软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2015,26(3):533−549 [doi: 10.13328/j.cnki.jos.004769]

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel: +86-10-62562563

一种并行化的启发式流程挖掘算法*

鲁法明[1,2], 曾庆田[3,1], 段 华[4], 程久军[2], 包云霞[4]

[1](山东科技大学 信息科学与工程学院,山东 青岛 266590)

[2](嵌入式系统与服务计算教育部重点实验室(同济大学),上海 200092)

[3](山东科技大学 电子通信与物理学院,山东 青岛 266590)

[4](山东科技大学 数学与系统科学学院,山东 青岛 266590)

通讯作者: 曾庆田, E-mail: qtzeng@163.com, http://dzxy.sdust.edu.cn/contents/100/372.html

摘 要: 启发式流程挖掘算法在日志噪音与不完备日志的处理方面优势显著,但是现有算法对长距离依赖关系以及2-循环特殊结构的处理存在不足,而且算法未进行并行化处理.针对上述问题,基于执行任务集将流程模型划分为多个案例模型,结合改进的启发式算法并行挖掘各个案例模型所对应的 C-net 模型;再将上述模型集成得到完整流程对应的 C-net.同时,将长距离依赖关系扩展为决策点处两个任务子集之间的非局部依赖关系,给出了更为准确的长距离依赖关系度量指标和挖掘算法.上述改进措施使得该算法更为精确、高效.

关键词: 流程挖掘;启发式挖掘算法;长距离依赖关系;案例模型;案例簇

中图法分类号: TP301

中文引用格式: 鲁法明,曾庆田,段华,程久军,包云霞.一种并行化的启发式流程挖掘算法.软件学报,2015,26(3):533−549. http://www.jos.org.cn/1000-9825/4769.htm

英文引用格式: Lu FM, Zeng QT, Duan H, Cheng JJ, Bao YX. Parallelized heuristic process mining algorithm. Ruan Jian Xue Bao/Journal of Software, 2015,26(3):533−549 (in Chinese). http://www.jos.org.cn/1000-9825/4769.htm

Parallelized Heuristic Process Mining Algorithm

LU Fa-Ming[1,2], ZENG Qing-Tian[3,1], DUAN Hua[4], CHENG Jiu-Jun[2], BAO Yun-Xia[4]

[1](College of Information Science and Engineering, Shandong University of Science and Technology, Qingdao 266590, China)

[2](The Key Laboratory of Embedded System and Service Computing, Ministry of Education (Tongji University), Shanghai 200092, China)

[3](College of Electronic Communication and Physics, Shandong University of Science and Technology, Qingdao 266590, China)

[4](College of Mathematics and System Science, Shandong University of Science and Technology, Qingdao 266590, China)

Abstract: Heuristic process mining algorithm has a significant advantage in dealing with noise and incomplete logs. However, existing heuristic process mining algorithms cannot handle long-distance dependencies and lenth-2-loop structures correctly in some special situations. Besides, none of them are parallelized. To address the problems, process models are divided into multiple case models according to executed activity set at first. Then the C-nets corresponding to case models are discovered with an improved heuristic process mining algorithm in parallel. After that, these C-nets are integrated to derive the complete process model. Meanwhile, the definition of long- distance dependencies is extended to non-local dependencies between two activity sets in decision points. In addition, a more accurate long- distance dependency metrics and its corresponding mining algorithm are presented. These improvements make the proposed algorithm more accurate and efficient.

Key words: process mining; heuristic mining algorithm; long distance dependency; case model; case cluster

* 基金项目: 国家自然科学基金(61170079, 61202152, 61472229, 61472284); 山东省科技发展项目(2014GGX101035); 山东省优秀中青年科学家科研奖励基金(BS2014DX013); 青岛市科技计划基础研究项目(13-1-4-153-jch, 2013-1-24); 同济大学嵌入式系统与服务计算教育部重点实验室开放课题基金(ESSCKF201403); 山东科技大学群星计划(qx2013113, qx2013354)

收稿时间: 2014-06-30; 修改时间: 2014-09-30; 定稿时间: 2014-11-21

流程挖掘技术通过对日志数据的挖掘分析,实现实际业务流程的自动发现、符合性检查、社交网络/组织挖掘以及模型扩展、性能分析、案例预测等智能化处理.随着企业信息系统中事件日志的日益积累以及企业在竞争和飞速变化的环境中对更好地支持和改善业务流程的迫切需要,流程挖掘技术得到了学术界和企业界的广泛重视和研究、应用[1].流程发现通常是流程挖掘的起点,已有很多流程发现算法被提出[2].

