事件的模式就定义为 $A = \{\zeta, R\}$ ，其中 ζ 为前文中介绍的事件有序集合， $|\zeta| = n$ 。 R 为一个关系序列，包含了 ζ 中任意两个事件间的关系， $R = \{R(E_1, E_2), R(E_1, E_3), \dots, R(E_1, E_n), R(E_2, E_3), \dots, R(E_2, E_n), \dots, R(E_{n-1}, E_n)\}$ 。而 $R(E_i, E_j) \in \mathfrak{R}$ 。一个模式的大小等于 $|\zeta|$ ，一个大小为 k 的模式称为 k -模式。

例如图 2-6 中的 $\zeta = \{A, B, D\}$, $R = \{R(A, B), R(A, D), R(B, D)\} = \{a, b, b\}$ 。

(2) **子模式** 定义一个模式 $A = \{\zeta, R\}$, 一个 A 的子模式 $A_i = \{\zeta_i, R_i\}$, A_i 是一个模式, 并且其中 $\zeta_i \subseteq \zeta, R_i \subseteq R$ 。

(3) **支持度** 一个模式的**绝对支持度**定义为在数据库中包含这个模式的序列条数; 一个模式的**相对支持度**定义为在数据库中包含这个模式的序列数量占整个数据库序列条数的半分比。

(4) **频繁模式** 在本文中采用支持度为衡量指标, 即设定一个支持度阈值 (最小支持度), 若一个模式的相对支持度大于等于这个阈值, 那么这个模式就是**频繁模式**。

值得一提的是, 本文利用 (1) 中定义的模式来定义频繁模式, 在下文中我们将以 $A = \{\zeta, R\}$ 的形式来定义模式, 只是关系的表达 R 的顺序有所不同, 在本文中 $R = \{R(E_1, E_2), R(E_1, E_3), R(E_2, E_3), R(E_1, E_4), \dots, R(E_1, E_n), \dots, R(E_{n-1}, E_n)\}$, 即新加入一个事件都与前面的所有事件进行关系定义, 并放在关系序列的后面。这是本文算法的需要。

例如在表 2-1 中, 若设置最小支持度为 100%, 那么 Before(a,c) 及 Before(a,d) 都是频繁模式。指定最小支持度, 挖掘出所有频繁模式正是本文的最后目标。虽然用最小支持度作为兴趣度挖掘频繁模式很有效率, 但是在用频繁模式推导关联规则的过程中最小支持度也存在着自身的缺点。下面一节将讨论关联规则生成及兴趣度对他的影响, 这部分不是本文的主要研究内容, 所以只是略作介绍。

2.4.2 时间间隔事件关联规则挖掘

2.4.2.1 关联规则

挖掘频繁模式的目标就是构造关联规则, 这样在实际运用中才能发挥作用。虽然构造关联规则不是本文的研究重点, 但是鉴于关联规则的重要性, 下面将介绍关联规则的相关内容。

文献[9,23, 24]中介绍了基于瞬时事件序列的关联规则挖掘, 在此基础上可以定义基于时间间隔事件的序列关联规则挖掘。已知数据库 D , 沿用前文中的定义, 下面是一个泛化的关联规则定义:

$$r : A_i \xrightarrow[D, \lambda]{R_{ij}} A_j$$

上式中 r 表示为基于兴趣度 λ 的 A_i 和 A_j 间关联规则。关联规则的意义可作如下解释: 沿用前文叙述的定义, 有一个数据库 D 中的频繁模式 $A = \{\zeta, R\}$ 其中

$\zeta = \{E_1, E_2, \dots, E_{|\zeta|}\}$ 。设 E_l 为序列 ζ 中的一个间隔事件， A 可以被 E_l 分为两个模式 $A_i = \{\zeta_i, R_i\}$ ， $A_j = \{\zeta_j, R_j\}$ ，其中 $\zeta_i = \{E_1, \dots, E_l\}$ ， $\zeta_j = \{E_{l+1}, \dots, E_{|\zeta|}\}$ 。在这里 ζ 被分割，同样的 R 也相应的被分割。而式中的 R_{ij} 表示 ζ_i 序列中事件和 ζ_j 序列中事件的关系。

2.4.2.2 兴趣度研究

兴趣度度量，也称作质量度量，在关联规则挖掘的过程中直接影响到挖掘的结果，是很重要的概念。

(1) **支持度—置信度模型** 支持度—置信度模型是经典、传统的关联规则生成模型。前文已经描述了支持度的概念，而支持度作为度量有一个明显的缺点，即稀少模式问题。也就是数据库中出现次数少的，非频繁模式可能更具有现实意义，而用支持度挖掘频繁模式的时候就将这样的模式剪枝掉了。

在挖掘关联规则的过程中第一步用支持度来筛选出频繁模式，第二步用置信度来筛选出相关的关联规则。在文献[25]中给出了置信度的概念： $confidence(X \Rightarrow Y) = P(Y/X) = P(XY)/P(X)$ 。在式子中看出置信度的意义是在 X 发生的条件下 Y 发生的概率，在很多文献中，一个规则的置信度大于等于所设置的最小置信度，那么这个规则就是关联规则。但是置信度作为兴趣度度量也存在一个明显的缺陷，就是只考虑了 X 与 XY 的关系，并没有考虑 Y 。例如当 X 与 Y 是独立的，那么 $P(XY)/P(X) = P(Y)$ ，若 Y 模式出现的概率足够大，即大于最小置信度，那么 $X \Rightarrow Y$ 成立，而实际上 X 与 Y 并没有关系。这就容易产生大量的没有意义的关联规则。

(2) **其他兴趣度度量** 鉴于前文所叙述的支持度—置信度模型的缺点，一些文献[26-32]改进或发现了一些其他的兴趣度度量。因为要挖掘出现次数少的并且具有现实意义的模式要加入主观人为因素，在数据处理方面有较大难度。所以这些改进主要针对置信度的缺陷，本节主要简单介绍三种兴趣度度量：

a) **兴趣度**，在文献[33]中给出了兴趣度的定义，如式 2-1 所示，兴趣度的意义是在 X 发生的条件下 Y 发生的概率与 Y 出现的概率的比值。兴趣度不具有向下闭合性。

$$lift(X \Rightarrow Y) = P(XY) / (P(X)P(Y)) \quad (2-1)$$

兴趣度的取值范围为 $[0, +\infty)$ 。值得说明的是当兴趣度为 1 时说明 X 与 Y 是独立的，并非关联。兴趣度大于 1 时说明 Y 在 X 条件下出现的概率更大一些，在这种情况下 $X \Rightarrow Y$ 是正相关规则。

b) **信任度**，文献[34]中给出了信任度的定义，如式 2-2 所示，信任度是一个

蕴含性的度量，它表示一种期望的概率。它的范围是 $[0, +\infty]$ ，当信任度为 1 时表示 X 与 Y 没有关系，相互独立；为 $+\infty$ 时规则是完备规则；而当信任度大于 1 则表示存在更大的期望。信任度很好的弥补了置信度不能解决相互独立的两个模式的缺陷。

$$conviction(X \Rightarrow Y) = (P(X)P(\bar{Y}) / P(X\bar{Y})) \quad (2-2)$$

c) **匹配度**，对比前述的几种兴趣度度量，匹配度有更实际的意义。文献[8]中给出了匹配度的概念，表达式如式 2-3 所示。匹配度的范围是 $[-1, 1]$ 。当匹配度等于 0 时 $P(XY) = P(X)P(Y)$, X, Y 无关，规则并非关联规则。当匹配度大于 0 时 $P(XY) > P(X)P(Y)$ ， X, Y 正相关，特别的匹配度为 1 时 $P(X) = P(Y)$ ，模式 X 与模式 Y 同时出现或同时不出现。

$$match(X \Rightarrow Y) = \frac{P(XY)}{P(X)} - \frac{P(\bar{X}\bar{Y})}{P(\bar{X})} \quad (2-3)$$

2.5 本章小结

本章主要介绍了本课题的基本理论知识，带有时间间隔事件和瞬时事件的主要区别就是关系定义，这也正是本文研究的重点与难点。在这一章中改进了 Allen 关系定义，并且介绍了各种基本概念。下面的文章将在此基础上展开。

第3章 频繁模式候选集生成

3.1 引言

本章开始介绍本文的主要研究成果,从本章开始本文提出了一个新的基于时间间隔事件序列频繁模式挖掘算法。前面已经介绍过基于 Apriori 思想的算法 GSP,本文中的算法也是基于这种思想,将这种思想应用于带有时间间隔的事件序列频繁模式上,并且给予相应的改进,使得能够高效的、准确的挖掘出频繁模式。

前面已经介绍过这种算法的基本思想分为两个步骤,在第一步中挖掘出频繁模式候选模式,而第二步是统计支持度,计算出需要的频繁模式。这两步循环进行直至没有候选模式,即挖掘出所有的频繁模式为止。

本文对这种思想的两个部分都进行了改进,在本章中描述了其他基于这种思想的候选集生成方法及详细描述了改进的时间间隔事件频繁模式候选集生成的过程,并且在这种过程中一一证明了改进的算法理论基础。

3.2 现有方法总结

前文已经提到过现今大多数算法都应用于序列模式挖掘领域中,带有时间间隔的事件序列模式挖掘算法较少。但是在这较少的算法中大多数都应用生成候选模式—计算支持度模型。在这一小节中本文将介绍几种候选模式生成算法:

(1) **枚举树算法** 在第一章中已经介绍过枚举树算法,这里要描述的是它的候选集生成方法。之所以叫做枚举树是因为这种算法将所有的频繁模式候选集枚举出来以树的形式存储,然后再通过不同的遍历方法计算其支持度确定是否是频繁模式。

这样固然能够列举出所有的候选频繁模式,但也正因为如此在候选模式生成过程中并没有为算法的效率做出贡献,将剪枝的任务都留给支持度计算步骤上。换言之,枚举树算法中的候选频繁模式生成过程简单,但是却生成了很多冗余的模式,给下一步的遍历带来了冗余。

在本文中提出的候选模式生成算法很好的解决了这一问题。

(2) **索引集算法** 前文中同样介绍过索引集算法,索引集算法严格上来说并不是基于 Apriori 思想的,但是它本质上还是基于候选模式生成—支持度计算模型^[35-37]的。如图 1-2 中所示,在得到 1-索引集后遍历索引集中指针指向的序列,得到的是长度为 2 的频繁模式候选集。在这一算法中遍历并不是所有数据库条目,

虽然减少了一定的候选频繁模式，但是在生成候选集的时候还是没有加入剪枝的因素。

另外，除了上述的缺陷，前文也提到过索引集算法的候选集生成有一个明显的缺陷，就是当一条记录中有相同的模式时，索引集算法并不能生成全部的候选频繁模式，即有缺失。从而将导致最后的频繁模式结合的缺失。

本文提出的候选集算法改进了这一缺陷，能够挖掘出所有的频繁模式。

(3) GSP 算法 前文提到的 GSP 算法是序列模式挖掘，相应的时间间隔序列模式挖掘算法的思想是一样的。GSP 算法是比较传统的基于 Apriori 的算法，它生成候选频繁模式的方式是将一层的频繁模式与自身相结合生成下一层的候选频繁模式。这里说的“一层”意指具有相同长度的频繁模式。

例如图 3-1 a) 中所示的两个 3-频繁模式，当 (A,B,D) 去掉第一项 A，而 (B,D,E) 去掉项 E 后剩余的相等，即都是 (B before D) 模式，那么将两个频繁模式合并成如图 3-1 b) 中所示的模式，该模式加入候选集。

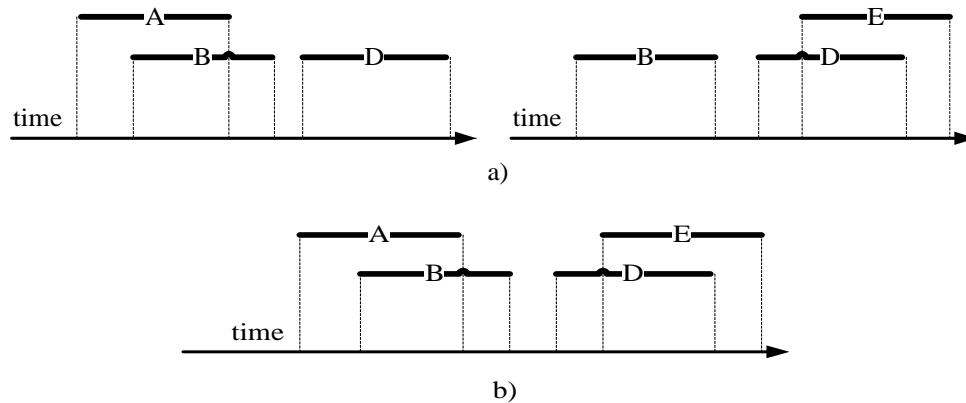


图 3-1 GSP 算法频繁模式候选集生成示例

在 GSP 算法产生候选集的过程中，要判断两个模式分别去除第一项和最后一项之后是否相等，这在序列模式挖掘中是比较容易得，但是对于带有时间间隔的事件来说，关系比较复杂，判断一个模式和另一个模式是否相等是比较消耗时间的。因此不同于 GSP 算法中将一层的频繁模式与自身相结合产生候选频繁模式，本文提出一种将上一层频繁模式与 2-频繁模式相结合产生下一层的候选频繁模式集。这样更能在保证完全的基础上减少候选频繁模式的数量。

3.3 改进的频繁模式候选集生成方法

本文中生成频繁模式候选集的主要思想是用长度为 k 的频繁模式与长度为 2 的频繁模式相结合产生长度为 $k+1$ 的频繁模式候选集。因此首先要产生 2-频繁模式。前文中叙述了各个挖掘算法的效率，在这其中文献[15]证明索引集算法效

率较高，所以在本文中对索引集算法的缺陷做了一点改进，然后引用索引集算法来挖掘 2-频繁模式。所以在这一小节中首先叙述 2-频繁模式的生成，然后再引入改进的频繁模式候选集生成算法。

3.3.1 二层频繁模式生成

在本文中引用文献[21]中叙述的索引集算法来挖掘出所有的 2-频繁模式，并且针对上文叙述的索引集算法的不足做了小的改进。算法首先遍历数据库得到所有的 1-频繁模式，并存储 1-频繁模式。然后构造 1-频繁模式的索引集，其数据结构如图 3-2 a)所示。例如表 2-1 的数据库中，假设支持度为 100%，得到 1-频繁模式 C 后，构造 C 的索引集为图 3-2 b)所示。对比之前的索引集算法的改进是当构造一个频繁模式的索引集时，将一条记录中所有该模式的时间间隔及位置都记录下来，这样就能避免索引集的缺陷。

在构造 1-频繁模式索引集后生成 2-频繁模式，根据索引集查找指向的客户序列，并且从索引集指定的位置之后开始将序列中的项加入 1-频繁模式。同样在上述例子中，根据 C-索引集可以得到 2-频繁模式候选集。客户序列 1 中：Start(C,D), Overlap(C,B)，客户序列 2 中 Before(C,C), Start(C,D), Before(C,D)，客户序列 3 中 C 是最后一项，因此不产生频繁模式候选项。若最小支持度为 50%，可以得到 2-频繁模式 Start(C,D)。

客户ID	时间间隔序列	在序列中位置序列
a)		
1	(14,18)	3
2	(16,18) (19,21)	3,4
3	(29,31)	4
b)		

图 3-2 索引集数据结构示例

在得到所有的 2-频繁模式后，将其以一定数据结构存储起来以满足后续挖掘的要求，其数据结构如图 3-3 a)所示，例如上述例子中的 Start(C,D)可以表示为如图 3-3 b)。图中的时间间隔序列表示的是符合这种模式的所有间隔事件对。

事件1	事件2	时间间隔序列	关系
a)			
C	D	(14,18)(14,20), (16,18)(16,20)	s
b)			

图 3-3 2-频繁模式数据结构示例

值得一提的是在本小节中叙述的 2-频繁模式数据结构是根本的定义，在后续的章节中会在这基础上再增加元素。后面将会进行详细论述。

3.3.2 改进的算法思想

本小节将于前面叙述的基础上详细论述本文对频繁模式候选集生成算法的改进思想，并且证明相关理论。在开始论述前本文给出一个定义如下：

定义 1: 在一个模式 A 的所有事件中，若 E 的结束时间是最晚的，那么就称 E 为 A 的**最晚结束事件**；相对的，若 E 的开始时间是最早的，那么就称 E 为 A 的**最早开始事件**。

在本文中连接 k -频繁模式与 2-频繁模式生成 $k+1$ -频繁模式候选集，合并的规则就是若 k -频繁模式的最晚结束事件等于 2-频繁模式的最早开始事件，就将两者合并产生候选频繁模式。例如在表 2-1 所示的数据库中，最小支持度为 50%，假设已经挖掘出所有的 2-频繁模式，现在用 2-频繁模式与 2-频繁模式相结合产生 3-频繁模式候选集，如图 3-4 所示。

在图 3-4 中，第一列代表已经挖掘完成的 k -频繁模式，这里 $k=2$ ，最晚结束事件用下划线标出；第二列表示 2-频繁模式，最早发生事件也用下划线表示。将最晚结束事件与最早发生事件相等的模式相结合，就得到第三列的 3-频繁模式候选集。

基于以上方法，对于每个频繁模式，为了能够快速找到最早发生事件与之最晚结束事件相等的 2-频繁模式，本文将 2-频繁模式按照序列中的第一个事件分块，即以最早发生事件为 key 做一个索引集。这样能够更快的在大量的 2-频繁模式中找到合适的构造候选频繁模式。

在上述的算法运行过程中存在两个问题。第一，不能从一个模式的事件序列中直接看出最晚结束事件，例如“contain”关系的两个事件组成的序列，在序列最后的不是最晚结束事件。为了减少在运算过程中消耗的时间，在多事件模式的数据结构中加入一项“最晚结束时间”可以解决这个问题。而对于 2-频繁模式来说不存在这个问题，因为在 2-频繁模式中最早发生事件总是事件序列中的第一个。第二，在结合 k -频繁模式和 2-频繁模式时，虽然得出了候选频繁模式的序列，但是关系并没有完全得到。例如将 $\text{Overlap}(A,B)$ 与 $\text{Overlap}(B,C)$ 结合， A 与 C 的关系是不明确的，即 (A,B,C) 不是一个完整的模式。为了解决这一问题，在多事件模式的数据结构中需将属于该模式的所有模式都枚举出来，即记录所有的时间段，再将确定要加入候选模式的项的时间段与这些时间段比较，得出完整的候选频繁模式。

k -频繁模式	2-频繁模式	$k+1$ -频繁模式候选集
Overlap(a, <u>b</u>)	Overlap(<u>a</u> ,b)	$\{(a,b,c),(o,b,b)\}$
Before(a, <u>c</u>)	Before(<u>a</u> ,c)	$\{(a,b,d),(o,b,b)\}$
Before(a, <u>d</u>)	Before(<u>a</u> ,d)	$\{(a,b,c),(b,b,b)\}$
Before(a, <u>d</u>)	Before(<u>a</u> ,d)	$\{(a,b,d),(b,b,b)\}$
Before(a, <u>b</u>)	Before(<u>a</u> ,b)	$\{(a,b,d),(b,b,b)\}$
Before(b, <u>c</u>)	Before(<u>b</u> ,c)	$\{(a,c,d),(b,b,s)\}$
Before(b, <u>d</u>)	Before(<u>b</u> ,d)	$\{(b,c,d),(b,b,s)\}$
Start(c, <u>d</u>)	Start(<u>c</u> ,d)	

