一种基于逻辑的频繁序列模式挖掘算法

刘端阳 冯 建 李晓粉
(浙江工业大学计算机科学与技术学院 杭州 310023)

摘 要 传统的类Apriori频繁序列模式挖掘算法都是基于支持度框架理论,需要预先设定支持度阈值,而这通常需要较深的领域知识或大量的实践,因此目前仍没有一种很好的设定方法。同时,序列模式的挖掘结果往往数量很大且不易理解,可用性较低。针对上述问题,提出了一种基于逻辑的频繁序列模式挖掘算法即LFSPM算法,并首次在频繁序列模式挖掘算法中引入了逻辑的思想,通过逻辑规则过滤,大大优化了结果集。实验证明,该算法较好地解决了支持度设置问题及挖掘结果可理解性不高的问题。

关键词 频繁序列模式,数据挖掘,逻辑,支持度阈值
中图法分类号 TP311 **文献标识码** A **DOI** 10.11896/j.issn.1002-137X.2015.5.052

Logic-based Frequent Sequential Pattern Mining Algorithm

LIU Duan-yang FENG Jian LI Xiao-fen
(College of Computer Science and Technology, Zhejiang University of Technology, Hangzhou 310023, China)

Abstract Traditional Apriori-like sequential pattern mining algorithms are based on the theoretical framework of support, which need pre-set support threshold, but this often requires in-depth domain knowledge or a lot of practice. Consequently, there is still no good way to set it. Meanwhile, the results of sequential patterns are too large to understand and apply. To solve these problems, this paper presented a logic-based frequent sequential pattern mining algorithm LFSPM, and introduced the thought of logic into frequent pattern mining process for the first time. Through using logical rules to filter, it optimizes the result sets greatly. Experiments show good performance of the proposed approach to solve these problems.

Keywords Frequent sequential pattern, Data mining, Logic, Support threshold

1 引言

频繁序列模式挖掘是数据挖掘中非常重要的一个研究领域,是由Rakesh Agrawal和Ramakfishnan Srikant针对超市中购物篮数据的分析提出来的[1]。它与传统关联规则挖掘的不同之处在于:频繁序列模式挖掘针对的目标数据都是带有时间属性的序列数据,这种数据在我们日常生活当中是很常见的,比如信用卡消费序列、城市交通数据序列、大型超市客户的购物序列等等,这些都是序列模式挖掘分析的目标数据,有重大的商业价值。

频繁序列模式的挖掘主要是基于大项集的挖掘,它是经典的关联规则挖掘算法Apriori的变形。这类算法目前存在两个主要问题:第一,支持度阈值设定问题,即传统类Apriori算法是基于支持度框架的,必须预先设定一个最小支持度阈值作为判断是否为频繁模式的标准,然而一般情况下用户对支持度阈值并没有准确的认识,在挖掘过程中主要通过多次试探或丰富的经验来设定,缺少统一的评判标准,阈值设定就成为了这类算法的难点;第二,挖掘的规则数量庞大,挖掘结果对于用户来说难以理解。具体来讲,如果序列模式P是频繁的,则P的全部子序列模式也都是频繁的,反之亦然。虽然类Apriori算法可以在挖掘过程中对候选集进行有效剪枝,但它同时也导致了序列模式结果集的规模呈指数级增长,大大增加了用户理解序列模式结果集的难度。为此,本文针对这两个问题,首次在频繁序列模式挖掘中引入了逻辑的思想,降低了挖掘结果对支持度阈值的依赖性,同时压缩了规则集的规模,较大地提高了规则集的可理解性和可用性。

2 研究现状

为了减小序列模式的挖掘结果集,根据不同的应用需求,出现了各类的序列模式压缩算法,其中较为常见的是闭合的序列模式挖掘算法[2-5]。闭合序列模式挖掘算法的挖掘结果能够完整地表达出序列模式全集的信息,但是结果数量却远小于全集的数量,因此是对序列模式全集的一个无损压缩。此外,极大序列模式挖掘也是序列模式压缩算法的一个研究热点[6,7]。一个频繁序列模式是极大序列模式当且仅当该序列模式的任何超序列模式都不是频繁序列模式。极大序列模式挖掘是对序列模式全集的一个高度压缩,但是它却丢失了子集支持度信息,是一个有损的压缩方法。此外,针对不同的需求还出现了其他扩展的序列模式压缩算法。文献[8]首次提出了用概括项集模式来对频繁模式进行压缩,它从全部频