对日志噪音和不完备日志的处理,是流程发现技术的难点[3].启发式流程挖掘系列算法[4,5]通过考虑任务的执行频度等信息,在前述两个问题的处理方面取得了一定突破.此外,对长距离依赖关系[1]的处理也是启发式流程挖掘的特色之一.然而,现有启发式流程挖掘算法认为:存在任务 a 到 b 的长距离依赖当且仅当 a 的发生几乎总是导致 b 发生,而且 a 与 b 在所有案例中的执行频度接近.实际上,长距离依赖关系中的后继任务 b 可能位于一个循环结构中,此时,b 的执行频度可能会远大于 a 的执行频度;此外,长距离依赖于 a 的有可能是互斥的两个任务 b 与 c,而此时,b 与 c 的执行频度之和与 a 的执行频度相当,单独的 b 或 c 的执行频度都将小于 a 的频度.这两类情况下的长距离依赖关系通过现有的启发式流程挖掘算法无法得到;与此同时,现有的启发式流程挖掘算法未进行并行化处理,而且在 2-循环等一些特殊结构的处理上存在不足.

针对上述问题,本文对已有的启发式流程挖掘算法进行改进:一方面,对长距离依赖关系给出更为准确的度量指标;另一方面,将整体流程模型根据执行任务集拆分为多个案例模型,先借助启发式挖掘算法挖掘各个案例模型对应的 C-net 模型[5];之后,将各个 C-net 模型融合以得到完整的流程模型.

于是,由于各案例模型的挖掘可并行进行,从而能有效提高流程发现的效率.此外,本文对流程模型中 2-循环结构进行了细分,并给出了相应的度量指标.

1 基本概念与运行实例

本节结合业务流程模型及其运行日志实例介绍流程挖掘相关的基本概念.

图 1 为业务流程的 WF-net[6]模型,其中:虚线框表示的变迁结点对应不可见任务[7],用以表达流程控制结构.该流程中存在两个决策点:一个对应任务 A 执行完毕后存在的两个互斥分支{B}与{C};另一个对应任务 D 执行完毕后的 3 个互斥分支{E},{G}以及由任务 H 与 I 构成的选择结构分支.需要指出:上述这两个决策点不是相互独立的,D 执行完毕后分支的选择依赖于 A 执行完毕后的分支选择情况.具体而言,当 D 执行完毕后:欲选择执行{E},则之前必须在 A 执行完后选择执行{B};欲选择执行{H}或{I},则必须在 A 执行完后选择执行{C}.上述两决策点分支选择间的依赖关系,通常称为长距离依赖关系[4].

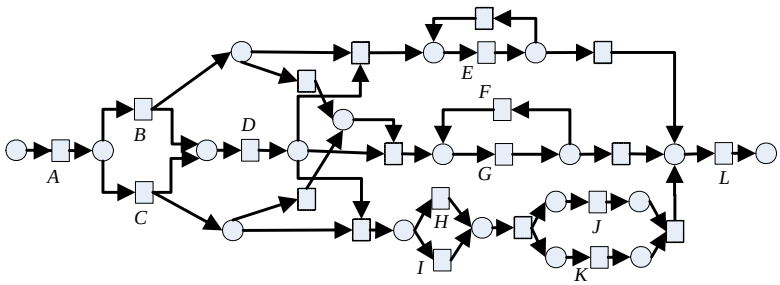


Fig.1 A process model example

图 1 流程模型实例

当案例在流程中运转时,被选择执行的可见任务形成的序列称为案例的运行轨迹.所有案例对应的运行轨迹构成一个袋集,称其为流程的一个简单事件日志[1].例如,表 1 给出了部分仿真案例在图 1 流程中的运行轨迹,它对应的简单事件日志为 $L=[\sigma_1^{10},\sigma_2^{20},\sigma_3^{10},\sigma_4^{10},\sigma_5^{10},\sigma_6^{10},\sigma_7^{10},\sigma_8^{10},\sigma_9^{10},\sigma_{10}^{10}]$,其中,$\sigma_i(i=1,2,\ldots,10)$为运行轨迹 ID,轨迹右上角的数字表示轨迹的出现频度.执行任务集相同的案例在流程模型中的路由结构是相同的,对应完整业务流程的一个子模型,称其为当前任务集对应的案例模型.例如,由表 1 中的案例运行轨迹可得图 1 流程的 7 个案

例模型,具体见表 2.设某案例模型对应的任务集合为 T,称事件日志中所有以 T 为执行任务集的案例运行轨迹构成的袋为当前案例模型的事件日志,记作 $L|_T$.例如,对于任务集 $T_1=\{A,B,D,E,L\}$ 对应的案例模型,其事件日志为 $L|_{T_1}=[\sigma_1^{10},\sigma_2^{20}]$,其他案例模型对应的日志见表 2.

