一种新的工作流频繁模式挖掘算法研究

高 昂[1] 杨 扬[1] 王玥薇[1,2]
(北京科技大学信息工程学院 北京 100083)[1] (公安海警高等专科学校 宁波 315801)[2]

摘 要 为了提高工作流模型挖掘技术的准确性,提出了一种新的工作流频繁模式挖掘算法。首先,阐述了工作流模型依赖矩阵的定义,并利用工作流日志建立了依赖矩阵。然后采用活动间的依赖关系作为频繁项集,设计了一种基于依赖矩阵的频繁项集自动生成算法。最后对频繁项集进行处理,得到最终的工作流频繁模式。该算法能够处理活动间交叠关系和具有串、并行关系的工作流模型,因此更具优越性。

关键词 工作流模型,矩阵,频繁模式挖掘
中图法分类号 TP391 **文献标识码** A

New Algorithm Research for Mining Workflow Frequent Pattern

GAO Ang[1] YANG Yang[1] WANG Yue-wei[1,2]
(College of Information Engineering, University of Science and Technology Beijing, Beijing 100083, China)[1]
(Public Security Marine Police Academy, Ningbo 315801, China)[2]

Abstract To improve mining accuracy of workflow models, a new algorithm for mining workflow frequent pattern was proposed. Firstly, the Workflow Model depend Matrix (WM) was defined, and set up WM by using workflow logs. Secondly, using the depend relation of activities as frequent itemsets, an alogrithm was designed to automatically generate frequent itemsets based on WM. Finally, got the workflow frequent pattern by disposing frequent itemsets. The algorithm has advantage in disposing the interleaving relations between activities and workflow models with the serial or parallel relations.

Keywords Workflow model, Matrix, Frequent pattern mining

1 引言

传统的工作流建模需要投入大量的时间,一般由商业顾问和管理者完成,他们对模型的理解往往会影响模型的质量[1]。与之相比,工作流管理系统日志中包含丰富的信息(包括商业流程中各个活动的执行过程),可以利用这些更为客观的信息建立工作流模型。从工作流的生命周期(模型设计阶段、模型实现阶段、模型运行和模型维护阶段)来看,传统的建模方法主要集中于前两个阶段,而工作流挖掘技术主要针对运行与维护阶段,并将这两个阶段产生的信息反馈于设计和实现阶段,为模型再设计提供信息。工作流挖掘技术既可作为商业智能的一部分,也可为商业流程分析提供支持。

本文通过扩展经典的 Apriori 算法[2]作为新的工作流模型频繁模式挖掘方法,挖掘模型中的频繁模式。工作流频繁模式是体现活动间逻辑关系强弱的重要工具,利用工作流模型频繁模式能够在流程发生变化时及时地发现现有的模型变化及趋势,并量化活动间逻辑关系的强弱。在商务智能领域,利用频繁模式可以实时监控工作流管理系统运行,在关键的决策点可以提供后续活动走势预测、商业决策和风险预测等方面支持。与其他工作流模式挖掘方法相比,本算法以活动间依赖关系为频繁模式项集,解决了其他算法不能处理的活动间交叠关系,能够处理具有串、并行关系的工作流模型,更具优越性。

2 基本概念

活动是工作流模型的基本组成单元,$V=(a_1,a_2,\cdots,a_n)$ 表示日志中所有活动的集合。实例是四元组($InsNO,a_i,ST,ET$)的集合,其中 $InsNO$ 为实例编号,$a_i\in V$,ST 和 ET 分别为活动 a_i 的开始事件发生时间和结束事件发生时间。S_0 表示日志中所有实例的集合。实例 I 的活动集为 $AS(I)=\{a_i:(InsNO,a_i,ST,ET)\in I\}$。假设工作流模型中不存在循环,则任何实例 I 都不存在重复活动。

定义 1 实例集合 S 上,活动 a_j 局部依赖于活动 a_i,记为 $a_i\pi a_j\Leftrightarrow\forall$ 实例 $I\in S$,如果有 $a_i\in AS(I)$,$a_j\in AS(I)$且活动 a_i 的结束时间 ET_i 早于活动 a_j 的开始时间 ST_j 同时成立。