到稿日期:2014-06-05 返修日期:2014-08-19 本文受浙江省自然科学基金(LY14F020018),国家自然科学基金(61202204)资助。
刘端阳(1975-),男,博士,副教授,主要研究方向为数据挖掘、分布式计算,E-mail:ldy@zjut.edu.cn;冯 建(1989-),男,硕士,主要研究方向为数据挖掘、人工智能;李晓粉(1989-),女,硕士,主要研究方向为数据挖掘、人工智能。

繁项集中找出 k 个最具有代表性的项集来概括整个频繁项集的集合。这 k 个代表模式的特点是不但基本覆盖了所有的频繁模式，并且对它们的支持度有一个很好的近似。文献[9]提出了一种在有噪音的环境中挖掘序列模式精简集的算法，它采用了一种不需要保留候选序列模式的方法来检查最大序列模式，并且采用更高效率的剪枝技术。而文献[10]利用频繁模式和支持度的信息来构造马尔可夫随机域模型(MRF)，此模型可以起到对频繁模式结果进行压缩的作用。文献[11]提出基于聚簇的挖掘近似模式的算法 ApproxMAP，它可以从序列数据库中挖掘出更长的频繁序列模式，从而展现给用户更少的模式。文献[12]提出了 CSP 算法，它扩展了概括项集模式的方法，提出了在序列模式中的概要模式的概念，并挖掘出 k 个最有代表性的序列模式。文献[13]提出在模式聚类函数的基础上生成一个压缩的偏序(Partial Order)的算法，实验结果显示该算法可以对频繁序列模式进行高效、高质量的压缩，可以得到数量更少、信息量更大的模式，从而提高发现的频繁访问序列的兴趣性。然而这些算法都或多或少存在一些问题，只能适应某一特定场景；同时针对频繁序列模式挖掘中存在的支持度阈值设置问题，仍然缺少一种有效的方法。

文献[14]将关联规则挖掘与逻辑等价相结合，提出了一种基于逻辑思想的关联规则挖掘算法，它不需要提供最小支持度阈值，通过逻辑等价规则对一些不符合逻辑的结果集进行筛选，提高了结果集的置信度和可用性。由于关联规则的挖掘与频繁序列规则挖掘有一定相似之处，因此，本文借鉴这一方法将逻辑的思想引入到频繁序列模式挖掘过程中。但由于序列数据带有时间属性，且数据结构较关联规则的目标数据更加复杂，本文针对序列数据进行了合理的设计，提出了一种基于逻辑的频繁序列模式挖掘算法 LFSPM(Logic-based Frequent Sequencial Patterns Mining)，该算法较好地解决了频繁序列模式挖掘中存在的支持度阈值设置问题及结果集可用性不高的问题。

3 逻辑映射关系

如何定义合理的最小支持度是频繁序列模式挖掘待解决的主要问题之一。尽管有研究已经证实合理的最小支持度阈值是存在的，但是它却很难被发现。Sim 等[14]提出的相干规则(coherent rule)挖掘算法是一种基于逻辑特性的关联规则挖掘算法，该方法通过使用逻辑的特性，可以在没有支持度阈值的情况下发现项目集之间的关系。它将关联规则映射为等价逻辑，每一条从关联规则到等价逻辑的映射必须满足表 1 中的条件。

表 1 规则映射条件

等价关系	$p \equiv q$	$\neg p \equiv \neg q$
关联规则	$X \rightarrow Y$	$\neg X \rightarrow \neg Y$
关联规则真假	满足的条件	
T	$X \rightarrow Y$	$\neg X \rightarrow \neg Y$
F	$X \rightarrow \neg Y$	$\neg X \rightarrow Y$
F	$\neg X \rightarrow Y$	$X \rightarrow \neg Y$
T	$\neg X \rightarrow \neg Y$	$X \rightarrow Y$

