文章编号:1001 - 9081(2008)06 - 1450 - 04

基于 XML 的完全频繁查询模式挖掘算法

陈超祥[1],叶时平[1],华 成[2],金林樵[1]
(1. 浙江树人大学 信息科技学院,杭州 310015; 2. 浙江大学 计算机科学与技术学院,杭州 310027)
(ccx0725@126.com)

摘 要:使用树结构建模对 XML 查询进行研究,提出了一种基于树同构的查询包含检测方法。采用最右分枝扩展方法,系统地枚举查询模式树的同根子树。在枚举过程中,采用 Diffset 结构记录包含同根子树的事务集的查询事务标识,并给出挖掘算法 DiffFRSTMiner。实验结果证实了该算法合理、高效,并可以减少一定的内存开销。

关键词:XML;数据挖掘;频繁查询模式
中图分类号: TP301.6 **文献标志码**:A

XML-based mining algorithm of complete frequent query pattern

CHEN Chao-xiang[1], YE Shi-ping[1], HUA Cheng[2], JIN Lin-qiao[1]
(1. *College of Information Science and Technology, Zhejiang Shuren University, Hangzhou Zhejiang* 310015, *China*;
2. *College of Computer Science, Zhejiang University, Hangzhou Zhejiang* 310027, *China*)

Abstract: To study XML query with tree structure modeling, a query and detection method based on tree isomorphism was proposed, systematically enumerating the same root subtree of query pattern tree with the most right branch expansion. In the enumeration process, the Diffset data structure was used to record the query item logo of item set, and the DiffFRSTMiner mining algorithm was proposed. This item set includes the same root subtree. The experimental results prove that the algorithm is efficient, and can reduce definite memory overhead.

Key words: XML; data mining; frequent query pattern

0 引言

近年来,XML[1] 已成为 Internet 环境下数据表示和交换的标准,被广泛应用于搜索引擎、电子商务和电子数据交换等方面。而挖掘 XML 数据的用户频繁查询模式,将用户访问频繁的数据进行缓存,并采用对应的缓存替换策略,能大大提高对数据源的查询处理能力;挖掘用户的查询行为规律,得到对 XML 数据的频繁查询行为模式,并结合 XML 数据存储的代价指标,采取相应的映射和存储策略,可以大大降低存储代价,提高 XML 关系存储的性能。因此,基于 XML 数据的频繁树结构挖掘已成为数据挖掘研究的重要内容之一。

已有的频繁树结构挖掘算法主要有两类算法:Apriori 推演类算法[2-3]通过多次扫描事务集计算得到频繁项目集;另一类重要的算法是基于频繁查询模式树 FP-growth 的算法[4]。理论与实验表明,FP-growth 算法性能优于 Apriori 算法,并可克服 Apriori 算法存在的一些问题:如可能产生大量的候选集;以及生成候选项目时由于模式匹配引起巨大时空开销。

然而,现有 FP-growth 算法都采用子树嵌入的查询包含检测方法[5-6]。子树嵌入的包含检测大部分只考虑简单的结构挖掘,并不完全符合 XML 查询语义。为此,本文提出采用基于树同构的查询包含检测方法,即 XML 查询用树结构建模,采用基于最右分枝扩展的方法系统地枚举查询模式树(Query Pattern Tree, QPT)的同根子树(Rooted Subtree, RST)。在枚举过程中采用 Diffset 的数据表示方法记录包含同根子树的频繁查询模式事务集的查询事务标识,比直接用 Tidset 数据结构记录查询事务标识更高效,并可以减少内存开销,最后给出了挖掘算法 DiffFRSTMiner。

1 查询模式挖掘的相关概念与定义

计算一个同根子树的频繁度,需要判断该同根子树是否包含于查询模式树集合中的查询模式树。本文使用基于查询模式树包含的概念进行判断。

定义 1 查询模式树包含。给定两个查询模式树 $QPT = \langle V, E\rangle$, $QPT' = \langle V', E'\rangle$,如果存在一个映射 ϕ,满足如下条件,那么称 QPT 包含于 QPT',记做 $QPT \subseteq QPT'$:

1) $\phi(\mathrm{Root}(QPT)) = \mathrm{Root}(QPT')$;

2) $\forall v \in V; \exists v' \in V'$,使得 $v' = \phi(v)$,且 $v'.\mathrm{label} = v.\mathrm{label}$ 或者 $v'.\mathrm{label} = "*"$;

3) 对于 $(u,v) \in E$,必存在 $\phi(u')$, $\phi(v') \in V'$,且 $\phi(v')$ 为 $\phi(u')$ 的子节点或者后代节点。

查询模式树包含检测的实质是查询包含检测,即检测一个 XML 查询的结果是否是另一个查询结果的子集。对于给定的两个查询 q_1 和 q_2,如果 q_1 查询结果是 q_2 查询结果的一个子集,就认为查询 q_1 包含于查询 q_2。

定义 2 查询包含。给定两个 XML 查询 q_1 和 q_2,如果 q_1 对应的查询模式树 QPT_1 包含于查询 q_2 对应的查询模式树 QPT_2,那么 q_1 查询结果是 q_2 查询结果的一个子集。

最右分枝扩展方法指在模式树的最右分枝上进行扩展,无重复地枚举出所有的候选同根子树。采用最右分枝扩展方法,首先由枚举得到的候选频繁节点,枚举出所有 1 边 RST,

收稿日期:2007 - 12 - 04;**修回日期**:2008 - 01 - 31。 **基金项目**:浙江省科技计划重点项目(2007C21042)。

作者简介:陈超祥(1975 -),女,浙江浦江人,讲师,硕士,主要研究方向:数据挖掘、人工智能; 叶时平(1967 -),男,浙江丽水人,副教授,硕士,主要研究方向:GIS、网络计算; 华成(1974 -),男,新疆乌鲁木齐人,博士研究生,主要研究方向:XML 数据库、Agent 技术、普及计算; 金林樵(1962 -),男,浙江湖州人,副教授,主要研究方向:网络计算。

然后由 1 边 RST 生成 2 边 RST,再由 2 边 RST 生成 3 边 RST,直至生成所有的 n 边 RST(n 为所有生成的 RST 的最大边数)。

定义 3　查询模式树 RST 的最右分枝扩展。给定一个 k 边 RST,记为 RST^k,它的根节点编号为 $\mathrm{Root}(RST^k)=0$,最右叶节点编号为 $rml(RST^k)=k$。从根节点到节点最右叶节点的路径称为 RST^k 的最右分枝。

定义 4　直接前缀树。给定一个 k 边同根子树,记为 RST^k,$\forall p \geqslant 0, y=\mathrm{anc}^p(rml(RST))$,在节点 y 上添加一个标记为 x 的节点 v,得到 $k+1$ 边同根子树 RST^{k+1},v 的编号为 $rml(RST^{k+1})=k+1$,称 RST^k 经最右分枝扩展得到 RST^{k+1},将 RST^k 称为 RST^{k+1} 的直接前缀树。

定义 5　RST 等价类。给定一个 k 边同根子树,由该同根子树为直接前缀树经最右分枝扩展得到的 $k+1$ 边同根子树集合为一个等价类,表示为 $[P_{k+1}]$。

最右分枝扩展就是在一个直接前缀树的最右分枝上的节点进行扩展,生成新的同根子树,这些同根子树可以构成一个等价类。最右分枝扩展的过程就是生成等价类的过程。因此,最右分枝扩展也可称为基于等价类的扩展。基于等价类扩展生成所有同根子树的方法如下:

1) 从查询事务数据集中找出所有标记不同的频繁节点,生成其节点编号范围 *SList*,记录频繁节点出现的查询模式树 QPT;