两个活动如果仅在一部分实例集上依赖并不能保证依赖关系的正确性,只有当它们在整个日志实例集上依赖时才能保证这个依赖关系是正确的。

定义 2 活动 a_j 全局依赖于活动 a_i,记为 $a_i\pi a_j\Leftrightarrow$对于实

到稿日期:2008-10-14 返修日期:2008-12-25 本文受国家自然科学基金(60673160)资助。
高 昂(1981—),男,博士研究生,主要研究方向为工作流挖掘、电子商务、Web 服务等,E-mail:gaoang318@163.com;杨 扬(1955—),男,教授,博士生导师,主要研究方向为电子商务、图像处理、网格计算等;王玥薇(1980—),女,硕士研究生,讲师,主要研究方向为电子商务、网格计算等。

例集合 S，有 $a_i\pi_s a_j$（其中 $S=\{I:a_i\in AS(I),a_j\in AS(I),I\in S_0\}$）。

由定义 1 易知，依赖具有传递性，即实例集合 S 中如果有 $a_i\pi_s a_k$ 且 $a_k\pi_s a_j\Rightarrow a_i\pi_s a_j$。从依赖的传递性可以得知，在建立的工作流模型中并不需要表示所有活动间的依赖关系，因为一部分依赖关系可以通过其他活动间的依赖关系传递表达。由此，在全局依赖的基础上提出了直接依赖的定义。

定义 3 活动 a_j 直接依赖于活动 a_i，$a_i\pi^d a_j\Leftrightarrow a_i\pi a_j$，且对于 $\forall I\in S,\neg\exists a_k$，使 $a_i\pi a_k$ 和 $a_k\pi a_j$ 同时成立（其中令 $S'=\{I:a_i\in AS(I),a_j\in AS(I),I\in S_0\}$，$S$ 为 S' 中所有具有相同的实例活动集的实例的集合）。

从直接依赖的定义可知，直接依赖关系首先是全局依赖关系，这样保证了依赖关系的正确性。其次考察定义 3 中实例集 S 的范围。由于具有不同活动集的实例间可能蕴涵着逻辑或关系，S 的范围为日志 S_0 中所有包含活动 a_i 和 a_j 且具有相同实例活动集的实例集合。这个定义将 S_0 中所有包含 a_i 和 a_j 的活动分为多个子集。后续的算法将利用这一点分别挖掘每个子集中 a_i 和 a_j 间的直接依赖关系并汇总，得到活动间的逻辑或关系。显然，当活动 $a_i\pi^d a_j$ 时，在扩展有向图中应该存在一条有向边，由活动 a_i 指向 a_j。

定义 4 模式 $G=(V,F)$ 为某实例 I_j 的子集，称实例 I 支持一个有向图表示的工作流模式 P。如果模式 G 中的依赖关系在实例 I 中是成立的，记为 $G\Uparrow I$。

定义 5 日志中所有实例集合 S_0，对于工作流模式 G 的支持度 $P=|\{I:I\in S_0,G\Uparrow I\}|/|S_0|\times 100\%$，其中 $|S_0|$ 表示日志中的实例数。同时可得，对于实例集 S'，有向图所表示的工作流模式 G' 在实例集 S' 上是频繁的，如果实例集 S' 中有对模式 G' 的支持度 $s\geqslant$ 最小支持度阈值 P_s。

定义 6 $\forall I_i,I_j\in$ 实例集 S，满足 $AS(I_i)=AS(I_j)$，则实例集 S 对应的依赖矩阵定义为 v 阶布尔矩阵 $D=(d_{ij})$，$v=|DS(I)|$，$|DS(I)|$ 为集合 $DS(I)$ 的元素个数。其中 if 活动 $a_i\pi_s a_j$，$d_{ij}=1$；else if $a_j\pi_s a_i$，$d_{ij}=-1$；else $d_{ij}=0$。

3 频繁模式挖掘算法基本思想

Agrawal 等人最早提出了串行模式挖掘方法，该方法事先定义阈值，利用 Apriori 算法挖掘串行实例中不小于阈值的极大串行序列作为串行模式，并且对算法进行了优化[3,4]。Mannila 等人利用窗口方法的最终目的是找到序列的集合，集合中的每个序列都被足够多的窗口所包含到[5]。但是文献[6,7]中的挖掘方法只将每个活动作为一个原子事件，没有考虑每个活动从开始事件到结束事件之间的时间间隔，所以只能处理串行的工作流模式序列。