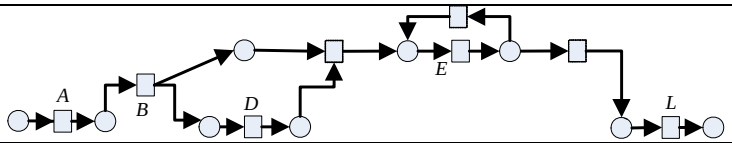
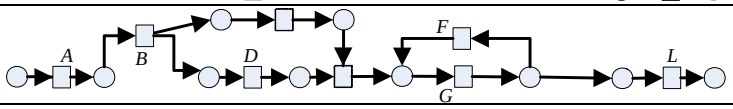
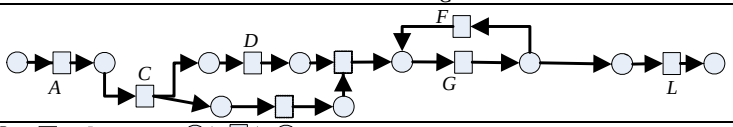
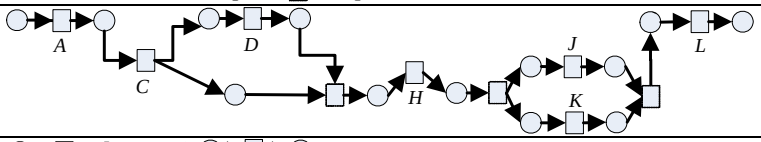
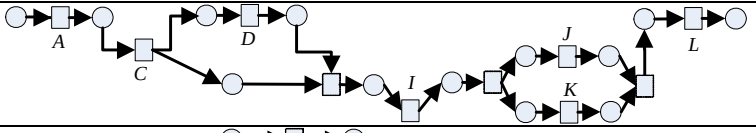
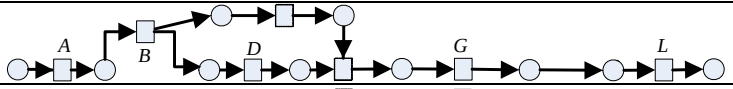
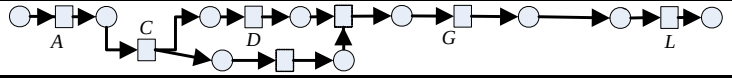
Table 1 Running trace information of cases

表 1 案例运行轨迹

轨迹 ID	任务轨迹	频度	轨迹 ID	任务轨迹	频度
σ_1	$\langle A,B,D,E,L\rangle$	10	σ_6	$\langle A,C,D,H,K,J,L\rangle$	10
σ_2	$\langle A,B,D,E,E,E,L\rangle$	20	σ_7	$\langle A,C,D,I,J,K,L\rangle$	10
σ_3	$\langle A,B,D,G,F,G,F,G,L\rangle$	10	σ_8	$\langle A,C,D,I,K,J,L\rangle$	10
σ_4	$\langle A,C,D,G,F,G,F,G,L\rangle$	10	σ_9	$\langle A,B,D,G,L\rangle$	10
σ_5	$\langle A,C,D,H,J,K,L\rangle$	10	σ_{10}	$\langle A,C,D,G,L\rangle$	10

Table 2 Case models

表 2 案例模型

任务集合	子事件日志	案例模型
$T_1=\{A,B,D,E,L\}$	$[\sigma_1^{10},\sigma_2^{20}]$	
$T_2=\{A,B,D,F,G,L\}$	$[\sigma_3^{10}]$	
$T_3=\{A,C,D,G,F,L\}$	$[\sigma_4^{10}]$	
$T_4=\{A,C,D,H,J,K,L\}$	$[\sigma_5^{10},\sigma_6^{10}]$	
$T_5=\{A,C,D,I,J,K,L\}$	$[\sigma_7^{10},\sigma_8^{10}]$	
$T_6=\{A,B,D,G,L\}$	$[\sigma_9^{10}]$	
$T_7=\{A,C,D,G,L\}$	$[\sigma_{10}^{10}]$	

显然,整体流程模型中任务间的因果、并发、长距离依赖关系会在案例模型中得到保持,而互斥和循环结构则会减少.由此,案例模型的规模以及控制结构的复杂度较整体流程模型会有所降低,案例模型的挖掘难度也会变小;而且,各案例模型的挖掘可并行进行.基于这一思路,后文首先给出案例模型的挖掘方法;之后,将各案例模型进行融合以得到完整的业务流程模型.