2) 由节点开始,通过孩子扩展方式,生成 1 边 RST,并找出所有的频繁 1 边 RST;

3) 由 1 边 RST 开始,基于等价类进行扩展,按照孩子方式扩展和联结,生成 2 边 RST;

4) 以此类推,由 k 边 RST,基于等价类扩展生成 $k+1$ 边 RST,从而生成所有的 RST。

定义 6　同根子树的频繁度(Tidset)。频繁项目集挖掘中,为计算一个项目集的支持度,将记录事务集中包含该项目集的事务的标识集合,记为 Tidset,并为每个 RST 关联一个 Tidset,称同根子树的频繁度,记做 *RST. Tidset*。

若 RST 出现在某个查询模式树 QPT 中,则将该 QPT 的 id 插入 *RST. Tidset* 中,用 Tidset 记录所有包含该同根子树的 QPT 的标识。显然,RST 的频繁度等于 *RST. Tidset* 的大小,记为 $|RST.\ Tidset|$,由此可计算得到其支持度。

2　Diffset 原理描述

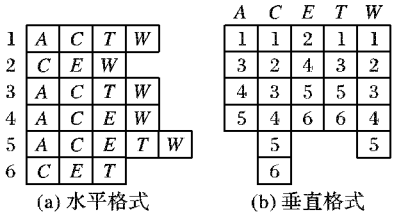
Diffset 最初由文献[7]提出,用于提高挖掘频繁项目集的性能。本文将引用文献[7]中的一个示例,结合该示例来说明 Diffset 的原理,以及如何利用 Diffset 来挖掘频繁项目集。

例 1　给定如表 1 所示的事务集,$I=\{A,C,E,T,W\}$ 是 5 个不同的项目,图 1 为一个关联规则挖掘中常用的数据表示格式,该格式可称为水平(横向)数据格式。在该格式中,对于一个事务,将该事务的标识以及该事务包含的项目集按图 1(a)方式进行记录。另一种记录项目集及包含它的事务标识的方法如图 1(b)所示,对于每个项目,记录包含该项目的所有事务标识,这种方式称为垂直(纵向)格式。

采用垂直格式时,对于一个项目,将包含该项目的事务标识集合用 Tidset 的数据结构进行记录。图 2 为利用 Tidset 计算项目集频繁度的示意图。该图中首先给出每个项目的 Tidset,如 A 的 Tidset 记为:*A. Tidset* $=\{1,3,4,5\}$,C 的 *Tidset* 记为:*C. Tidset* $=\{1,2,3,4,5,6\}$。然后依次求得项目集的 Tidset。例如项目集 AC 的 Tidset 由 A 和 C 的 Tidset 取交集得到,为 *AC. Tidset* = *A. Tidset* ∩ *C. Tidset* $=\{1,3,4,5\}$。从而计算得到 AC 的频繁度为 $|AC.\ Tidset|=4$,支持度为 $|AC.\ Tidset|/|D|=0.67$,与用户给出的最小支持度进行比较,就可判定该项目集是否频繁。依此类推,就可以计算得到项目集的支持度。

表 1　示例事务集

Transation ID	项目集
1	{A,C,T,W}
2	{C,E,W}
3	{A,C,T,W}
4	{A,C,E,W}
5	{A,C,E,T,W}
6	{C,E,T}