本文主要通过扩展经典的 Apriori 算法挖掘工作流频繁模式。Apriori 算法中，频繁项集仅是一个事务中项的集合，没有顺序关系。但在工作流频繁模式挖掘算法中，活动间有先后顺序，而且存在并行结构，所以采用活动间依赖关系作为频繁项集和候选项集的项，即每个实例表示为形如 $d_{ij}=n(n=1,0,-1)$ 的依赖矩阵元素集合，通过函数 *CompMatrix-Patterns* 计算频繁项集，再通过函数 *CompPatterns* 得到最终的工作流频繁模式集合。此外，Apriori 算法中对频繁项没有要求。但在工作流模型中，依赖矩阵 D_i 包含活动集合 $AS(D_i)$ 中任意活动间的依赖关系，其表示的模型才具有意义，所以在工作流模式挖掘中，要求项集频繁且完整。

4 频繁模式挖掘过程

本文提出的工作流频繁模型挖掘过程可以分为 4 个步骤。下面以某公司处理客户投诉的工作流模型为例，详细解释模型挖掘过程。图 1 是以有向图表示的某公司处理客户投诉的过程模型。其中，活动 a_1 为投诉登记；活动 a_2 为邮寄调查表；活动 a_3 为投诉评估，在投诉评估后，可以选择处理投诉或跳过处理；活动 a_4 为处理调查表；活动 a_5 为处理投诉；活动 a_6 为检查结果；活动 a_7 为归档。

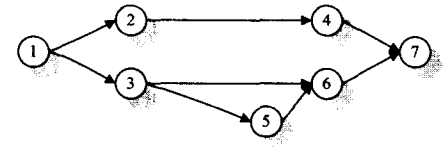

图 1 某公司处理客户投诉过程模型

当公司接到投诉并进行登记后，处理调查表和评估投诉两个活动在逻辑上具有并行关系，在评估完成之后，可以选择处理投诉或跳过处理。

4.1 数据预处理

对日志中每一个实例 I，算法关心的是活动集 $AS(I)$ 中的每个活动开始事件和结束事件发生的先后顺序，而并非开始事件和结束事件的具体发生时间，所以首先将日志数据进行初步整理，得到每个实例的活动的开始事件和结束事件的时间序列。如果 $AS(I_i)=AS(I_j)$，且实例 I_i 和 I_j 中各个活动的开始事件和结束事件有相同的时间序列，则称实例 I_i 和 I_j 是重复的。工作流日志中存在大量的重复实例，在数据预处理中过滤掉重复的实例，可以大大地减少后续步骤中处理的数据量。

对日志 S_0，在去掉重复实例后得到工作流实例集 S，将其分为多个子集 $S_1,S_2,\cdots,S_n$，满足 $\forall S_i$ 中如果 $I_i\in S_i$，则 $\forall I_j\in S_i$，都有 $AS(I_i)=AS(I_j)$，且 $S=S_1\cup S_2\cup S_3\cup\cdots\cup S_n$。

对于图 1 所示的工作流模型，将日志经过上述预处理后得到实例集 S 如下，其中 a_i^s 和 a_i^e 分别表示活动开始事件和活动结束事件。

$I_1=(a_1^s,a_1^e,a_2^s,a_3^s,a_2^e,a_4^s,a_3^e,a_5^s,a_4^e,a_5^e,a_6^s,a_6^e,a_7^s,a_7^e)$，

$I_2=(a_1^s,a_1^e,a_2^s,a_3^s,a_3^e,a_5^s,a_2^e,a_4^s,a_5^e,a_6^s,a_4^e,a_6^e,a_7^s,a_7^e)$，

$I_3=(a_1^s,a_1^e,a_3^s,a_2^s,a_5^s,a_6^s,a_2^e,a_4^s,a_4^e,a_6^e,a_7^s,a_7^e)$，

$I_4=(a_1^s,a_1^e,a_2^s,a_3^s,a_3^e,a_6^s,a_6^e,a_2^e,a_4^s,a_4^e,a_7^s,a_7^e)$，

$I_5=(a_1^s,a_1^e,a_2^s,a_3^s,a_3^e,a_5^s,a_5^e,a_6^s,a_2^e,a_4^s,a_6^e,a_4^e,a_7^s,a_7^e)$。