2 案例模型的挖掘

本节首先给出案例模型所对应任务依赖图以及 C-net 模型[1]的挖掘方法,挖掘过程与文献[5]中的柔性启发式流程挖掘算法相近,仅在 2-循环结构与任务绑定信息的处理上进行了细化.

2.1 任务依赖图重构

下面先由案例模型对应的案例运行轨迹导出任务依赖关系以及循环结构,在此基础上重构任务依赖图.

(1) 因果关系挖掘

在案例模型中,如果任务 a 与 b 不在一个循环结构内,且 a 到 b 存在因果关系,则显然在该案例模型对应的日志子集中会多次出现 a 与 b 相继发生的情况,除非噪音 b 不会出现在 a 前.由此,可通过如下指标度量两个任务间是否存在因果关系:

定义 1(相继因子)[4]. 设 $L|_T$ 为某案例模型对应的日志子集,a 与 b 是 T 中两不同的任务,记轨迹 σ 在 $L|_T$ 中的频度为 $L|_T(\sigma)$,a 与 b 相继发生的次数为 $|a>_{L|_T} b|=\sum_{\sigma\in L|_T} L|_T(\sigma)\times|\{1\leqslant i\leqslant|\sigma|\,|\,\sigma(i)=a\wedge\sigma(i+1)=b\}|$.称

$$a\Rightarrow_{L|_T} b=\frac{|a>_{L|_T} b|-|b>_{L|_T} a|}{|a>_{L|_T} b|+|b>_{L|_T} a|+1}$$

为当前案例模型中 a 到 b 的相继因子.

以 T_1 对应的案例模型为例,由日志子集 $[\sigma_1^{10},\sigma_2^{20}]$ 可得 A 到 B 的相继因子 $A\Rightarrow_{L|_{T_1}} B=\frac{30-0}{30+0+1}=0.968$,其他任务间的相继因子类似可得,见表 3.

Table 3 Task successive factors in the case model corresponding to T_1

表 3 T_1 所对应案例模型中的任务相继因子

⇒	*A*	*B*	*D*	*E*	*L*
A		**0.968**	0	0	0
B	−0.968		**0.968**	0	0
D	0	−0.968		**0.968**	0
E	0	0	−0.968		**0.968**
L	0	0	0	−0.968	

显然,相继因子 $a\Rightarrow_{L|_T} b$ 的取值介于−1~1 之间:

- $a\Rightarrow_{L|_T} b$ 越接近 1,则 a 与 b 相继发生的次数越多、而 b 在 a 前发生的次数越少.由此一来,可设定一个接近 1 的阈值,只要相继因子超过该阈值,就可判定 a 到 b 具有因果关系;
- $a\Rightarrow_{L|_T} b$ 越接近−1,则意味着相反的情况,b 是因,而 a 为果;
- $a\Rightarrow_{L|_T} b$ 越接近 0,则说明 a 在 b 前发生的次数与 b 在 a 前发生的次数越接近,此时两者不具有因果关系.

(2) 循环结构挖掘

对长度大于 2 的循环结构而言,如由任务 a,b,c 顺序构成的循环结构,案例轨迹中会多次出现类似 *abcabc*…*abc* 的执行序列,此时,相继因子 $a\Rightarrow_{L|_T} b,b\Rightarrow_{L|_T} c$ 和 $c\Rightarrow_{L|_T} a$ 的取值都将接近 1,从而可以得到循环结构导致的 a 到 b、b 到 c、c 到 a 的因果关系,进而重构出原来的循环结构.

对长度等于 2 的循环结构(例如 T_2 所对应案例模型中由 F,G 构成的循环)而言,虽然循环中的两个任务相互具有因果关系,但两者在案例轨迹中不断交替出现,此时,$F\Rightarrow_{L|_{T_2}} G=G\Rightarrow_{L|_{T_2}} F=\frac{20-20}{20+20+1}=0$,从而无法得到由该循环导致的 F 到 G、G 到 F 的因果关系.为此,下面为长度为 2 的循环单独定义新的度量指标:

定义 2(2-循环因子)[4]. 设 $L|_T$ 为某案例模型对应的日志子集,a 与 b 是 T 中两不同的任务,记 $L|_T$ 中序列 *aba* 的频度为 $|a>>_{L|_T} b|=\sum_{\sigma\in L|_T} L|_T(\sigma)\times|\{1\leqslant i<|\sigma|-1\,|\,\sigma(i)=a\wedge\sigma(i+1)=b\wedge\sigma(i+2)=a\}|$,称

$$a\Rightarrow^2_{L|_T} b=\frac{|a>>_{L|_T} b|+|b>>_{L|_T} a|}{|a>>_{L|_T} b|+|b>>_{L|_T} a|+1}$$

为当前案例模型中 a 与 b 的 2-循环因子.