(a) 水平格式　　(b) 垂直格式

图 1　数据格式

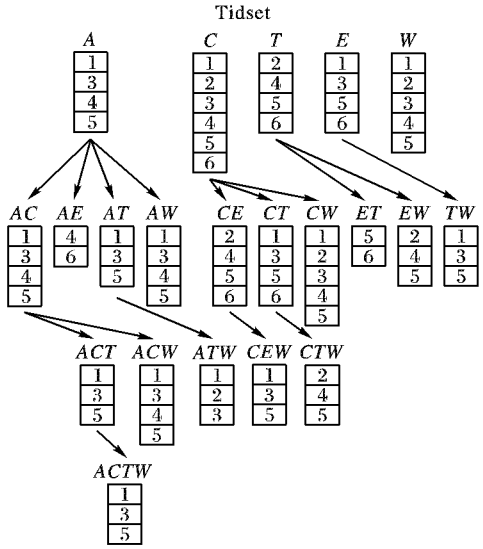


图 2　采用 Tidset 计算项目集的频繁度

而给定一个由项目 A 和 C 组合形成的项目集 AC,定义 AC 的 Diffset 为:

AC. Diffset = *A. Tidset* − *C. Tidset*

式中,"−"表示两个集合的求差集的计算。上式中 AC 的 *Diffset* 的含义为包含 A 的事务标识集合和包含 C 的事务标识集合的差集。*AC. Diffset* $=\{1,3,4,5\}-\{1,2,3,4,5,6\}=\{\}$。此差集中的标识包含于 *A. Tidset*,而不包含于 *C. Tidset*。

对于一个项目,定义它的 Diffset 为事务集标识集合 D 与该项目的差集。例如,项目 A 与项目 C 的 Diffset 分别如下:

项目 A 的 Diffset 为:*A. Diffset* = *D* − *A. Tidset* $=\{1,2,3,4,5,6\}-\{1,3,4,5\}=\{2,6\}$

项目 C 关联的 Diffset 为:*C. Diffset* = *D* − *C. Tidset* $=\{1,2,3,4,5,6\}-\{1,2,3,4,5,6\}=\{\}$,其中 D 为事务集所有的标识集合。

根据项目 A 和项目 C 的 Diffset,可以计算得到 AC 的 Diffset:

$AC.Diffset = A.Tidset - C.Tidset = (D - A.Diffset) - (D - C.Diffset) = C.Diffset - A.Diffset = \{\}$

上式中的{} 表示空集。

AC 的频繁度为：$Freq(AC) = |A.Tidset \cap C.Tidset| = Freq(A) - |AC.Diffset| = 4$。

对应 AC 的支持度为：$Supp(AC) = Freq(AC) / |D| = 4/6 = 0.67$。

图3为采用Diffset计算项目集的频繁度示意图。对比图2可以看出，Diffset记录的事务标识的数目远少于基于Tidset的方法，而且Diffset的运算过程也相对简单，从而提高了挖掘的效率，减少了内存开销。

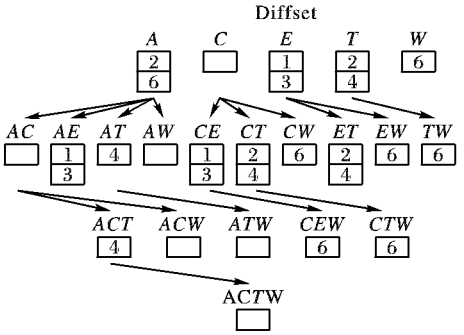


图3 采用Diffset计算项目集的频繁度

3 基于Diffset的DiffFRSTMiner挖掘算法

算法中freq(RST)表示RST的频繁度，F_i 表示挖掘得到的 i 边频繁同根子树集合，RST^k 表示 k 边同根子树，第11行的“-”表示集合的求差集计算。

算法 DiffFRSTMiner (D, $minSupp$)