在表 1 中 X 和 Y 是两个项目，一条关联规则 $X \rightarrow Y$ 映射为等价规则 $p \equiv q$，当且仅当(1) $X \rightarrow Y$ 为真，(2) $X \rightarrow \neg Y$ 为假，(3) $\neg X \rightarrow Y$ 为假，(4) $\neg X \rightarrow \neg Y$ 为真。当被应用到多条记录时，关联规则 $X \rightarrow Y$ 映射为等价规则 $p \equiv q$，当且仅当(1) $Sup(XY) > Sup(X \rightarrow Y)$，(2) $Sup(XY) > Sup(\neg XY)$，(3) $Sup(\neg X \rightarrow Y) > Sup(\neg X \rightarrow Y)$，(4) $Sup(\neg X \rightarrow Y) > Sup(\neg XY)$，这 4 条规则标识如表 2 所列。

表 2 规则关联表

同时出现的频率		结论 Y	
		Y	$\neg$Y
前提 X	X	$Q_1 = Sup(XY)$	$Q_2 = Sup(X \rightarrow Y)$
	$\neg$X	$Q_3 = Sup(\neg XY)$	$Q_4 = Sup(\neg X \rightarrow Y)$

相干规则是指如果规则 $X \rightarrow Y$ 存在，那么规则 $\neg X \rightarrow \neg Y$ 也同时存在。因此，根据相干规则的特性，对于特定结论 Y 的相干规则的挖掘在文献[14]中已经被提出。算法思路为：第一，算法先从给定的交易数据库中找到所有的项目集合 I，其中 $I = X \cup Y$；第二，把幂集合 A(X 的所有分布情况)映射为二进制系统，最后计算 X 与 Y 同时出现的频繁情况来得出相干规则，前提条件是必须满足 $Sup(XY) > Sup(X \rightarrow Y)$。但是，在序列数据中，由于特殊的时间属性，相干规则的特性无法被应用，即如果规则 $X \rightarrow Y$ 存在，那么 $\neg X \rightarrow \neg Y$ 不一定存在。因此，在序列数据中只需考虑 $X \rightarrow Y$，本文只使用(1) $Sup(XY) > Sup(X \rightarrow Y)$，(2) $Sup(XY) > Sup(\neg XY)$ 两条规则进行逻辑过滤，通过逻辑优化结果集，减小冗余规则的时间消耗，从而提高时间效率，同时减少结果的规模，增加可用性。

4 相关定义

令 $I = \{i_1, i_2, \cdots, i_n\}$，其中 $i_j (j = 1, 2, \cdots, n)$ 是不同的项，项的集合 I 为项集。

定义 1(序列) 一个序列 S 即是项集的有序列表，记为 $S = (s_1, s_2, \cdots, s_n)$，其中 $s_i (i = 1, 2, \cdots, m)$ 是一个项集，也称一个元素，且 $s_i \subseteq I$。

定义 2(序列包含) 一个序列 $S_a (a_1, a_2, \cdots, a_n)$ 被另外一个序列 $S_b (b_1, b_2, \cdots, b_m)$ 所包含，当且仅当如果存在整数 $1 \leqslant j_1 < j_2 < \cdots < j_n < m$，使得 $a_1 \subseteq b_{j1}, a_2 \subseteq b_{j2}, \cdots, a_n \subseteq b_{jm}$。相应地，我们将 S_a 称作 S_b 的子序列(Subsequence)，S_b 称作 S_a 的超序列(Supersequence)，记为 $S_a \subseteq S_b$，如两序列长度不等，则成真包含关系，记 $S_a \subset S_b$。

定义 3(序列数据库) 序列数据库 SDB(Sequence DataBase)是元组(Sids)的集合，其中 Sid 是对应序列的序列号。元组的个数称为序列数据库的大小，记为 $|SDB|$，如表 3 所列。

表 3 序列数据库样例

序列号(Sid)	序列(Sequence)
S_1	⟨(a,f),(d),(e),(a)⟩
S_2	⟨(e),(a),(b)⟩
S_3	⟨(e),(a,b,f),(b,d,e)⟩

定义 4(序列支持度) 一个序列 S_a 在序列数据库 SDB 中的绝对支持度即 SDB 中包含 S_a 的元组的数目，记为 $sup(S_a)$；相对支持度为 SDB 中包含 S_a 的元组在整体数据库元组中所占的百分比，即 $sup(S_a)/|SDB|$。本文若不特别申明，均指相对支持度。