$S_1=\{I_1,I_2,I_5\}$，$S_2=\{I_3,I_4\}$。

4.2 计算依赖矩阵

经过数据预处理之后得到实例集 $S_1,S_2,\cdots,S_n$，对每一个实例集 S_i 按照依赖矩阵定义，分别得到对应的依赖矩阵。上述处理客户投诉过程模型中的实例集 S_1,S_2 按照定义 6 可得依赖矩阵 D_1 和 D_2。算法中采用活动间的依赖关系作为实例项，则可得 1 项候选集合 $C_1=\{d_{ij}=1\cup d_{ij}=0\cup d_{ij}=-1,a_i,a_j\in V,$且 $i<j\}$。按照定义 5，对 1 项候选集合 C_1 中的每个项计算 S_0 上的支持度，将支持度大于阈值 P_s 的项记入 1 项频繁集合 L_1。

$$D_1=\begin{array}{c}\\ a_1\\ a_2\\ a_3\\ a_4\\ a_5\\ a_6\\ a_7\end{array}\begin{array}{c}\begin{array}{ccccccc}a_1 & a_2 & a_3 & a_4 & a_5 & a_6 & a_7\end{array}\\ \begin{bmatrix}0&1&1&1&1&1&1\\-1&0&0&1&0&0&1\\-1&0&0&0&1&1&1\\-1&-1&0&0&0&0&1\\-1&0&-1&0&0&1&1\\-1&0&-1&0&-1&0&1\\-1&-1&-1&-1&-1&-1&0\end{bmatrix}\end{array}$$

$$D_2=\begin{array}{c}\\ a_1\\ a_2\\ a_3\\ a_4\\ a_6\\ a_7\end{array}\begin{array}{c}\begin{array}{ccccccc}a_1 & a_2 & a_3 & a_4 & a_5 & a_6 & a_7\end{array}\\ \begin{bmatrix}0&1&1&1&1&1\\-1&0&0&1&0&1\\-1&0&0&1&1&1\\-1&-1&-1&0&0&1\\-1&0&-1&0&0&1\\-1&-1&-1&-1&-1&0\end{bmatrix}\end{array}$$

依赖矩阵都是沿对角线对称，所以可以将依赖矩阵简化为上三角矩阵，在后续的步骤中降低算法的复杂度。

$$D_1=\begin{array}{c}\\ a_1\\ a_2\\ a_3\\ a_4\\ a_5\\ a_6\\ a_7\end{array}\begin{array}{c}\begin{array}{ccccccc}a_1 & a_2 & a_3 & a_4 & a_5 & a_6 & a_7\end{array}\\ \begin{bmatrix}1&1&1&1&1&1\\&0&1&0&0&1\\&&0&1&1&1\\&&&0&0&1\\&&&&1&1\\&&&&&1\\&&&&&\end{bmatrix}\end{array}$$

$$D_2=\begin{array}{c}\\ a_1\\ a_2\\ a_3\\ a_4\\ a_6\\ a_7\end{array}\begin{array}{c}\begin{array}{ccccccc}a_1 & a_2 & a_3 & a_4 & a_5 & a_6 & a_7\end{array}\\ \begin{bmatrix}1&1&1&1&1\\&0&1&0&1\\&&1&1&1\\&&&0&1\\&&&&1\\&&&&\end{bmatrix}\end{array}$$

4.3 计算矩阵频繁项集

利用 *CompMatrixPatterns* 函数计算矩阵频繁项集。C_k 代表 k 项候选频繁集；L_k 代表 k 代项频繁集。首先计算所有频繁项集，但是这些频繁项集并不一定完整。然后去掉其中不完全的矩阵频繁项集，得到最终的频繁集。

CompMatrixPatterns(全体实例集合 S_0，1 项频繁集合 L_1，阈值 P_s)

{

Max K=1；

for(k=2；$L_k\neq\emptyset$；k++)

{

由 $k-1$ 项频繁集 L_{k-1} 生成 k 项候选集 C_k；

for(矩阵 D_1，D_2，…，D_n)

{

for(k 项候选集 C_k 中每项 C_n)

{if(矩阵 D_i 中存在 k 个元素值=C_n 中元素)//找出矩阵 D_i 支持的项 c；

$C_1=C_1\cup C_n$；$C_n.count$++；

}

$L_k=\{d_{ij}\in C_1\mid C_n.count/|S_0|\times100\%>P_s\}$；

If($L_k\neq\emptyset$) Max K++；

}

for(k= Max K；k>1；k--)

{

for(L_k 中每一项 C_n)