输入：D 为查询模式事务集，$D = \{QPT_1, QPT_2, \cdots, QPT_n\}$；minSupp为最小支持度。

输出：频繁同根子树集合。

```
1)  F1 = {frequent 1-edge RST^1}
                                  //生成1边频繁同根子树集合
2)  for each RST^1 ∈ F1, get RST^1.Diffset
                                  //得到1边频繁同根子树的Diffset
3)    k = 1;
4)    while Fk ≠ ∅ do begin
5)      for each RST^k ∈ Fk
6)        for each RST^(k+1) obtained by child extension RST^k
                                  //孩子扩展方式生成RST^(k+1)
7)          get RST^(k+1).Diffset by RST^k.Diffset
                                  //计算得到RST^(k+1)的Diffset
8)          if (freq(RST^k) - |RST^(k+1).Diffset|) ≥
              minSupp × |D|);     //判断是否频繁
9)            F(k+1) = F(k+1) ∪ RST^(k+1);  //RST^(k+1)加入F(k+1)
10)         for each RST^(k+1) obtained by join RST^k with RST^k' ∈
              Fk(RST ≠ RST')
                                  //联结方式生成同根子树
11)           RST^(k+1).Diffset = RST^k'.Diffset - RST^k.Diffset
12)           if (freq(RST^k) - |RST^(k+1).Diffset|) ≥
                Minsupp × |D|);   //判断是否频繁
13)             F(k+1) = F(k+1) ∪ RST^(k+1);  //RST^(k+1)加入F(k+1)
14)         k = k + 1;
15)   end                         //end of while
16)   Return Fk(k = 1, ..., n)
```

在上面的算法伪码程序行中，第1行生成1边频繁同根子树集合。第3至第14行为由1边频繁同根子树集合开始，通过孩子扩展生成所有的频繁同根子树。第5至8行表示通过孩子扩展方式生成由 k 边同根子树生成 $k+1$ 边同根子树 RST^{k+1}，并计算其支持度。支持度计算采用基于 *Diffset* 的方法。若为频繁同根子树，则加入相应的 $k+1$ 边频繁同根子树集合中。第9至13行为由两个 k 边同根子树联结生成 $k+1$ 边同根子树 RST^{k+1}。*Diffset* 由这两个 k 边同根子树的 *Diffset* 取差集得到。判断生成的同根子树是否频繁，若频繁，则加入相应的 $k+1$ 边频繁同根子树集合中。按照上述步骤，挖掘得到所有的频繁同根子树。

4 算法复杂性分析和实验验证

4.1 算法复杂性分析

在挖掘XML频繁查询模式的算法中，主要计算步骤是遍历查询事务数据集的所有QPT、枚举子树以及计算频繁度。

算法的第一步需要遍历整个查询事务数据集，生成所有的0边RST，即枚举节点。在此过程中得到所有的不同标记，并统计查询树的数目、不同标记的数据、树的最大深入等信息。对于一个查询模式树RST，对其进行遍历的时间为 $O(n)$，n 为该查询模式树的大小 $n = |RST|$。

计算 F_2 时，每个边的支持度需要用一个二维数组记录，同时生成各个边的 *SList*。这个步骤所需要的时间为每个查询树 $O(n^2)$。

计算 $F_k (k \geq 3)$ 时，每个 F_k 中的同根子树由一个等价类成员扩展生成，需要进行等价类成员与元素的联结操作。在计算频繁度时，还需要进行 *SList* 的联结操作（孩子扩展）或两个同根子树的 *Tidset* 求交集的操作（同胞扩展）。如果[P]有 n 个元素，总的代价为 $O(qn^2)$，其中，q 为 *SList* 的联结成本或两个同根子树的 *Tidset* 求交集的代价。联结操作的代价为 $O(me^2)$，其中，m 为两个 *SList* 中不同的查询事务表示的数目，e 为每个 *SList* 中不同的查询事务中同根子树发生的次数，总的代价为 $O(m(en)^2)$。对于同胞扩展，由于只进行 *Tidset* 的比较操作，因此，q 为1，总代价为 $O(qn^2)$。

4.2 算法实验验证与性能分析

为验证和评估该算法，本文做了在一定数据集和参数范围内的实验。实验在一台RAM为256 MB的1.8 GHz PC上进行，操作系统为Windows 2000，算法用C++实现。