定义 5(频繁序列模式) 给定一最小支持度阈值 min_sup，若序列 S 的支持度不小于 min_sup，则称序列 S 为频繁序列模式，其中，长度为 l 的序列模式记为 l-模式。所有的频繁序列的集合记为 FS。

5 算法描述

这一部分主要描述本文提出的 LFSPM 算法，用以挖掘序列数据库中高可用的频繁序列模式。

算法 1 LFSPM 算法

输入：序列数据库 D，最小支持度 min_sup

输出：一个高可用的频繁序列模式集合

1. 扫描整个序列数据库，并计算每一个项目 i_j 的支持度，并与 min_sup 比较，得到 1 项频繁序列模式 L_1。
2. 如果 L_1 不为空，则 L_1 为一项频繁序列模式。则设 i=1，根据长度为 i 的种子集 L_i，通过连接操作和剪切操作生成长度为 i+1 的候选序列模式 C_{i+1}。
   2.1 连接操作：如果去掉序列模式 s_1 的第一个项目与去掉序列模式 s_2 的最后一个项目所得到的序列相同，则可以将 s_1 与 s_2 进行连接，即将 s_2 的最后一个项目添加到 s_1 中。
   2.2 剪切阶段：若某候选序列模式的某个子序列不是序列模式，则此候选序列不可能是序列模式，将它从候选序列模式中删除。
3. 扫描序列数据库，计算每个候选序列模式的支持数，产生长度为 i+1 的序列模式。
4. 对 i+1 序列模式中的每一项，产生所有{X,Y}项目集，使得 X,Y 分别是 s_{i+1} 的子集，$X \cup Y = s_{i+1}$ 并且满足 $X \cap Y = \emptyset$，同时满足任意 $a \in X, b \in Y$，a 在 s_{i+1} 中的位置在 b 的前面。
5. 对每一个项目集(X,Y)，设置前置条件为 X，结论为 Y，并计算 3 个项目集的支持数：Q_1：Sup(XY)，Q_2：Sup(→XY)，Q_3：Sup(X→Y)。注意 Q_2、Q_3 可以分别通过(Sup(X)−Q_1)、(Sup(Y)−Q_1)来计算。
6. 验证每一个项目集(X,Y)是否满足 $Q_1 > Q_2$，$Q_1 > Q_3$，如果符合，则将该 i+1 项集加入到 L_{i+1} 中去。
7. 如果 L_i 不为空，则 i=i+1，$L_{all} = L_{all} \cup L_i$，回到第 2 步重复执行；否则，继续下一步。
8. 生成频繁序列模式集。对每一项目集 $s_i \subset L_{all}$，产生它所有的规则集(X→Y)，满足 X,Y 分别是 s_i 的子集，并且 $X \cap Y = \emptyset$，$X \cup Y = s_i$，同时满足任意 $a \in X, b \in Y$，a 在 s_i 中的位置在 b 的前面。计算它们的置信度 conf(X→Y)=Sup(XY)/Sup(X)。
9. 输出频繁序列模式集。

在这一节将给出一个例子来更加形象地解释本文提出的 LFSPM 算法的具体流程。假设现在有 7 个项目在一个序列数据库中，具体数据见表 4。

表 4 序列数据库示例

Sid	Sequence
1	⟨(i_1),(i_2),(i_3,i_4)⟩
2	⟨(i_1),(i_3)(i_4)⟩
3	⟨(i_1),(i_2),(i_3),(i_5)⟩
4	⟨(i_1),(i_5)⟩
5	⟨(i_1),(i_3),(i_5)⟩
6	⟨(i_3),(i_4,i_5),(i_6)⟩
7	⟨(i_1),(i_4),(i_5)⟩
8	⟨(i_2),(i_4,i_5)⟩
9	⟨(i_2),(i_3),(i_4,i_5)⟩
10	⟨(i_3,i_4),(i_5)⟩