以 T_2 所对应案例模型中的任务 F 与 G 为例,2-循环因子 $F \Rightarrow^{2}_{L|_{T_2}} G = \dfrac{10+20}{10+20+1} = 0.968$ 接近 1,据此可以推断 F 与 G 构成一个长度为 2 的循环.然而,图 2 所示的两种结构都对应着一个长度为 2 的循环,两者都会导致循环中两任务的 2-循环因子取值接近 1.此时,仅靠 2-循环因子无法区分两种结构.不过,图 2(a)对应的案例运行轨迹中,a 的首次执行可能在 b 的首次执行前,也可能相反,两种情况机会均等;而图 2(b)对应的案例轨迹中,a 的首次执行在无噪音时会始终在 b 的首次执行前.由此,可通过如下指标区分两种结构:

定义 3(并发校正因子). 设 $L|_T$ 为一个案例模型对应的事件日志子集,a 与 b 是 T 中两不同的任务,记 $L|_T$ 中 a 的首次执行先于 b 首次执行的发生次数为 $|a \propto_{L|_T} b|$,称

$$|a \Uparrow_{L|_T} b| = 1 - \left| \frac{|a \propto_{L|_T} b| - |b \propto_{L|_T} a|}{|a \propto_{L|_T} b| + |b \propto_{L|_T} a| + 1} \right|$$

为当前案例模型中 a 与 b 的并发校正因子.

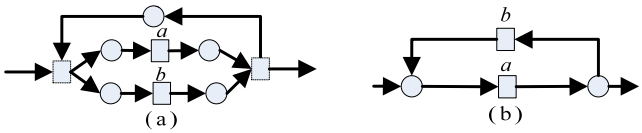


Fig.2 Two process structures leading to the confusion of length-2-loop factors

图 2 导致 2-循环因子混淆的两种结构

在 T_2 所对应案例模型中,并发校正因子 $|F \Uparrow_{L|_T} G| = 1 - \left| \dfrac{10-0}{10+0+1} \right| = 0.091$,由此可判定 F 和 G 构成的是一个长度为 2 的循环结构,存在 F 到 G 以及 G 到 F 的因果关系,不存在 F 与 G 之间的并发关系;反之,如果并发校正因子接近 1,则说明两任务具有图 2(b)所示的结构,此时,两任务间具有并发而非因果关系.

此外,对于案例模型中存在的长度为 1 的循环结构,其度量指标如下:

定义 4(1-循环因子)[4]. 设 $L|_T$ 为一个案例模型对应的事件日志子集,a 是 T 中一个任务,$L|_T$ 中任务序列 aa 出现的次数记为 $|a >_{L|_T} a| = \sum_{\sigma \in L|_T} L|_T(\sigma) \times |\{1 \leqslant i \leqslant |\sigma| \,|\, \sigma(i) = a \wedge \sigma(i+1) = a\}|$,称

$$a \Rightarrow^{1}_{L|_T} a = \frac{|a >_{L|_T} a|}{|a >_{L|_T} a| + 1}$$

为当前案例模型中任务 a 的 1-循环因子.

$a \Rightarrow^{1}_{L|_T} a$ 的取值越接近 1,则序列 aa 的出现次数越大,可认为 a 自身构成长度为 1 的循环.在 T_1 所对应案例模型中,1-循环因子 $E \Rightarrow^{1}_{L|_T} E = \dfrac{40}{40+1} = 0.976$,由此可判定 E 自身构成循环,E 到其自身存在因果关系.

(3) 任务依赖图导出

在前述相继因子与循环结构挖掘结果的基础上,设置相继因子、2-循环因子、并发校正因子和 1-循环因子的阈值后,便可得任务依赖关系的集合.例如,对日志 $L|_{T_1} = [\sigma_1^{10}, \sigma_2^{20}]$ 而言,设置各因子阈值为 0.9 后,可得任务依赖关系集$\{\langle A,B\rangle, \langle B,D\rangle, \langle D,E\rangle, \langle E,E\rangle, \langle E,L\rangle\}$,该依赖关系集对应的任务依赖图如图 3 所示.