{

if(d_{ij}($d_{ij}\in C_n$)组成的矩阵为上三角完全矩阵)

delete L_{k-1}，L_{k-2}，…，L_i 中 C_n 的子项；

else delete C_n；

}

}

Retrun L_1，L_2，…，$L_{Max\,k}$；

}

上述依赖矩阵经函数 *CompMatrixPatterns* 处理后得到矩阵频繁项集 $L_2=\{(D_1',D_2')\}$，其中

$$D_1'=\begin{array}{c}\\ a_1\\ a_2\\ a_4\\ a_7\end{array}\begin{array}{c}\begin{array}{ccc}a_3 & a_5 & a_6\end{array}\\ \begin{bmatrix}1&1&1\\&1&1\\&&1\\&&\end{bmatrix}\end{array},\quad D_2'=\begin{array}{c}\\ a_1\\ a_3\\ a_5\\ a_6\end{array}\begin{array}{c}\begin{array}{ccc}a_2 & a_4 & a_7\end{array}\\ \begin{bmatrix}1&1&1\\&1&1\\&&0\\&&\end{bmatrix}\end{array}$$

4.4 生成频繁模式

利用 *ComPatterns* 函数将生成的频繁矩阵元素项集 L_1，L_2，…，$L_{Max\,k}$ 中的项生成以有向图表示的频繁模式。对于频繁矩阵元素项集 L_k，其中每个项 C_n 中的矩阵元素集合$\{d_{ij}:d_{ij}\in C_n\}$可以组成一个上三角矩阵 D_n，即对应活动集合 $AS(C_n)$的依赖矩阵。*ComPatterns* 把活动集$AS(S_i)$分为 4 个集合：*FormerSet*，*CurSet*，*NextSet*，*LeftSet*。

CompPatterns(*Depend Matrix*：D_n)

{

FormerSet=*NextSet*=*temp*=$\emptyset$，*CurSet* =（活动 a_j：$a_j\in AS(S_i)$且a_j 不依赖于其他任何活动），*LeftSet* = $AS(S_i)$ − *CurSet*；

while *LeftSet*$\neq\emptyset${

for (*CurSet* 中的每一个活动 a_j){

CurrentAct = a_j，*NextSet* =（活动 a_u：$a_u\in$ *LeftSet*，且 a_u 不依赖于其他任何 *LeftSet* 中的活动)；

for(*NextSet* 中的每一个活动 a_u){

for(*LeftSet* 中的每一个活动 $a_v\neq a_u$){

if(活动 a_v 依赖于活动 a_j)矩阵 D_i 中令元素 $d_{jv}=0$；

}

}

temp=*temp*∪*NextSet*；*NextSet*=$\emptyset$；

}

FormerSet=*FormerSet*∪*CurSet*；*CurSet*=*temp*；

LeftSet=*LeftSet*−*CurSet*；

}

}

由函数 *ComPatterns* 处理频繁依赖矩阵 D_1'，D_2'可得包含直接依赖关系的矩阵 D_3'，D_4'，如下所示：

$$D_3'=\begin{array}{c}\\ a_1\\ a_2\\ a_4\\ a_7\end{array}\begin{array}{c}\begin{array}{ccc}a_3 & a_5 & a_6\end{array}\\ \begin{bmatrix}1&&\\&1&\\&&1\\&&\end{bmatrix}\end{array},\quad D_4'=\begin{array}{c}\\ a_1\\ a_3\\ a_5\\ a_6\end{array}\begin{array}{c}\begin{array}{ccc}a_2 & a_4 & a_7\end{array}\\ \begin{bmatrix}1&&\\&1&1\\&&\\&&\end{bmatrix}\end{array}$$

（下转第 251 页）

图 5 给出了并行复算加速比在故障率不同的情况下对容错并行算法性能的影响。容错并行算法的期望执行时间随着并行复算加速比的增大而减小，加速比和效率随之而增大。故障率较高时，增大并行复算的加速比可以有效提高容错并行算法的性能。然而，故障率较低时，并行复算的加速比对容错并行算法性能的影响也较小。

结束语 传统的并行算法性能度量用于评估无故障时并行程序的性能。本文建立了考虑系统故障情况下的性能模型来预测容错并行算法的完成时间，以此为基础评估了程序段的运行时间、数据保存开销、故障率以及并行复算加速比等系统参数对容错并行算法性能的影响。程序段的选择是个优化设计问题，存在最优程序段运行时间；数据保存开销对容错并行算法的影响较大，它比并行复算加速比对容错并行算法性能的影响更大，数据保存开销越低，容错并行算法的性能越好；故障率越低，系统发生故障的概率越低，容错并行算法的性能越好；并行复算加速比的增大可以提高容错并行算法的性能，尤其是对于故障率较高以及程序段运行时间较长的容错并行算法，增大并行复算加速比可以有效提高容错并行算法的性能。

参 考 文 献

[1] Grama A，Gupta A，Karypis G，et al. Introduction to Parallel Computing，Second Edition[M]. Addison Wesley，January 2003