由于本文算法采用的查询包含检测是基于树同构的概念，不同于现有算法中采用子树嵌入的查询包含检测方法，因此，不能将本文提出的算法与现有算法直接进行比较。本文对基于Diffset方法的DiffFRSTMiner算法和基于Tidset方法的FRSTMiner算法的性能进行了实验研究，并对二者性能进行了比较。

实验采用DBLP提供的标准DTD-DBLP.dtd作为XML数据源的模式，生成DTD树。然后由DTD的同根子树，采用Zipf概率分布，从同根子树中生成查询事务集 D。采用Zipf分布，主要是Zipf分布符合Web查询分布特征。查询事务集 D 中QPT的数据集特征如下：平均节点数为7.5，最大深度为7，最大扇出为11。

为了对支持度变化时的算法的性能进行分析，实验使用200 KB大小的数据集分别进行计算。该数据集包含200 000 QPT，取支持度分别为0.1%、0.5%、1.0%、1.5%、2.0%。算法的性能如图4所示。从图中可以看出，算法

DiffFRSTMiner 的性能要好于算法 FRSTMiner。

本文也对数据集大小和算法计算量之间的关系进行了分析。设定支持度为(2%)时,取大小为 50 KB、100 KB、200 KB、300 KB 至 500 KB 的数据集(由于 200 KB 大小的数据集就包含 200 000 QPT,因此,这些数据集具有足够数量的 QPT),分别进行了 FRSTMiner 和 DiffFRSTMiner 两种算法的挖掘性能分析,结果如图 5 所示。从图中可以看出,算法的计算量的增长与最小支持度和数据集的大小基本呈线性关系,算法 DiffFRSTMiner 的性能依然优于算法 FRSTMiner。

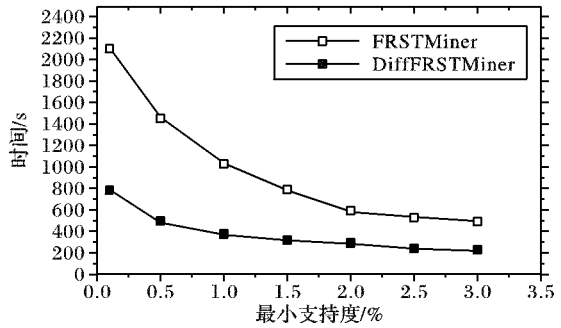


图 4　Zipf 分布下不同最小支持度下算法挖掘性能

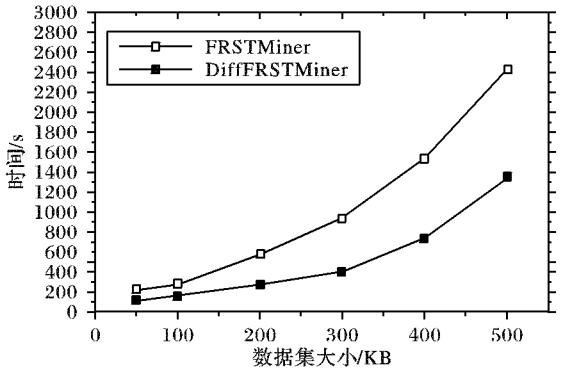


图 5　Zipf 分布下不同 QPTs 数据集大小时算法挖掘性能

5　结语

基于树包含概念的查询包含检测方法进行频繁查询模式挖掘,不同于现有的简单的图(树)匹配概念,符合 XML 查询语义。实验表明 DiffFRSTMiner 算法能够有效挖掘同根子树,而引入 Diffset 的挖掘算法由于减少了记录的查询事务数目,简化了挖掘得到的同根子树的支持度计算过程,具有更好的性能。

参考文献:

[1] BRAY T, PAOLI J, SPERBERG-McQUEEN C M, *et al.* Extensible Markup Language (XML) 1.0 (Fourth Edition) [EB/OL]. [2006-08-16]. http://www.w3.org/TR/xml.