图 3-4 候选集生成示例

综上，多事件频繁模式的数据结构如图 3-5 所示。例如上例中的频繁模式 $A = \{\zeta, R\}$ ， $\zeta = \{a, b, c\}$, $R = \{o, b, b\}$ 。事件序列为 ζ ；关系序列如第二章定义的，在这里等于 R ；时间段序列为 $\{(5,10) (8,12) (14,18), (8,14) (9,15) (16,18), (8,14) (9,15) (19,21)\}$ ，而对于相同的模式，最晚结束事件是相同的，所以即使时间段序列表示出了所有符合此模式的模式，最晚结束事件只有一个，在这里为 b 。

事件序列	关系序列	时间段序列	最晚结束事件
------	------	-------	--------

图 3-5 多事件模式数据结构

3.3.3 剪枝策略

在上面的论述中已经说明了产生候选集的主要思想，从候选集构造过程中可以看出 2-频繁模式的数量能直接影响到生成的候选模式的数量，其中大多数候选频繁模式都是冗余的。而作为下一步支持度统计的输入，候选集的数量直接影响到整个挖掘过程效率。因此本文提出一种策略来减少用于生成候选频繁模式的 2-频繁模式数量。在具体介绍该策略之前，本文给出了几个证明：

定理 1： 一个长度为 $k+1$ 频繁模式由一个长度为 k 的频繁模式和一个长度为 2 的频繁模式组成，那么这个 2-频繁模式至少出现在 $k-1$ 个长度为 k 的 $k+1$ -频繁模式子模式中。

证明：设长度为 $k+1$ 频繁模式为 A_{k+1} ，而组成它的 2-频繁模式中的两个事件

分别为 E_i 和 E_j ，分别在 A_{k+1} 的第 i 位和第 j 位，根据支持度的向下闭合性， A_{k+1} 有 $k+1$ 个长度为 k 的频繁子模式。那么在这 $k+1$ 个子模式中至少有 k 个包含 E_i ，同样至少有 k 个包含 E_j 。因此可知这 $k+1$ 个子模式中至少有 $k-1$ 个包含那个 2-频繁模式。

策略：基于以上理论本文提出可以维护一个事件集合，当得到 k -频繁模式时，先遍历这些 k -频繁模式得到出现次数大于等于 $k-1$ 的事件集合。在与 2-频繁模式相结合时，首先检查此 2-频繁模式的两项是否在集合中，若不在直接将此 2-频繁模式删除。若都在再检查是否最早发生事件与最晚发生事件相同。

基于以上理论，下面证明用上述方法得到的候选集是完整的，即任意频繁模式都能从候选集中得到。

定理 2：结合 3.3.2 节的算法与 3.3.3 节的剪枝策略得到的频繁模式候选集是完整的。

证明：首先算法得到了所有的 2-频繁模式，假设这种算法得到了所有长度为 k 的频繁模式，然后用这些 k -频繁模式与经过剪枝的 2-频繁模式构造长度为 $k+1$ 的频繁模式候选集，现证明这个候选集是完整的，即能产生所有的 $k+1$ -频繁模式。假设存在一个 $k+1$ -频繁模式没有生成，不失一般性，这个 $k+1$ -频繁模式的候选模式是由一个 k -频繁模式与一个 2-频繁模式构成的，由于已经产生了所有的 k -频繁模式，因此原因在于这个 2-频繁模式被剪掉了。根据定理 1，这个 2-频繁模式至少出现在 $k-1$ 个长度为 k 的 $k+1$ -频繁模式子模式中，而这样的 2-频繁模式时本文中算法留用的，矛盾，假设不成立，定理 2 成立。

3.3.4 算法实现

基于以上理论本小节给出了算法 $CandidateSetGenerate(FrequentSet, 2-frequentSet)$ 的伪代码。

Input: k -频繁模式集 $FrequentSet$ ，经上一步剪枝的 2-频繁模式集 $2-frequentSet$

Output: 长度为 $k+1$ 的频繁模式候选集

- (1) $CandidateSet = \emptyset, ItemSet = \emptyset$
- (2) for all ($A_k \in FrequentSet$) do
- (3) for all ($item \in A_k$) do
- (4) if ($item \notin ItemSet$) do
- (5) $item.num = 1$
- (6) $ItemSet = ItemSet \cup item$

```

(7)         end if
(8)         else  $item.num = item.num + 1$ 
(9)         end for
(10)    end for
(11)    for all ( $item \in ItemSet$ ) do
(12)        if ( $item.num < k - 1$ ) do
(13)             $ItemSet = ItemSet - item$ 
(14)        end if
(15)    end for
(16)    for all ( $A_k \in FrequentSet$ ) do
(17)        for all ( $A_2 \in 2-frequentSet$ ) do
(18)            if ( $A_2$  中  $item \in ItemSet$ ) do
(19)                if ( $A_2$  最早发生事件 =  $A_k$  最晚结束事件) do
(20)                     $CandidateSet = CandidateSet \cup join(A_2, A_k)$ 
(21)                end if
(22)            else  $2-frequentSet = 2-frequentSet - A_2$ 
(23)            end if
(24)        end for
(25)    end for
(26)    return  $CandidateSet$ 
    
```

其中 2—15 行给出了维护 2-频繁模式集合的过程，即剪枝过程。16—25 行则是利用修剪过的 2-频繁模式结合和 k -频繁模式集合，通过上文中论述的方法构造 $k+1$ -频繁模式候选集。

3.4 本章小结

在这一章中主要介绍了本文中的算法是如何产生候选频繁模式集的，主要提出一个合成策略和一个剪枝策略。作为算法的第一个部分，它的输出是一个包含相同长度候选频繁模式的集合，也是算法第二个部分—支持度计算的输入。只有经过支持度计算才能判定一个候选频繁模式是频繁模式。下面第四章将介绍本文算法的支持度计算部分。

第4章 支持度统计

4.1 引言

本文在挖掘时间间隔事件频繁模式时采用生成候选集—支持度统计的模型，另外在第二章中介绍过支持度有自己的局限性，即不能挖掘出稀少模式，但是本文试验中采用仿真数据，并没有真实场景的真实数据，所以并没有加入主观判断因素。因此本文中依然采用支持度为兴趣度衡量来挖掘频繁模式。

在生成候选集—支持度统计的模型中，当候选集生成后需要进行支持度运算，得出支持度大于等于给出的最小支持度的模式，即是频繁模式。上一章中已经详细的描述了本文提出的频繁模式候选集的生成算法，在本章中本文将从两种角度提出不同的支持度统计方法。

在本章中首先介绍了几种现存的比较经典的支持度计算方法，然后本章中将提出两种支持度计算方法，在本章的最后将总结第三章与第四章的内容。

4.2 现有方法总结

传统上的方法^[38,39]都是对于每一个候选频繁模式扫描数据库，例如对于 k -频繁模式来说，查询一个含有 m 个事件的序列需要 $O(mk)$ ，而对于 n 个候选频繁模式来说就需要 $O(mnk)$ ，这样的消耗是很大的。现有的算法中很多都对其进行了改进，下面就介绍比较经典的几种。

(1) **SPADE 支持度计算方法** SPADE^[12]是一种序列频繁模式挖掘算法，但是它的思想是可以应用于时间间隔事件序列的挖掘的，因此在这一小节中介绍了这种计算方法。

支持度计算都是与算法本身存储模式的数据结构绑定的，SPADE 也一样。SPADE 算法主要是经过将横向数据库转换成纵向的，每一个模式都配有一张表，在表中标有这个模式出现的客户序列号及发生的时间时刻，这一点和索引集算法相似。它的频繁模式候选集是通过上一层的模式的表进行联合操作得到的，在支持度的计算一步，算法遍历表中指向的客户序列计算该模式出现的次数。

这种计算方式能够遍历数据库中较少的客户序列，但是在算法开始数据库的转换是要花费很大时间代价的。

(2) **索引集支持度计算方法** 索引集算法^[19]中和 SPADE 中比较相似的是在应用的索引集中同样标记了包含该模式的客户序列号，但不同的是索引集查询指针指向的客户序列来产生候选频繁模式，并且在记录候选频繁模式的同时记录该

候选频繁模式的支持度。

(3) **枚举树支持度计算方法** 在文献[40]详细介绍了基于枚举树挖掘的算法,和 SPADE 算法相似的地方是,枚举树算法也将挖掘出的频繁模式利用 **IsId-List**^[41] 存储起来,即转换成纵向的。并将所有上一层的频繁模式结合形成下一层的候选频繁模式。对于支持度计算,枚举树采用的是传统的模式,即对于一个候选模式,都要遍历数据库来统计它的支持度,并且把该模式的每一条记录都存储在 **IsId-List** 中,最后再将那些不满足最小支持度的模式删除,并且在枚举树数据结构中将这一枝剪掉。