[2] Yang Xuejun，Du Yunfei，Wang Panfeng，et al. FTPA：Supporting Fault Tolerant Parallel Computing Through Parallel Recomputing[J]. IEEE Transactions on Parallel and Distributed Systems (Accepted)

[3] Duda A. The Effects of Checkpointing on Program Execution Time[J]. Information Processing Letters，1983，16：221-229

[4] Young J W. A First Order Approximation to the Optimal Checkpoint Interval[J]. Comm. ACM，1974，17(9)：530-531

[5] Kim JunSeong，Lilja D J. Characterization of Communication Patterns in Message-Passing Parallel Scientific Application Programs[C]// Proceedings of the Second International Workshop on Network-based Parallel Computing：Communication，Architecture，and Applications. January 1998：202-216

[6] Vetter J S. Communication Characteristics of Large-scale Scientific Applications for Contemporary Cluster Architectures[C]// International Parallel and Distributed Processing Symposium. 2002

[7] Zamani R，Afsahi A. Communication Characteristics of Message-Passing Scientific and Engineering Applications[C]// Proceedings of the 17th IASTED International Conference on Parallel and Distributed Computing and Systems (PDCS '2005). Phoenix，AZ，USA，November 2005：644-649

[8] Amdahl G M. Validity of the single-processor approach to achieving large scale computing capabilities[J]. AFIPS Conference Proceedings，1967，4(30)：483-485

[9] Gustafson J. Reevaluating Amdahl's Law[J]. Communication of ACM，1988，31(5)：532-533

(上接第 233 页)

经过 *ComPatterns* 函数处理的矩阵 D_n 包含了日志中所有的活动间直接依赖关系，可以直接通过 D_n 生成频繁模式，由有向图 $G=(V,F)$ 表示，其中 $V=AS(C_n)$ 代表有向图的节点集合 $(a_1,a_2,\cdots,a_n)$；F 是有向图的边集。对于矩阵 D_n 元素 $d_{ij}=1$，则工作流模型中存在一条从有向边 a_i 指向 a_j；反之，$d_{ij}=-1$ 存在一条从有向边 a_j 指向 a_i。

最后根据直接依赖矩阵 D_3'，D'_4 得到最终的工作流频繁模式，如图 2 所示。

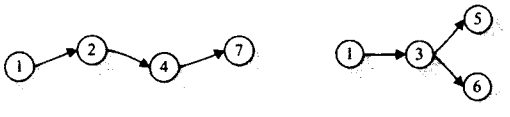

图 2 最终工作流频繁模式

结束语 工作流频繁模式包含了丰富的信息，可以为流程动态变化问题的解决提供基础。同时，可以为工作流管理系统中重要的商业决策、风险预测等提供支持，也为工作流模型优化提供依据。利用日志信息挖掘工作流频繁模式，是近年来新出现的研究领域。本文提出的算法与其他工作流模式挖掘方法相比，能够处理具有串、并行关系的工作流模式，更具优越性，在工作流模型中，活动间除串、并行关系外还存在循环关系。本算法在处理活动间并行循环关系时存在不足，不能完全处理循环关系，这是今后工作的主要方向。

参 考 文 献

[1] 范玉顺. 工作流管理技术基础[M]. 北京：清华大学出版社，2001：20-35

[2] Agrawal R，Imielinski T，Swami A. Mining association rules between sets of items in large databases [C]// Proceedings of the ACM SIGMOD International Conference Management of Date. Washington，1993：207-216

[3] Agrawal R，Srikant R. Mining sequential patterns[C]// Proceedings of International Conference on Data Engineering. Taipei，Taiwan，1995：135-139

[4] Srikant R. Agrawal R. Mining sequential patterns：generalizations and performance improvements[C]// Proceedings of the 5th International Conference on Extending Database Technology (EDBT). Avignon，France，1996：117-133

[5] Mannila H，Toivonen H，Verkamo A I. Discovering frequent episodes in sequences[C]// Proceedings of the First International Conference on Knowledge Discovery and Data Mining. Avignon，France，1995：194-204

[6] Sadiq W，Orlowska M E. Analyzing Process Models Using Graph Reduction Techniques[J]. Information Systems，2000，25(2)：117-134

[7] Sadiq S，Orlowska M. On correctness issues in conceptual modeling of workflows[C]// Proceedings of the 5th Euopean Conference on Information Systems. Cork，Ireland，1997：19-21