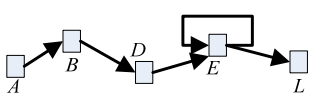


Fig.3 Task dependency graph mined from the sub-log corresponding to T_1

图 3 由 T_1 所对应日志挖掘到的任务依赖图

当日志规模较小或者日志中噪音较多时,原本具有依赖关系的两个任务可能相继因子取值偏小,或者用户

设置的阈值过高也会导致原本存在的任务依赖关系无法挖掘得到.此时,可通过传统启发式流程挖掘算法中的如下措施进行处理:

- 第一,假设所有任务都有前驱任务,则对于任意任务 a,如果任务 b 满足 $\forall t \in T: b \Rightarrow_{L|_T} a \geqslant t \Rightarrow_{L|_T} a$,则即使 $b \Rightarrow_{L|_T} a$ 未超过阈值,也在任务依赖图中添加由 b 到 a 的有向弧;
- 第二,假设所有任务都有后继任务,对于任务 a,如果 b 满足 $\forall t \in T: a \Rightarrow_{L|_T} b \geqslant a \Rightarrow_{L|_T} t$,则即使 $a \Rightarrow_{L|_T} b$ 未超过阈值,也在任务依赖图中添加由 a 到 b 的有向弧;
- 第三,对于任意任务 a 与 b,设任务 t 使 $t \Rightarrow_{L|_T} a$ 取最大值,则只要 $t \Rightarrow_{L|_T} a - b \Rightarrow_{L|_T} a$ 小于一个趋近 0 的最佳相对偏离因子 δ_r,则在任务依赖图中添加由 b 到 a 的有向弧;
- 第四,设任务 t 使 $a \Rightarrow_{L|_T} t$ 取最大值,则只要 $a \Rightarrow_{L|_T} t - a \Rightarrow_{L|_T} b$ 小于 δ_r,则在任务依赖图中添加由 a 到 b 的有向弧.

在上述处理过程中,为保证各任务有前驱和后继,需对日志进行预处理:在每个案例的运行轨迹中添加起始任务 *start* 和终止任务 *end*;之后,计算任务间的 1-循环因子、2-循环因子、并发校正因子和相继因子,再结合指定的各因子阈值和最佳相对偏离因子阈值确定流程中存在的循环结构和因果关系;最后,由挖掘到的循环结构和因果关系导出任务间的依赖关系集以及各案例模型对应的任务依赖图,具体步骤见算法 1.相比文献[5]中任务依赖图的挖掘算法,本算法根据并发校正因子对 2-循环结构进行了细化.

算法 1. 案例模型所对应任务依赖图的挖掘.

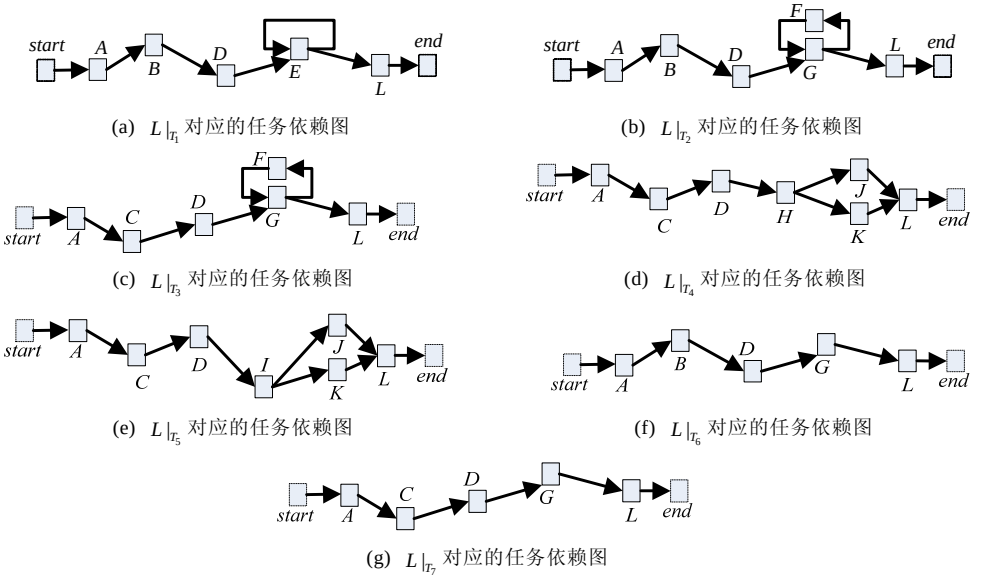
输入:任务集合 T 所对应案例模型的事件日志子集 $L|_T$;

输出:案例模型所对应任务依赖图中任务间因果依赖关系的集合 *Causal*.