在这个例子中，最小支持度设为 0.2。本文提出的算法的具体计算步骤如下：

1. 扫描整个序列数据库，并计算每一个项目 i_j 的支持度，并与 *min_sup* 比较，得到 1 项频繁序列模式 L_1，见表 5。

2. L_1 不为空，对 L_1 中项集进行连接和剪枝操作，得到候选序列模式 C_2，例如 i_1 与 i_2 连接生成 2 项集⟨(i_1),(i_2)⟩。

3. 扫描数据库，计算每个候选序列模式的支持数，产生长度为 2 的序列模式，得到 2 项频繁序列模式 L_2，见表 6。

4. 对 2 序列模式中的每一项，产生所有{X,Y}项目集，使得 X,Y 分别是 s_i 的子集，并且满足 $X \cap Y = \emptyset$，例如，⟨(i_1),(i_3)⟩可分为{(i_1),(i_3)}。

5. 对每一个项目集(X,Y)，设置前提条件为 X，结论为 Y，并计算 3 个项目集的支持数。例如计算{(i_1),(i_3)}，Q_1：$Sup(XY)=0.4$，Q_2：$Sup(\rightarrow XY)=0.6-0.4=0.2$，$Q_3$：$Sup(X \rightarrow Y)=0.7-0.4=0.3$。

6. 验证每一个项目集(X,Y)是否满足 $Q_1 > Q_2$，$Q_1 > Q_3$，例如{(i_1),(i_3)}经验证满足以上条件，将其加入到 L_2 中。

7. 当 $i=3$ 时，L_i 为空，继续下一步。

8. 生成频繁序列模式集。对每一序列模式 $s \subset L_{all}$，产生它所有的规则集(X→Y)，满足 X,Y 分别是 s 的子集，并且 $X \cap Y = \emptyset$，同时满足任意 $a \in X, b \in Y$，a 在 s 中的位置在 b 的前面，如 $i_1 \rightarrow i_3$，将规则 $i_1 \rightarrow i_3$ 加入结果集。

9. 输出频繁序列模式集，见表 7。

表 5 1 项频繁序列模式

ID	支持度	ID	支持度
i_1	0.6	i_4	0.7
i_2	0.4	i_5	0.8
i_3	0.7		

表 6 2 项频繁序列模式

序列模式	支持度	序列模式	支持度
⟨(i_1),(i_2)⟩	0.2	⟨(i_2),(i_4)⟩	0.3
⟨(i_1),(i_3)⟩	0.4	⟨(i_2),(i_5)⟩	0.3
⟨(i_1),(i_4)⟩	0.3	⟨(i_3),(i_4)⟩	0.3
⟨(i_1),(i_5)⟩	0.4	⟨(i_3),(i_5)⟩	0.3
⟨(i_2),(i_3)⟩	0.3	⟨(i_4),(i_5)⟩	0.2

表 7 所有规则置信度

候选规则	置信度	序列模式	置信度
$i_1 \rightarrow i_3$	0.67	$i_3 \rightarrow i_5$	0.71
$i_1 \rightarrow i_5$	0.67		

6 实验结果与分析

6.1 实验环境与实验数据

实验环境为 Intel core i3-3120M 2.5GHz CPU，4GB DDR3 内存，操作系统为 Windows7 系统，实验采用 java 语言基于 eclipse 编程环境实现。实验测试数据集采用 IBM 数据生成器随机生成，该生成器是目前数据挖掘相关研究中使用广泛和权威的生成器，可以通过设置参数生成不同特性的数据集。本次实验主要使用 3 个模拟数据集，具体模拟数据的参数如表 8 所列。实验数据的具体细节如表 9 所列，本实验中最小支持度阈值为变量。

表 8 模拟数据参数

参数	定义	数据 1	数据 2	数据 3
-tlen	平均每笔商品数	3	5	10
-nitems	项目数	50	100	300
-ncust	客户数	200	1000	5000
-slen	平均每序列长度	10	10	10

表 9 实验数据描述

数据库	交易数	商品数
模拟数据 1	9145	50
模拟数据 2	47661	100
模拟数据 3	477997	300

6.2 实验分析

实验首先比较 LFSPM 算法与典型类 Apriori 算法——GSP 算法在不同支持度下的实验结果，数据集为表 9 所列的模拟数据 2；然后比较不同数据集使用 LFSPM 算法运行效率情况，数据集为模拟数据集 1、2、3。