枚举树算法是比较传统的算法,思路简单明了,但是明显的无论是在候选频繁模式剪枝的问题上还是在支持度计算的问题上,该算法都存在着较大的效率缺陷。本文将在第五章给出本文算法与枚举树算法的比较。

4.3 改进的支持度计算方法

在本节中将提出本文对支持度计算方法的两种改进策略。第一,基于索引集算法的启发,在第三章中提出的候选集生成的算法基础上,算法可以记录候选频繁模式出现的序列号,然后在支持度计算的过程中,只查询这些有记录的序列,这将大大减少算法搜索空间。但是同时,传统上的对于每一个候选频繁模式都要搜寻一次数据库的弊端还是没有避免。因此本文又提出第二种策略,在这种策略的基础上,对于每一层的候选频繁模式,即具有相同长度的频繁模式,只要遍历一次数据库即可,而不是对于每一个候选频繁模式都要遍历一次数据库。下面就对上述内容进行详细论述。

4.3.1 数据库查找范围

在支持度计算的过程中,数据库查找范围是影响算法效率的主要因素,因此在本文中提出了一种能够减小数据库扫描范围的策略。在第三章论述的 2-频繁模式存储结构及多个事件模式存储结构分别更改为如图 4-1 a) b) 所示,在两种数据结构中都加入“客户序列号”一项,它和时间间隔序列是对应的,对于 k -模式,无论是候选还是频繁,有在“时间间隔序列”一项中有 $n*k$ 个时间段,其中 $n \geq$ 该模式支持度,因为可能在一条序列中出现不止一个这种模式。而客户序列号有 n 个,分别标明这个模式 n 次出现的客户序列,值得说明的是,若一个模式在一条客户序列中出现不止一次,则要记录两个相同的客户序列号。例如在本文的数据库示例中,对于频繁模式 **Before(a,c)** 来说,经过修改的数据结构为如图 4-1 c) 所示。

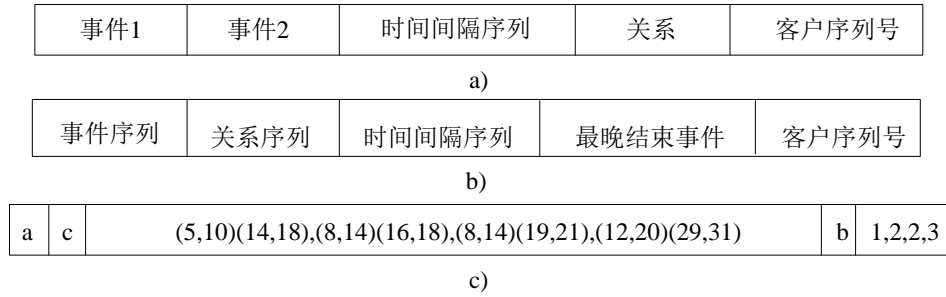


图 4-1 模式存储结构示例

客户序列号是在候选频繁模式生成的时候产生的，当产生 2-频繁模式时，利用索引集算法将客户序列号记录下来。而在利用本文提出的算法生成 $k+1$ -频繁模式候选集时，除了“最晚结束事件等于最早发生事件”和“2-频繁模式中的两项需要在至少 $k-1$ 个 k -频繁模式出现”外再加入一条限制条件，就是“合并的 k -频繁模式和 2-频繁模式的序列号相等”。并在产生的候选频繁模式中记录这一序列号。这一策略在产生候选频繁模式序列号的同时也减少了冗余的候选频繁模式的产生。

如此，在计算支持度的过程中，算法只需要遍历那些存储的序列即可。但是这里仍然有一个明显的缺陷，当合并 k -频繁模式和 2-频繁模式时，若序列号太多，查找匹配的序列号就会降低一定的效率，因此在本文中序列号都是有序从小到大存储的，可有效降低查找的时间。

4.3.2 支持度计算

上一小节描述了索引集的构造，这一小节将在此基础上论述本文提出的支持度计算方法。不同于枚举树算法的多次遍历数据库，本文提出一种对于具有同一长度的候选频繁模式 k -模式，只遍历一次数据库计算出支持度。

值得特别说明的是本文提出的支持度计算方法是针对每一个层次的频繁模式候选集而言，即对具有相同长度的所有候选频繁模式遍历一次数据库。因此将在将序列号添加到每个模式的数据库的同时，为每一层的候选频繁模式构造一个客户序列号集合，这个集合中为包含这一层次中的任意一个候选频繁模式的客户序列的序号。因此在计算一个层次的候选频繁模式的支持度时只需遍历这个集合所指向的客户序列即可。

在说明本文提出的支持度计算算法前首先说明一个定义，对于 k -候选频繁模式来说，在一条客户序列中的模式如果是这个候选频繁模式的前缀，就称这个模式是**活动的**。

本文提出的支持度计算的主要思想就是遍历一条客户序列，将所有活动的模

式记录下来，当遇到候选集中有该模式的时候，就将对应的候选频繁模式的支持度加一。

具体算法 $\text{SupportCount}(CandidateSet, EL)$ 实现如下：

Input: k -频繁模式候选集 $CandidateSet$ ，客户序列 EL ，候选频繁模式长度 k

Output: k -频繁模式候选集 $CandidateSet$

```

(1)  while (( $EL \rightarrow nextEvent$ )  $\neq$  null) do
(2)       $event = EL \rightarrow nextEvent$ 
(3)      for all ( $A \in activePattern$ ) do
(4)           $newPattern = join(A, event)$ 
(5)          if ( $newPattern.size = k \&\& newPattern \in CandidateSet$ ) do
(6)               $newPattern.support = newPattern.support + 1$ 
(7)          else if ( $newPattern$  是  $CandidateSet$  中某个模式的前缀) do
(8)               $activePattern = activePattern \cup newPattern$ 
(9)          end if
(10)     end for
(11)     if ( $event$  是  $CandidateSet$  中某个模式的前缀) do
(12)          $activePattern = activePattern \cup event$ 
(13)     end if
(14) end while
    
```

在 SupportCount 中 2—10 行逐个将一条客户序列中的事件与前面已经存储的前缀相结合，如果正式存在于频繁模式候选集中的模式，那么相应的候选模式支持度加一，若是结合后长度不足 k ，但是频繁模式候选集中一个模式的前缀，那么把这个合并后的模式也并入到 $activePattern$ 集合中。11—12 行是针对单项模式而言的，若这个单项模式是某一个候选模式的前缀，就将其并入到 $activePattern$ 集合中。下面举例说明上述算法过程，同样以本文的数据库示例为例，最小支持度为 50%，若现在已经从 3-频繁模式及 2-频繁模式生成 4-候选频繁模式集，只有一个模式 $A = \{(a, b, c, d), (o, b, b, b, s)\}$ ，客户序列号为 {1, 2}，因此分别对客户序列 1, 2 进行扫描，现在以客户序列 1 ($\{(a, b, c, d, b), (o, b, b, b, b, s, b, b, o, o)\}$) 为例，支持度计算过程如图 4-2 所示。

4.4 本章小结

本文在第三章叙述了候选集生成的算法，在第四章叙述了支持度计算的算法。下面这一小节将对两部分进行整合、总结。

nextEvent	newPattern	activePattern
a	—	a
b	{(a,b),(o)}	a, {(a,b),(o)}
c	{(a,c),(b)}, {(a,b,c),(o,b,b)}	a, {(a,b),(o)}, {(a,b,c),(o,b,b)}
d	{(a,d),(b)}, {(a,b,c),(o,b,b)}, {(a,b,c,d),(o,b,b,b,b,s)}	a, {(a,b),(o)}, {(a,b,c,d),(o,b,b,b,b,s)} {(a,b,c),(o,b,b)} 支持度+1
b	{(a,b),(b)}, {(a,b,b),(o,b,b)}, {(a,b,c,b),(o,b,b,b,b,o)}	a, {(a,b),(o)}, {(a,b,c),(o,b,b)}

图 4-2 支持度算法过程示例

4.4.1 算法流程

整个算法是基于频繁模式候选集生成—支持度统计模型的，本文提出的改进的算法的流程主要分为几个部分，分别是“2-频繁模式生成”、“合并 k 层频繁模式与 2-频繁模式生成 $k+1$ -频繁模式候选集”和“计算 k -频繁模式候选集中模式支持度”。算法的整个流程如图 4-3 所示。

4.4.2 算法实现

综合上文所述内容，下面给出整个算法的伪代码 $\text{FrequentMine}(D, \text{minsup})$ 。

Input: 数据库 D , 最小支持度 minsup

Output: 所有频繁模式集合 FrequentSet

- (1) 扫描 D 得到 1-频繁模式集, $\text{FrequentSet} = \text{FrequentSet} \cup 1\text{-频繁模式集}$
- (2) 构造 1-频繁模式索引集
- (3) 扫描索引集中指向的客户序列的 2-频繁模式: 2-frequentSet
- (4) $\text{FrequentSet} = \text{FrequentSet} \cup 2\text{-frequentSet}$
- (5) $\text{CandidateSet} = \text{CandidateSetGenerate}(2\text{-frequentSet}, 2\text{-frequentSet})$
- (6) $k = 3$
- (7) while ($\text{CandidateSet} \neq \text{null}$) do
- (8) for all ($EL \in \text{CandidateSet}$ 指向序列)
- (9) SupportCount ($\text{CandidateSet}, EL, k$)
- (10) end for
- (11) for all ($\text{pattern} \in \text{CandidateSet}$) do
- (12) if ($\text{pattern.support} \geq \text{minsup}$)

```

(13)           $k\text{-frequentSet} = k\text{-frequentSet} \cup \text{pattern}$ 
(14)      end if
(15)  end for
(16)       $\text{FrequentSet} = \text{FrequentSet} \cup k\text{-frequentSet}$ 
(17)       $\text{CandidateSet} = \text{CandidateSetGenerate}(k\text{-frequentSet}, 2\text{-frequentSet})$ 
(18)       $k = k + 1$ 
(19)  end while
    
```