步骤:

1. 设定相继因子、1-循环和 2-循环因子、并发校正因子及最佳相对偏离因子阈值 $\delta_{\Rightarrow}, \delta_{L1L}, \delta_{L2L}, \delta_{\Uparrow}, \delta_r$
2. $T := T \cup \{start, end\}$
3. FOREACH ($\sigma \in L_T$) σ:=start$\circ\sigma\circ$end;其中,$\circ$为轨迹拼接运算,用以在各案例轨迹的开始和末尾添加起始任务 start 和终止任务 end;
4. 令 $Loop_1 := \{(a,a) \in T \times T \mid a \Rightarrow^1_{L|_T} a \geqslant \delta_{L1L}\}$,用以记录各个长度为 1 的循环结构;
5. 令 $Loop_{2\_b} := \{(a,b) \in T \times T \mid (a,a) \notin Loop_1 \wedge (b,b) \notin Loop_1 \wedge a \Rightarrow^2_{L|_T} b \geqslant \delta_{L2L} \wedge a \Uparrow_{L|_T} b < \delta_{\Uparrow}\}$,用以记录各个形如图 2(b)的长度为 2 的循环结构;
6. 令 $StrongestFollow := \{(a,b) \in T \times T \mid a \neq end \wedge a \neq b \wedge (\forall y \in T: a \Rightarrow_{L|_T} b \geqslant a \Rightarrow_{L|_T} y)\}$,记录各个任务与其最强后继任务构成的因果关系;
7. 令 $StrongestCause := \{(a,b) \in T \times T \mid b \neq start \wedge a \neq b \wedge (\forall x \in T: a \Rightarrow_{L|_T} b \geqslant x \Rightarrow_{L|_T} b)\}$,记录各个任务与其前最强前驱任务构成的因果关系;
8. 令 $WeakOutgoing := \{(a,x) \in StrongestFollow \mid (a \Rightarrow_{L|_T} x < \delta_{\Rightarrow}) \wedge \exists_{(b,y) \in StrongestFollow}[(a,b) \in Loop_{2\_b} \wedge (b \Rightarrow_{L|_T} y - a \Rightarrow_{L|_T} x) > \delta_r]\}$;
9. 令 $StrongestFollow := StrongestFollow - WeakOutgoing$;
10. 令 $WeakIncoming := \{(x,a) \in StrongestCause \mid (x \Rightarrow_{L|_T} a < \delta_{\Rightarrow}) \wedge \exists_{(y,b) \in StrongestCause}[(a,b) \in Loop_{2\_b} \wedge (y \Rightarrow_{L|_T} b - x \Rightarrow_{L|_T} a) > \delta_r]\}$;
11. 令 $StrongestCause := StrongestCause - WeakIncoming$;
12. 令 $Follow := \{(a,b) \in T \times T \mid (a \Rightarrow_{L|_T} b \geqslant \delta_{\Rightarrow}) \vee \exists_{(a,c) \in StrongestFollow}(a \Rightarrow_{L|_T} c - a \Rightarrow_{L|_T} b < \delta_r)\}$;
13. 令 $Cause := \{(b,a) \in T \times T \mid (b \Rightarrow_{L|_T} a \geqslant \delta_{\Rightarrow}) \vee \exists_{(c,a) \in StrongestCause}(c \Rightarrow_{L|_T} a - b \Rightarrow_{L|_T} a < \delta_r)\}$;
14. 令 $Loop_{2\_a} := \{(a,b) \in T \times T \mid (a,a) \notin Loop_1 \wedge (b,b) \notin Loop_1 \wedge a \Rightarrow^2_{L|_T} b \geqslant \delta_{L2L} \wedge a \Uparrow_{L|_T} b \geqslant \delta_{\Uparrow}\}$,用以记录各个形如图 2(*a*)的长度为 2 的循环结构;
15. 返回 $Causal := Follow \cup Cause \cup Loop_1 \cup Loop_{2\_a} \cup Loop_{2\_b}$.

需要指出的是,各因子阈值的设置通常根据事件日志的质量以及挖掘结果进行确定:日志噪音较少或者用户希望挖掘到更为精简的流程模型(依赖关系更少)时,阈值设置应越接近 1;反之,阈值应设置为小一些的值.一般而言,在保证每个任务都有前驱与后继的前提下,阈值越接近 1,挖掘到的模型中依赖关系越少;反之越多.根据算法 1,当设置各因子阈值为 0.9 时,可得图 4 中各案例模型对应的任务依赖图.