表 10 描述了两个算法时间消耗、规则数、平均置信度的对比情况，最小支持度从 0 变化到 50%。可以看出，当支持度较大时，两算法的运行时间相差不大，但是 LFSPM 算法运行时间优于 GSP 算法；从支持度小于 20%开始，两者运行时间相差巨大。详情可见图 1，从图可以看出在相同的支持度下，LFSPM 算法拥有更高的执行效率，且当支持度较小时，LFSPM 算法的运行时间远远少于 GSP 算法；同时，从图 2 可以看出，随着支持度的降低，GSP 算法的结果规则数呈指数级增长，而 LFSPM 算法的规则数增长较为缓慢，当支持度小于 30%时，规则数保持不变为 64。因此，LFSPM 算法对支持度阈值的依赖性远小于 GSP 算法，它有效地解决了传统类 Apriori 算法对支持度阈值的依赖问题，同时保证了算法结果的质量。

表 10 实验结果对比

算法	最小支持度(%)	规则数	平均置信度(%)	执行时间(s)
LFSPM 算法	50	18	69.10	0.24
	40	46	64.84	0.52
	30	64	61.87	1.06
	20	64	61.87	1.81
	10	64	62.9	3.73
	0	64	62.9	6.48
GSP 算法	50	22	67.56	0.26
	40	84	60.23	0.53
	30	399	52.45	1.97
	20	2042	42.32	26.38
	10	34985	30.13	2343.20

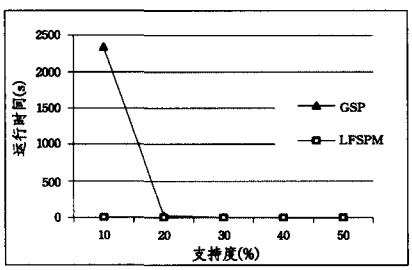

图 1 LFSPM 算法与 GSP 算法运行时间比较

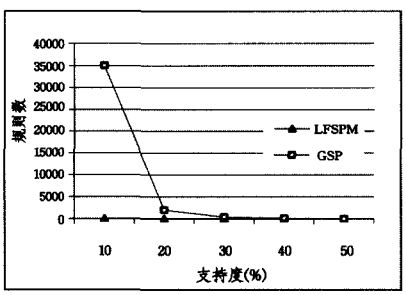

图 2 LFSPM 算法与 GSP 算法结果规则数比较

图 3 描述了两个算法在不同支持度下产生的规则的平均置信度的对比情况，从中可以看出，在相同支持度下，LFSPM 算法产生的规则集平均置信度都比 GSP 算法高，而且随着支持度的减小，两者的差距越来越大。从实验结果可以得出，LFSPM 算法得出的结果拥有更高的质量，尤其是当支持度阈值比较小的时候 。图 4 描述了 LFSPM 算法在不同规模的数据集下算法运行时间随支持度变化的对比情况。随着支持度的增大，算法运行的时间逐渐减少，当支持度较大时，不同规模的数据集的运行时间相差不大，当支持度较小时，数据规模越大，运行时间越长，但相对传统 GSP 算法仍然有较大的提升。

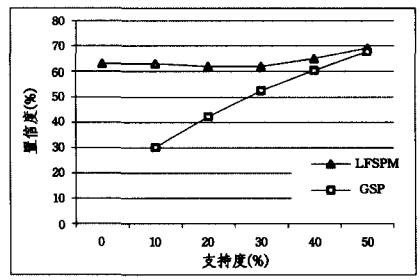

图 3 LFSPM 算法与 GSP 算法的结果置信度比较

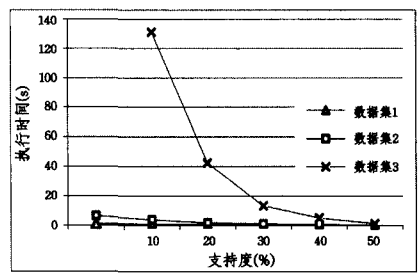

图 4 不同数据集的运行时间比较

结束语 本文提出了一种基于逻辑的频繁序列模式挖掘算法，首次在频繁序列模式挖掘中引入了逻辑的思想，通过在挖掘过程中加入逻辑过滤规则，去除掉大量不符合逻辑的、无用的规则集，使序列的结果集大大优化，从而减少了算法的时间消耗，而且提高了结果的质量；同时，有效地解决了支持度阈值设置问题，降低了算法对它的依赖性。