在上面的算法中，1—6 行是前期准备工作，产生 2-频繁模式，7—19 行为如图 4-3 所示的循环过程。

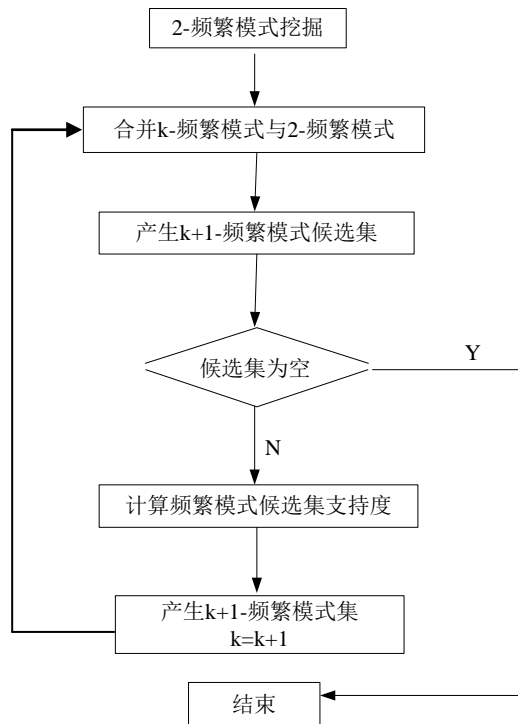


图 4-3 算法流程示例

综上，在支持度计算部分，本文提出了两个优化策略，其一为对于同一长度的候选频繁模式只遍历一次数据库完成计算，其二为在这一次遍历数据库的过程中，算法有效的减少了算法遍历空间，提高算法效率。至此本文论述了完整的频繁模式挖掘过程，下面第五章将对本文提出的理论给予实验证明。

第5章 实验及结果分析

5.1 引言

在这一章中本文将叙述实现本文提出算法的系统，主要分为三部分。第一，本系统采用仿真数据，在这一章的第一部分将描述数据的生成；第二，许多参数都对本文提出的算法有很大影响，在第二部分中本章将论述几个重要的参数对实验结果的影响；第三，在本文的前几章中已经较详细的描述了索引集算法及枚举树算法，那么在本章的第三部分将针对算法效率及正确率两个部分进行这三者间的对比。

本文之所以选择这两种算法进行对比实验是因为一方面在算法效率方面索引集算法是近几年算法中表现比较突出的，因为枚举树算法在效率上表现很差，没有对比意义。另一方面，在正确性对比上枚举树算法是公认的能挖掘出所有的频繁模式的算法，因此在正确性对比实验中以枚举树算法作为标准，即若是一种算法能够挖掘出同枚举树算法同等数量的频繁模式，那么这个算法就能挖掘出所有的频繁模式，反之则说明算法有一定的缺陷。

5.2 数据生成

本文中的实验是基于仿真数据的，在文中将介绍仿真数据器的原理。虽然本文的算法并没有应用于真实场景数据上，但是算法的对比是有意义的，因为在算法的效率对比上本文的算法主要与索引集^[17]算法作对比，而文[17]只是介绍了算法的实现，文[41]是对文[17]的扩展，在文[41]中介绍了仿真数据生成器的实现过程，索引集算法也是在这个数据生成器生成的数据上运行的。那么在本文中实现了同样的仿真数据生成器，并且在这个数据生成器的基础上再现了索引集算法和实现了本文的算法，并且在效率上对两个算法进行了对比。下面就介绍一下仿真数据生成器的实现过程。在文[41]中主要引入了五个参数，每个参数的概念和具体生成数据的方法如下：

5.2.1 主要参数

本文中实验都采用合成数据库，仿真数据。本文中数据生成主要依据五个主要指标，分别是：

(1) 事件类型，事件类型表示在数据库中一共有多少种事件，在不同的领域事件可代表不同事件，例如在零售业事件可代表不同商品、在医疗事业中事件可代表不同病症。在本文中为了表达计算方便，在仿真数据库中用不同数字表示不同

事件。

(2) 客户序列个数，客户序列个数可以看作客户的个数，关于这个客户所发生的事件都在一个客户序列中，这个参数可以间接的表示数据库的大小。

(3) 客户序列长度，客户序列长度表示在一条客户记录中有多少事件发生，这些事件可能重复。这一参数将直接影响到支持度的计算。客户序列长度并不是固定的，因此在本文中采用客户序列平均长度作为参数。

(4) 频繁模式个数，这里所说的频繁模式并不是算法挖掘出的频繁模式个数，而是在数据生成时系统自动生成的频繁模式个数，从而利用这些频繁模式来构造客户序列，算法实际上产生的频繁模式个数要多很多。

(5) 频繁模式长度，频繁模式长度即频繁模式中序列的长度，若有相同的事件发生算多个事件，通客户序列长度一样，不能确定频繁模式长度，所以在本文中采用的是频繁模式平均长度。

本文用这五个参数产生仿真数据，同时在下文也将给出这五个参数对算法运行结果的影响。为了叙述方便，这五个参数对应的字母表示如表 5-1 所示。

表 5-1 系统参数示例

系统参数	
事件类型	N_e
客户序列个数	$ D $
客户序列长度	L_c
频繁模式个数	N_p
频繁模式长度	L_p

5.2.2 数据生成过程

在数据生成的过程中，首先生成一定数量的频繁模式，然后根据这些频繁模式生成一定数量的客户序列。

(1) **频繁模式生成** 为了生成客户序列，系统首先生成一个存储一系列频繁模式的集合，集合的大小为 5.2.1 中叙述的 N_p 。而对于生成一个频繁模式来说，需要确定的参数有频繁模式的长度，频繁模式中事件，频繁模式中事件间关系及每个事件的发生时间段。系统将按照均值为 L_p 的泊松分布来分配一个频繁模式的长度 N ；然后根据已知的长度从事件集合（集合大小为 N_e ）中随即抽取 N 个事件，这 N 个事件中可能存在相同的事件；然后从集合 {o,s,b,f,e,c,m} 中随即抽取这 N 个事件的相互关系，共抽取 $N(N-1)/2$ 个，在这里值得一提的是为了符合

现实场景，若两个事件为相同事件，那么他们的关系只能是“meet”或“before”；最后系统将按照已有的事件间关系定义时间段。至此就产生了一个存储 N_p 个频繁模式的集合。

(2) **客户序列生成** 系统在生成频繁模式集合的基础上生成客户序列。同频繁模式生成类似，系统首先按照均值为 L_c 的泊松分布来确定客户序列的长度，然后从频繁模式的集合中随即抽取频繁模式组合成客户序列。这样共产生 $|D|$ 个客户序列。最后产生的以文件形式存储以供计算，格式如表 5-2 所示。

表 5-2 客户序列存储文件示例

客户序号	事件类型	开始时间	结束时间
1	76	1	4
1	24	4	8
1	35	6	11
2	3	6	11
...

5.3 算法效率验证

在这一节中本文将对算法的效率上的改进进行了验证，首先针对各个参数对算法效率的影响进行了实验，在这一部分中将三种算法都进行对比，然后将基于文[42]中产生的数据集对比本文提出的算法以及索引集算法效率。值得说明的是所有数据都是由仿真数据生成器生成，在第一部分中将使用比文[42]所用的数据集更大的数据集，而在第二部分中生成的数据集的各个参数都与文[42]生成数据集所用的参数相等，可以近似的看作是相等的数据集。

5.3.1 参数对算法效率的影响

在上述五个参数中由于定义的频繁模式并不是客户序列中实际出现的频繁模式，而客户序列长度只代表在生成数据集时的平均序列长度。因此本节对三个参数对算法的影响做了对比实验，其中三个参数为上述的客户序列个数 $|D|$ 、客户序列平均长度 L_c 、事件类别数量 N_e ，另外一个重要的参数为设置的最小支持度。参加对比实验的有本文作为对照实验的算法枚举树，索引集算法，和本文提出的改进算法三种算法，在这一节中将介绍上述参数对这三种算法性能的影响。

(1) **最小支持度** 最小支持度是决定频繁模式的关键性因素，支持度大于最小支持度的模式都定义为频繁模式。从理论上来说若最小支持度越小，将获得越多的频繁模式，从而产生更多的候选频繁模式，这无疑要花费更多的时间。反之

最小支持度设置的越大，产生的模式也越少，每一层的频繁模式也就越少，更加高效。

下面从实验上来证明，在实验中 $N_e=1000$, $N_p=2000$, $L_p=5$, $|D|=10$ 万, $L_c=20$ 。在实验中设置支持度的最小值为 0.05，最大的最小支持度为 0.1，步长为 0.01。实验结果如图 5-1 所示。从图 5-1 中可以看出随着支持度的增大，三种算法的效率都有显著的提高。这也正验证了前文中描述的支持度对算法效率影响的假设。并且枚举树算法效率提高的更显著，说明支持度对它的影响更大一些。