[2] AGRAWAL R, IMIELINSKI T, SWAMI A. Mining association rules between sets of items in large databases [C]// Proceedings of the ACM SIGMOD Conference on Management of Data. Washington: ACM Press, 1993: 207-216.

[3] HAN J, KAMBER M. Data mining: Concepts and techniques[M]. San Francisco: Morgan Kaufmann Publishers, 2001.

[4] HAN JIA-WEI, PEI JIAN, YIN YI-WEN. Mining frequent patterns without candidate generation [C]// Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data. New York: ACM Press, 2000, 29(2): 1-12.

[5] PEI JIAN, HAN JIA-WEI, LU HONG-JUN, *et al.* H-Mine: Hyper-structure mining of frequent patterns in large databases [C]// Proceedings of 2001 International Conference Data Mining (ICDM'01). San Jose: [s.n.], 2001:441-448.

[6] WANG KE, HE YU, HAN JIA-WEI. Mining frequent itemsets using support constraints [C]// Proceedings of 26th International Conference on VLDB. San Francisco: Morgan Kaufmann Publishers, 2000: 43-52.

[7] ZAKI M J. Fast vertical mining using diffsets, TR01-1[R]. New York: Rensselaer Polytechnic Institute, 2001.

(上接第 1449 页)

4　结语

本文对定性概率网进行了扩展,并基于粗糙集属性间依赖度求解理论,给出一种定性概率网推理冲突的解决方法。该方法只需要构建一个网络结构,并且能够较大程度上解决推理过程中产生的冲突结果。接下来将进一步研究节点间权重值的性质,以试图引入区间权重值等性质,以便能解决更多实际应用问题。

参考文献:

[1] PEARL J. Probabilistic reasoning in intelligent systems: networks of plausible inference [M]. San Francisco: Morgan Kaufmann Publishers, 1988: 117-133.

[2] WELLMAN M P. Fundamental concepts of qualitative probabilistic networks [J]. Artificial Intelligence, 1990, 44(3): 257-303.

[3] PARSONS S. Qualitative methods for reasoning under uncertainty [M]. Cambridge, MA: MIT Press, 2001.

[4] PARSONS S. Refining reasoning in qualitative probabilistic networks [C]// Proceedings of the Eleventh conference on Uncertainty in Artificial Intelligence. San Francisco: Morgan Kaufmann Publishers, 1995: 427-437.

[5] DRUZDZEL M J, HENRION M. Efficient reasoning in qualitative probabilistic networks [C]// Proceedings of the Eleventh National Conference on Artificial Intelligence. San Francisco: Morgan Kaufmann Publishers, 1993: 548-553.

[6] LIU C L, WWLLMAN M P. Incremental tradeoff resolution in qualitative probabilistic networks [C]// Proceedings of the Fourteenth Conference on Uncertainty in Artificial Intelligence. San Francisco: Morgan Kaufmann Publishers, 1998: 338-345.

[7] RENOOIJ S, Van Der GAAG L C. Enhancing QPNs for trade-off resolution [C]// Proceedings of the Fifteenth Conference on Uncertainty in Artificial Intelligence. San Francisco: Morgan Kaufmann Publishers, 1999: 559-566.

[8] LAK Z P. Rough set theory and its applications to data analysis [J]. Cybernetics and System, 1998, 29(7): 661-688.

[9] LAK Z P. Rough sets [J]. Compute Information Systems, 1982, 11(5): 341-336.

[10] 刘树安,杜红涛,王晓玲.粗糙集理论与应用发展[J].系统工程理论与实践,2001,21(10):64-68.

[11] RENOOIJ S, Van Der GAAG L C, PARSONS S. Context-specific sign-propagation in qualitative probabilistic networks [J]. Artificial Intelligence, 2002, 144(1): 207-230.

[12] CHENG J, BELL D, LIU W. Learning Bayesian network from data: an efficient approach based on information theory [J]. Artificial Intelligence, 1997, 137(1/2): 43-90.