参 考 文 献

[1] Agrawal R, Srikant R. Mining sequential patterns [C]// Proceedings of the Eleventh International Conference on Data Engineering, 1995. IEEE, 1995: 3-14

[2] Yan X, Han J, Afshar R. CloSpan: Mining closed sequential patterns in large datasets[C]// Proceedings of SIAM International Conference on Data Mining. 2003: 166-177

[3] Wang Jian-yong, Han Jia-wei. BIDE: efficient mining of frequent closed sequences[C]// Proceeding of the 2004 International Conference on Data Engineering. Boston, 2004: 79-90

[4] 童咏昕，张媛媛，袁玫，等. 一种挖掘压缩序列模式的有效算法[J]. 计算机研究与发展，2010，47(1)：72-80

[5] Chang L, Wang T, Yang D, et al. Seqstream: Mining closed sequential patterns over stream sliding windows [C] // Eighth IEEE International Conference on Data Mining, 2008 (ICDM' 08). IEEE, 2008: 83-92

[6] Luo C, Chung S M. A scalable algorithm for mining maximal frequent sequences using a sample[J]. Knowledge and Information Systems, 2008, 15(2): 149-179

[7] Chedyeams R, Pascal P, Maguelonne T. Speed: mining maximal sequential patterns over data streams[C]// Proceedings of the 3rd International IEEE Conference on Intelligent Systems. London: IEEE, 2006: 546-552

[8] Yan X, Cheng H, Han J, et al. Summarizing itemset patterns: a

profile-based approach[C]// Proceedings of the Eleventh ACM SIGKDD International Conference on Knowledge Discovery in Datamining. ACM, 2005: 314-323

[9] 王涛. 在有噪音的环境中挖掘序列模式精简基[J]. 华中科技大学学报: 自然科学版, 2006, 34(6): 35-38

[10] Kum H C, Pei J, Wang W, et al. ApproxMAP: Approximate mining of consensus sequential patterns[C]// Third SIAM International Conference on Data Mining (SIAM-DM). 2003: 311-315

[11] Chang L, Yang D, Tang S, et al. Mining compressed sequential patterns[M]// Advanced Data Mining and Applications. Springer Berlin Heidelberg, 2006: 761-768

[12] Xin D, Cheng H, Yan X, et al. Extracting redundancy-aware top-k patterns[C]// Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. ACM, 2006: 444-453

[13] 程舒通, 徐从富, 但红卫. 基于偏序压缩技术的频繁序列模式数据挖掘[J]. 计算机工程与应用, 2008, 44(3): 192-194

[14] Sim A T H, Indrawan M, Zutshi S, et al. Logic-based pattern discovery[J]. IEEE Transactions on Knowledge and Data Engineering, 2010, 22(6): 798-811

[15] Chen Chun-hao, Lan Guo-cheng, et al. Mining high coherent association rules with consideration of support measure[J]. Expert Systems with Applications, 2013, 40(16): 6531-6537

(上接第 233 页)

邻域优化以及优化后(采用剪枝与邻域优化)的时间开销来对优化策略的有效性进行评价。

如图 8 所示,在实验中可能世界数量为 5000 时,时间开销从大到小的顺序为:优化前、优化后、基于邻域优化、基于剪枝。因为当数据量太少时,使用剪枝的方法并没有减少需要计算的对象,反而增加了计算网格所需的时间,并没有把时间开销降下来。但是当不确定数目逐渐增多时,从图 9 中可以直观看出,可能世界数量为 1050(千)时,通过优化前、基于剪枝、基于邻域查询、优化后来计算 ULOF 值所需时间分别为 23892.3s、20874.33s、16983.77s、9996.87s,其中优化后的 ULOF 算法只用了优化前算法 41.8%的时间。从实验可知,本文提出的 ULOF 算法是可行的而且有效地降低了离群点检测的计算复杂度。

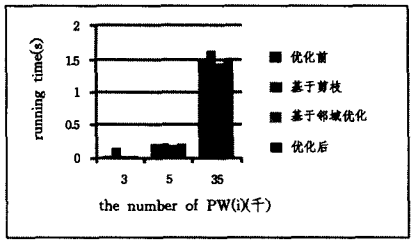

图 8 ULOF 算法优化效率对比图(k=10)