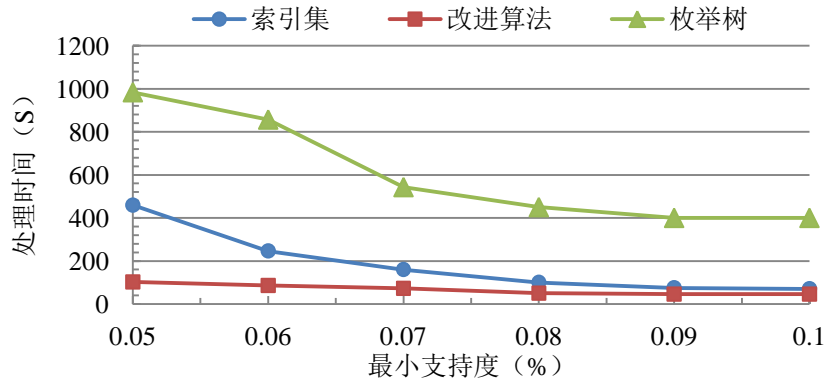


图 5-1 支持度参数对算法影响示例

(2) **客户序列个数** 客户序列个数表示为客户的数量，直接反应出数据库的大小，明显的，若是客户序列越多，挖掘频繁模式所花费的时间也就越多。基于客户序列个数这个参数的影响，与其它两种算法主要不同的是在本文提出的改进算法中用索引的策略来减少需要遍历的客户序列个数。下面来验证它对算法效率的影响，在实验中 $N_e=1000$, $N_p=2000$, $L_p=5$, $L_c=10$, $\text{minsupport}=0.05$ 。 $|D|$ 的范围从 10 万到 100 万，步长为 10 万。实验结果如图 5-2 所示，从图中可以看出三种算法的效率都随着客户序列数的增大而降低，其中又以本文改进的算法效率最高。另外枚举树算法效率降低的坡度最大，这说明枚举树算法受数据库大小的影响最大，不利于数据库扩展。相反，本文提出的算法和索引集算法的可扩展性更好一些。

(3) **事件类型** 事件类型的多少能够体现出一个模式的复杂与否，按照本文中系统生成数据的流程，事件类型越多，产生的模式就越多，无疑算法消耗的时间也就越长。这个参数对本文算法的影响主要体现在候选模式的生成上，若事件类型越多，在合并的时候需要查询是否符合的模式也就越多，从而消耗的时间就越多。下面就对上述的理论进行验证，在实验中 $N_p=2000$, $L_p=5$, $|D|=10$ 万, $L_c=10$, $\text{minsupport}=0.05$ 。而事件数目的范围为从 1000 到 10000，步长为 1000。实验结果如图 5-3 所示。

从图中可以看出三个算法的效率都随着事件类型数目的正大而降低，证实了上述理论。另一方面可以看出这一参数对枚举树算法的影响很大，因为和图 5-1 和图 5-2 相比，在相差不多的条件下，事件数目的增加能够显著的降低算法的效率。这主要是因为在这种算法中，需要针对每种模式维持一个表，即 IsId-List 表。如此在事件类型越多的情况下，模式的种类也就越多，枚举树算法需要维护的表就越多。而本文提出的算法弥补了这一缺陷。

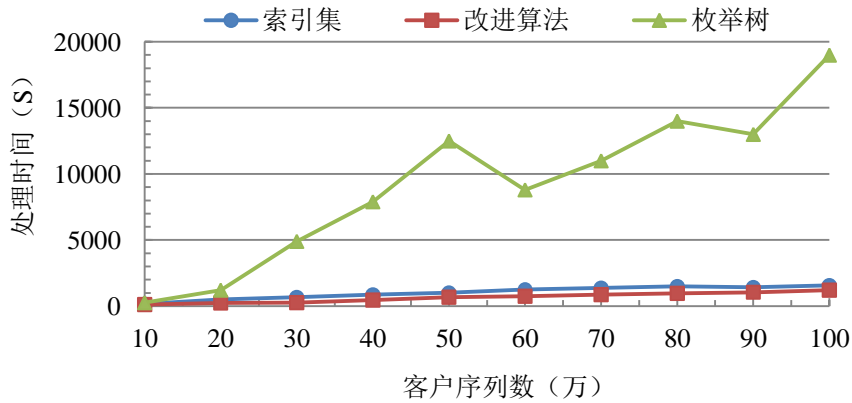


图 5-2 客户序列数目参数对算法影响示例

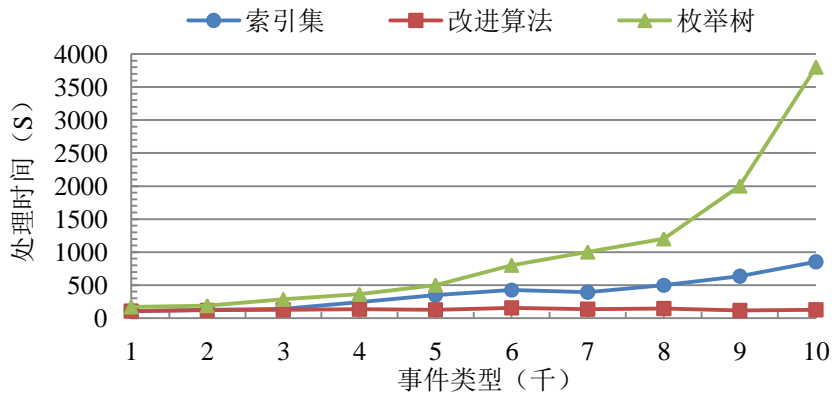


图 5-3 事件类型数目参数对算法影响示例

5.3.2 算法效率对比实验

前面介绍了各种参数对三种算法效率的影响，在这一小节中将重点对比索引集算法及本文中的算法，应用的数据集与文[41]中使用的数据集具有相同的参数。同样在两个方面：数据库大小以及事件类型对本文改进的算法已经索引集算法进行了对比。

(1) 客户序列个数 在实验中各个参数为 $N_e=1000$, $N_p=2000$, $L_p=5$, $L_c=10$,

minsupport=0.05。客户序列个数从 1 万到 10 万，步长为 1 万。实验对比图如图 5-4 所示。在客户序列个数上改进的算法有明显的优势，原因主要在于在支持度的计算上本文采用的优化策略能够有效的减小算法遍历空间。

(2) 事件类型 在实验中各个参数为 $N_p=2000$, $L_p=5$, $|D|=5$ 万, $L_c=10$, minsupport=0.05。事件类型数目从 1000 到 1 万，步长为 1000。实验对比图如图 5-5 所示。从图中可以看出在事件类型的影响上索引集算法和改进的算法效率相当。

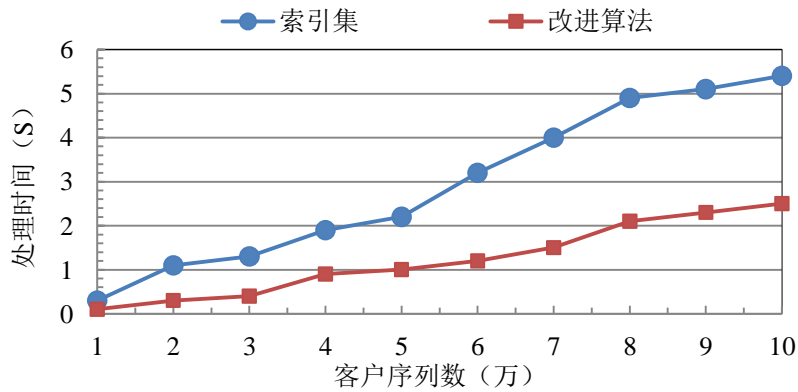


图 5-4 算法效率对比实验 1 示例

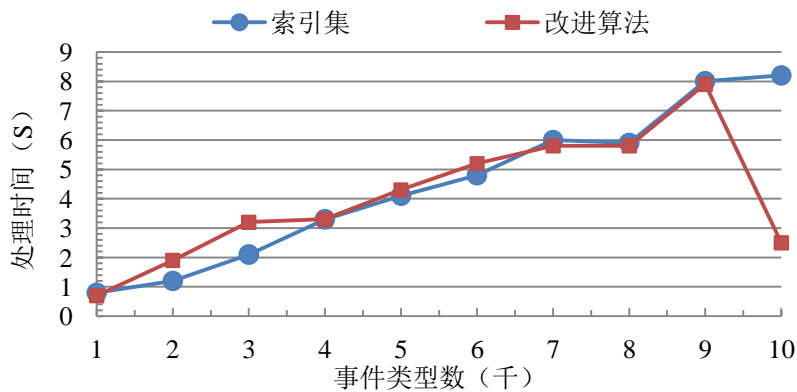


图 5-5 算法效率对比实验 2 示例

5.4 算法正确性验证

在上一节中基于不同的参数对不同的算法的效率进行了对比，本文提出的算法不止在算法效率上进行了改进，对比索引集算法来讲也改进了算法的正确性，即能挖掘出全部的频繁模式。所以这一节中将对三种算法的正确性进行对比，正确性的对比主要体现在挖掘的频繁模式数量上。在本文中无法得到应用的数据集中具体的频繁模式数目，然而在前文的叙述中已经说明本文改进得算法和枚举

树算法都能得到所有的频繁模式，因此如果在多次实验中改进的算法和枚举树算法能够挖掘出同样数量的频繁模式，就说明本文提出的算法能够挖掘出所有的频繁模式，也就证明了本文提出的算法是正确的。

5.4.1 支持度对正确率的影响

在实验中 $N_e=1000$, $N_p=2000$, $L_p=5$, $|D|=10$ 万, $L_c=20$ 。Minsupport 的范围从 0.05 到 0.1, 步长为 0.01。实验结果如图 5-6 所示，图中横坐标为最小支持度，纵坐标为挖掘出的频繁模式是数量。从图 5-6 中可以看出三种算法挖掘出的频繁模式数目都是随着支持度的增大而减小的。另一方面本文提出的改进算法产生的频繁模式数目与枚举树算法产生的频繁模式数目相等，而索引集算法产生的频繁模式数目略小，特别是在最小支持度较小时。因为在最小支持度比较小的时候，对频繁模式的出现率要求不高，从而更容易出现如图 1-3 所示的情况。这正说明了本文中提出的算法能够挖掘出所有的频繁模式。

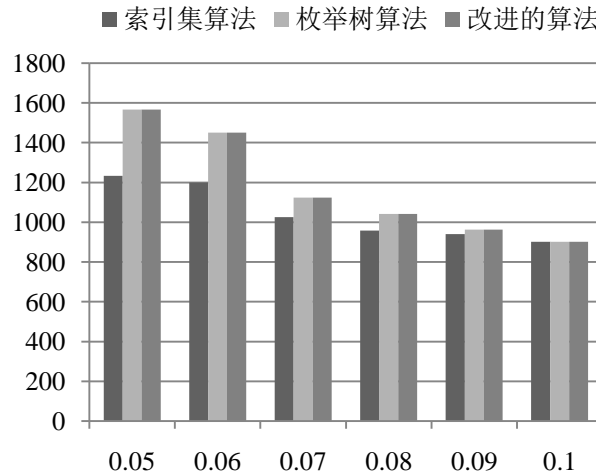


图 5-6 算法正确率对比示例 1

5.4.2 事件类型对正确率的影响

事件类型的数量是另一个对频繁模式数量产生较大影响的参数，事件类型越少，对于客户序列平均长度相同的模式来说，就更有可能多个相同的模式出现在同一客户序列中，这将严重影响索引集算法的挖掘正确率，相反更能体现出本文提出的算法的优越性。

在实验中各个参数设置如： $N_p=2000$, $L_p=5$, $|D|=10$ 万, $L_c=20$, minsupport=0.05。而事件类型的数量从 600 到 1000。实验结果如图 5-7 所示，图中横坐标为事件类型的数目，纵坐标为相应算法挖掘到的频繁模式的数量。

随着事件种类的增多，产生模式的数量也会增加。同时，因为事件种类增多，

而客户数列总数不变，因此同一客户序列中出现相同的模式的概率变小。索引集算法挖掘出的模式数目与改进的算法挖掘出的模式数目差距变小。

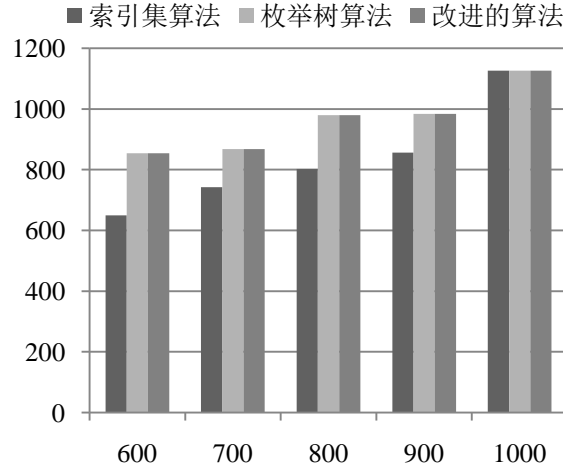


图 5-7 算法正确率对比示例 2

5.5 本章小结

本章对本文提出的理论进行了验证。主要从两个方面一算法的正确性与效率进行了验证。关于对比算法，因为索引集算法是近年来在这个领域中比较高效的算法，所以在效率验证上本文采用了索引集算法。而在正确率方面，毋庸置疑的，枚举树算法能够挖掘出所有的频繁模式，因此本文利用枚举树算法作为对比对算法进行正确性验证。

另外在本文中，改进的算法与对比算法都是在文[41]中仿真数据生成器的基础上运行的，在效率的对比上有着实际意义。在效率对比实验中，本文验证了各种参数对本文算法的影响，并且验证了在各种情况下算法在时间复杂度上的优越性。在正确性对比中，以枚举树算法作为标尺，本文中的算法能够挖掘出与枚举树算法相同数目的频繁模式，这也正说明了本文算法的正确性。

结 论

本文论述了带有时间间隔事件序列频繁模式挖掘的相关问题,首先描述了该问题的背景意义,然后总结了该问题在间隔事件序列挖掘领域的进展并给出了相关理论,然后在此基础上提出并验证了新的改进算法,最后对提出的算法进行了效率及正确性的验证。

本文最主要的贡献是提出了高效并且能够挖掘出所有频繁模式的算法,本文提出的算法是基于候选频繁模式生成—支持度计算模型的,主要提出了以下几种改进策略,其中前两项应用于频繁模式候选集生成阶段,后面两项应用于支持度统计阶段。

(1) k -频繁模式与 2 -频繁模式相结合产生 $k+1$ -频繁模式候选集,不同于传统的 k -频繁模式与 k -频繁模式相结合产生 $k+1$ -频繁模式候选集的过程中需要判断中间的 $k-1$ -频繁模式是否相同,本文中提出的策略在结合时只需比较已存储的最晚结束事件和最早发生事件即可,在关系复杂的时间间隔模式中这一策略能够减少很多比较操作。

(2) 修改 2 -频繁模式集合,根据在本文中证明的理论,能构成 $k+1$ -频繁模式的 2 -频繁模式至少出现在 $k-1$ 个 $k+1$ -频繁模式的长度为 k 的子模式中,利用这一点可以从 2 -频繁模式集合中去除很多无用的 2 -频繁模式。这一剪枝策略能够大大减少候选频繁模式的生成,而候选频繁模式是下一步支持度计算的输入,因此对这个算法的效率的影响是很大的。

(3) 构造索引,在产生频繁模式候选集的时候就将包含那些候选频繁模式的客户序列号记录下来,当支持度计算时只需遍历那些索引指向的客户序列即可。这一策略间接的减小了需要扫面的数据库规模又不会导致挖掘不完整。

(4) 在支持度计算中只扫描一次数据库,在第四章介绍了本文提出的支持度计算方法,与传统的支持度计算不同,这种方法只需要遍历一次数据库,无疑提高了整个算法的效率。

本文最后在算法效率及正确率两个方面对改进的算法及传统的算法进行了对比,证明本文提出的算法确实在效率上有了很大的提高。另外在对比算法效率的同时第五章也对各个参数对算法的影响进行了验证。

虽然本文提出了一种较以往算法高效的算法,但同时还有一些不足,主要表现为以下几个方面:

(1) 本文的重点放在算法效率的提高上,并没有考虑大容量数据情况,即数据不能一次性读入内存的情况。因此改进算法使之适应大容量数据是以后的工作

之一。

(2) 本文采用仿真数据集进行试验，数据的生成上并没有全面的考虑现实因素。因此算法并没有在一个真实场景下运行，这使得算法在现实生活中缺乏指导意义。

(3) 本文并未处理带有复杂属性的时间段事件，而一个时间段内发生的事情可能带有多种属性。例如在不同阶段发生的感冒病毒感染可能带有自己不同的病毒，即不同特征。在本文中只将事件泛化成一个数字，这一点和第二点都属于现实场景问题，这是以后的重点工作内容。

总之本文在简单的场景下对时间间隔事件序列频繁模式挖掘算法进行了深入的研究，并提出比较高效的算法，但是还存在一些问题需要日后的学习中更加深入仔细的研究。

参考文献

- [1] Agrawal R, Srikant R. Mining Sequential Patterns[C]. Proc.11th International Conference Data Engineer. (ICDE '95). Mar. 1995.
- [2] Dai H, Luo T, Nakagawa M. Using Sequential and Non-Sequential Patterns for Predictive Web Usage Mining Tasks[C]. Proc.IEEE International Conference Data Mining (ICDM '02). 2002: 669-672.
- [3] Yu C C, Chen Y L. Mining Sequential Patterns from Multi-Dimensional Sequence Data [J]. IEEE Trans., Knowledge and Data Engineer. Jan. 2005: 136-140.
- [4] Ayres J, Flannick J, Gehrke J et al: Sequential Pattern Mining Using A Bitmap Representation[C]. Proc.8th ACM SIGKDD International Conference Knowledge and Data Engineer. 2002: 429-435.
- [5] Papapetrou P, Benson G, Kollios G. Discovering Frequent Poly-Regions in DNA Sequences[C]. Proc. 6th International Conference ICDM workshops 2006. Dec.2006: 94-98.
- [6] Seno M, Karypis G. Slpminer: An Algorithm for Finding Frequent Sequential Patterns Using Length-Decreasing Support Constraint [C]. In Proc. of IEEE ICDM. 2002: 418-425.
- [7] Srikant R, Agrawal R. Mining Sequential Patterns: Generalizations and Performance Improvements [M]. Proc. of 5th International Conference on Extending Database Technology. 1996: 3-17 (An extended version is the IBM Research Report RJ 9994).
- [8] Han J, Pei J, Mortazavi-Asl B et al. FreeSpan: Frequent Pattern-Projected Sequential Pattern Mining[C]. Proc. of 6th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. 2000: 355-359.
- [9] Lin M Y, Lee S Y, Wang S S. DELISP: Efficient Discovery of Generalized Sequential Patterns by Delimited Pattern-Growth Technology [M]. Proc. of 6th Pacific-Asia Conference on Knowledge Discovery and Data Mining. 2002: 198-209.
- [10] Parthasarathy S, Zaki M J, Ogihara M et al. Incremental and Interactive Sequence Mining [M]. Proc. of 8th International Conference on Information and Knowledge Management. 1999: 251-258.
- [11] Wu S Y, Chen Y L. Mining Nonambiguous Temporal Patterns for Interval-Based Events [J]. IEEE Trans., Knowledge and Data Engineer. Jun. 2010: 742-758.
- [12] Papapetrou P, Kollios G, Sclaroff S et al. Discovering Frequent Arrangements of Temporal Intervals[C], Proc. of IEEE ICDM. 2009: 354-361.