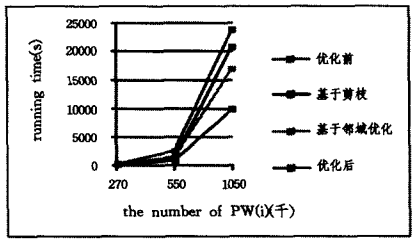

图 9 ULOF 算法优化效率对比图(k=10)

结束语 本文提出了基于密度的不确定数据的局部离群点检测,提出了 ULOF 算法中各参数的定义和并作详细介绍,由局部密度的大小来确定不确定数据集中点的异常程度,并提出了邻域查询优化以及网格剪枝优化策略,通过实验证实了 ULOF 算法是有效且可行的。经验证,算法能够有效提高离群点检测精度,降低计算复杂度,且参数 k 的作用重大,直接影响到邻近点的邻域查询范围。需要继续研究各参数与运行结果之间的关系,提出合理的各参数选择方法,同时也需要对不确定性数据的 TOP-K 做进一步研究以便于快速找到异常度最高的离群点组合。

参 考 文 献

[1] Garces H, Sbarbaro D. Outliers Detection in Environmental Monitoring Databases [J]. Engineering Applications of Artificial Intelligence, 2011, 24(2): 341-349

[2] Jampani R, Xu F, Wu M. A Monte Carlo Approach to Managing Uncertain Data [C]// Proc. SIGMOD, 2008: 687-700

[3] Widom J. Trio: A System for Integrated Management of Data, Accuracy, and Lineage [C]// Proc. of the Second Biennial Conference on Innovative Data Systems Research. Asilomar, 2005: 262-276

[4] Li F F, Yi K, Jestes J. Ranking Distributed Probabilistic Data [C]// Proc. SIGMOD Conference. ACM New York, NY, USA 2009: 361-374

[5] 张晓峰, 王丽珍, 陆叶. 一种基于属性加权的不确定 K-means 聚类算法[J]. 计算机研究与发展, 2009, 46(10): 504-508

[6] Tsang S, Kao B, Yip K Y. Decision Trees for Uncertain Data [C]// The 25th International Conference on Data Engineering New Jersey: IEEE Press, 2009: 441-444

[7] Kriegel H P, Pfeifle M. Density-based Clustering of Uncertain Data[C]// ACM Knowledge Discovery and Data Mining. ACM Press, 2005: 672-677

[8] Aggarwal C C. Managing and Mining Uncertain Data[J]. Advances in Database Systems, 2009(35): 75-89

[9] Ngai W K, Kao B, Chui C K, et al. Efficient Clustering of Uncertain Data[C]// ICDM, IEEE Computer Society, 2006: 436-445

[10] Qin B, Xia Y, Li F. A Bayesian Classifier for Uncertain Data [C]// SAC, ACM, 2010: 1010-1014

[11] 于浩, 王斌, 肖刚, 等. 基于距离的不确定离群点检测[C]// NDBC2009(第 26 届中国数据库学术会议论文集(A 集, 2009)) 2009: 15-18, 143-150

[12] Charu C, Aggarwal, Philip S Y. Outlier Detection with Uncertain Data [R]. IBM T. J, Watson Research Center. 2008

[13] Wang B, Xiao G, Yu H, et al. Distance-based Outlier Detection on Uncertain Data [C] // CIT (1). IEEE Computer Society, 2009: 293-298

[14] Liu B, Yin J, Xiao Y, et al. Exploiting Local Data Uncertainty to Boost Global Outlier Detection[C]// ICDM. IEEE Computer Society, 2010: 304-313

[15] Jiang B, Pei J. Outlier Detection on Uncertain Data: Objects, Instances, and Inferences [C] // ICDE. IEEE Computer Society, 2011: 422-433

[16] Liu Jing, Deng Hui-fang. Outlier Detection on Uncertain Data Based on Local Information [J]. Knowledge-based System, 2013, 7(51): 60-71

[17] 李健, 阎保平, 李俊. 基于记忆效应的局部异常检测算法[J]. 计算机工程, 2008, 34(12): 4-6