- [13]Pei J, Han J, Mortazavi-Asl B et al. Prefix Span: Mining Sequential Patterns Efficiently by Prefix-Projected Pattern Growth[C]. Proc. 17th International Conference Data Eng. (ICDE '01). 2001: 215-224.
- [14]Bayardo R J. Efficiently Mining Long Patterns from Databases[C]. In: Proc. of ACM SIGMOD International Conference on Management of Data. 1998: 85–93.
- [15]Zaki M. SPADE: An efficient Algorithm for Mining Frequent Sequences[C]. Machine Learning. 2001: 31–60.
- [16]Winarko E, Roddick J F. Discovering Richer Temporal Association Rules from Interval-Based Data [M]. Proc. on Data Warehousing and Knowledge Discovery. 2008.
- [17]Lin M Y, Lee S Y. Fast Discovery of Sequential Patterns by Memory Indexing [M]. Proc. 4th International Conference on Data Warehousing and Knowledge Discovery, Aix-en-Provence. 2008: 150–160.
- [18]Dhaval P, Wynne H, Mong L L. Mining Relationships Among Interval-based Events[C]. Proc. of IEEE ICDE. 2008.
- [19]Brin S, Motwani R, Silverstein C. Beyond Market Baskets: Generalizing Association Rules to Correlations[C]. Proc. of the ACM SIGMOD Conference on Management of Data. Tucson, USA, 1997: 265-276.
- [20]Allen J F. Maintaining Knowledge about Temporal Intervals [J], Comm. ACM, vol. 26, no. 11. 1983: 832-843.
- [21]Kam P S, Fu A W C. Discovering Temporal Patterns for Interval-Based Events [C]. Proc. 2nd International Conference Data Warehousing and Knowledge Discovery (DaWaK '00). 2005.
- [22]Agrawal R, Srikant R. Fast Algorithms for Mining Association Rules[C]. Proc. 20th Int of VLDB. 1994: 487–499.
- [23]Tan P, Kumar V. Interestingness Measures for Association Patterns: A Perspective [J]. Technical Report TR00-036, Department of Computer Science, University of Minnesota. 2000.
- [24]Hilderman R J, Hamilton H J. Evaluation of Interestingness Measures for Ranking Discovered Knowledge [M]. Lecture Notes in Computer Science 2035. 2005: 247.
- [25]Kamber M, Shinghal R. Evaluating the Interestingness of Characteristic Rules[C]. Proc. of ACM SIGKDD. 1996: 263–266.
- [26]Tan P, Kumar V, Srivastava J. Selecting the Right Interestingness Measure for Association Patterns[C]. Proc. of 8th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. 2002: 183–192.
- [27]Brin S, Motwani R, Ullman J D et al. Dynamic Itemset Counting and Implication Rules for Market Basket Analysis[C]. Proc. of SIGMOD'97 Conference on

- Management of Data. 1997: 255-264.
- [28] Omiecinski E R. Alternative Interest Measures for Mining Associations in Databases[C]. IEEE Trans Knowledge Data Engineer 15(1). 2003:39–79.
- [29] Steinbach M, Kumar V. Generalizing the Notion of Confidence [J]. Knowledge and Information System 12(3). 2007:279–299.
- [30] Hilderman R, Hamilton H. Knowledge Discovery and Interestingness Measures: A Survey [J]. Technical Report 99-04, Department of Computer Science, University of Regina. 1999.
- [31] Morzy T, Wojciechowski M, Zakrzewicz M. Efficient Constraint-Based Sequential Pattern Mining Using Dataset Filtering Techniques [C]. Proc. 5th International Baltic Conference Databases and Information Systems (DB&IS '02). 2002: 213-224.
- [32] 罗可, 吴杰. 关联规则衡量标准的研究控制与决策[J], 2003(5): 277-280.
- [33] Masseglia F, Cathala F, Poncelet P. The PSP Approach for Mining Sequential Patterns [M]. Proc. of 2nd European Symposium on Principles of Data Mining and Knowledge Discovery, Vol. 1510. 1998: 176-184.
- [34] Allen J, Ferguson G. Actions and Events in Interval Temporal Logic[J]. J Logic Computer. 1994: 531-579.
- [35] Han J, Pei J, Yin Y. Mining Frequent Patterns without Candidate Generation[C]. Proc. of ACM SIGMOD. 2000: 1–12.
- [36] Leleu M, Rigotti C, Boulicaut J F et al. GO-SPADE: Mining Sequential Patterns over Databases with Consecutive Repetitions[C]. Proc. of MLDM. 2003: 293–306.
- [37] Pei J, Han J, Wang W. Constraint-Based Sequential Pattern Mining in Large Databases[C]. Proc. of CIKM, 2002: 18–25.
- [38] Seno M, Karypis G. An Algorithm for Finding Frequent Sequential Patterns Using Length Decreasing Support Constraint[C]. Proc. of IEEE ICDM. 2002: 418–425.
- [39] Bayardo R J. Efficiently Mining Long Patterns from Databases[C]. Proc. of ACM SIGMOD. 1998: 85–93.
- [40] Wang J, Han J. Efficient Mining of Frequent Closed Sequences[C]. In: Proc. of IEEE ICDE. 2008: 79–90.
- [41] Winarko E, Roddick J F. Discovering Richer Temporal Association Rules from Interval-Based Data: Extended Report [M]. Proc. on Data Warehousing and Knowledge Discovery. 2008.
- [42] Yan X, Han J, Afshar R. CloSpan: Mining Closed Sequential Patterns in Large Databases [J]. Proc. of SDM. 2003.
- [43] Harms S, Deogun J, Tadesse T. Discovering Sequential Association Rules with

Constraints and Time Lags in Multiple Sequences[C]. International symposium on methodologies for intelligent systems (ISMIS). 2002: 432–442.

哈尔滨工业大学学位论文原创性声明及使用授权说明

学位论文原创性声明

本人郑重声明：此处所提交的学位论文《基于时间间隔的事件序列频繁模式挖掘算法研究》，是本人在导师指导下，在哈尔滨工业大学攻读学位期间独立进行研究工作所取得的成果。据本人所知，论文中除已注明部分外不包含他人已发表或撰写过的研究成果。对本文的研究工作做出重要贡献的个人和集体，均已在文中以明确方式注明。本声明的法律结果将完全由本人承担。

作者签字：刘亚男

日期：2012年1月5日

学位论文使用授权说明

本人完全了解哈尔滨工业大学关于保存、使用学位论文的规定，即：

(1) 已获学位的研究生必须按学校规定提交学位论文；(2) 学校可以采用影印、缩印或其他复制手段保存研究生上交的学位论文；(3) 为教学和科研目的，学校可以将学位论文作为资料在图书馆及校园网上提供目录检索与阅览服务；(4) 根据相关要求，向国家图书馆报送学位论文。

保密论文在解密后遵守此规定。

本人保证遵守上述规定。

作者签名：刘亚男

日期：2012年1月5日

导师签名：刘亚男

日期：2012年1月5日

致 谢

值此论文完成之际，谨向给予我关心即无私帮助的同学和老师表示诚挚的感谢。

首先我谨向我的导师张春慨副教授致以崇高的敬意和诚挚的感谢，感谢张老师在我的课题研究过程中对我耐心的指导以及指正。他和蔼可亲、诲人不倦，具有丰富的经验及渊博的学识。在我课题研究的过程中，张老师亲切的给予我教导，并且提出了很多建设性的意见，让我受益匪浅，开拓我的视野。不仅让我对课题有了深入的了解，并且也了解了很多课题外的有用知识。

感谢叶允明教授，在我论文的完整过程中给予了很多帮助，他耐心而细致的工作作风以及对工作的满腔热情给我留下很深的印象。

感谢我的同学张蕾、陈岩、兰钊、李源，以及我的师兄赵霖、闫学凯和黎聪，在我遇到问题的时候，同学与师兄能够亲切的毫不保留的和我讨论问题并指导我。在生活中同学们也给予了我很大帮助。

感谢我的父母，最难忘莫过父母恩，感谢父母对我从小到大的教育，感谢从小到大对我的支持，你们是我最大的精神动力。

感谢整个实验室的同学们，是你们构造了这个轻松的欢乐的学习环境。这个实验室对我来说就是一个家庭。

感谢深研院所有计算机院的同学们，有了你们我才能在收获知识的同时能够保持如此快乐轻松的心情。

最后感谢学校所有授业解惑的老师，有了你们的辛勤教导，我们才能有如今的收获。