(a) $L|_{T_1}$ 对应的任务依赖图　(b) $L|_{T_2}$ 对应的任务依赖图

(c) $L|_{T_3}$ 对应的任务依赖图　(d) $L|_{T_4}$ 对应的任务依赖图

(e) $L|_{T_5}$ 对应的任务依赖图　(f) $L|_{T_6}$ 对应的任务依赖图

(g) $L|_{T_7}$ 对应的任务依赖图

Fig.4　Task dependency graphs obtained by Algorithm 1 when setting the threshold to 0.9

图 4　阈值设为 0.9 时,由算法 1 挖掘到的任务依赖图

2.2　分支与合并结构的挖掘

在案例模型对应的任务依赖图中,任务节点的后继可能有多个.例如,图 4(e)中任务 *I* 的后继有 *J* 和 *K* 两个任务,这有两种可能:(1) *I* 结束后,并发使能 *J* 和 *K*;(2) *I* 结束后,*J* 与 *K* 择一执行.具体是哪一种情况,需要结合案例运行轨迹确定.在图 4(e)所对应的日志 $L|_{T_5}$ 中,案例在执行完毕 *I* 后会相继执行 *J* 和 *K*,由此可认为 *I* 之后的两个分支是并行分支,此时记$[\{J,K\}^{20}]$为任务 *I* 的后继绑定;如果情况相反,各案例在执行完毕 *I* 后在 *J* 与 *K* 中间择一执行,则认为 *I* 之后的两个分支是互斥分支,此时记$[\{J\}^{20},\{K\}^{20}]$为 *I* 的后继绑定.其中,花括号内部的任务构成并行分支,逗号分隔的多组任务之间构成互斥分支,每组任务右上角的数值为相应并行分支被执行的次数.存在两类特殊情况:

(1) 任务依赖图中,任务 *b* 与 *c* 都是 *a* 的后继;但与此同时,*c* 也是 *b* 的后继.此时,如果案例运行轨迹中任务 *a*,*b*,*c* 依次出现,则认为 *a* 只使能 *b* 而非并发使能 *b* 与 *c*,*c* 由 *b* 使能;

(2) 任务依赖图中,*b* 与 *c* 都是 *a* 的后继,而在案例运行轨迹中 *a* 多次出现.此时,如果 *a* 的任意两次执行之间 *b* 与 *c* 均只有一个被选择执行,则认为 *a* 之后 *b* 与 *c* 存在互斥分支.

由此一来,可根据任务依赖图与案例运行轨迹得到案例模型中各个任务后继分支结构的种类.

类似可求取任务依赖图中各个任务所对应前驱合并结构的种类以及各个任务的前驱绑定.例如,图 4(e)中 *L* 有 *J* 和 *K* 两个前驱任务,由于在 $L|_{T_5}$ 的所有案例运行轨迹中 *L* 执行前都会相继执行 *J* 与 *K*,由此可断定 *L* 之前 *J* 与 *K* 构成的合并结构是并行结构而非互斥结构;相应地,*L* 的前驱绑定为$[\{J,K\}^{20}]$.基于上述分析,可得由案例模型的任务依赖图及其对应的事件日志挖掘各任务前驱绑定与后继绑定信息的算法如下:

算法 2. 案例模型中,各任务前驱绑定与后继绑定信息的挖掘.

输入:任务集合 *T* 所对应案例模型的任务依赖图 $Causal|_T$ 及其相应的事件日志子集 $L|_T$;

输出:各个任务 t 的前驱绑定 $PreBinding(t)$与后继绑定 $PostBinding(t)$.

步骤:

1. **FOREACH** ($\sigma \in L|_T$){
2. **FOR** (i=1; $i<\sigma.length-1$; ++i){ //计算当前案例运行轨迹中σ_i的后继绑定信息
3. $postActivitySet:=\varnothing$
4. **FOR** ($j=i+1$; $j<\sigma.length$; ++j){
5. if ($\sigma_j==\sigma_i$) **break**;
6. if ((σ_i,σ_j)$\in Causal|_T$){
7. if ($\forall k:(i<k<j)\wedge(\sigma_k\in postActivitySet)\rightarrow(\sigma_k,\sigma_j)\notin(Causal|_T)^*$){ //*为关系闭包运算
8. $postActivitySet:=postActivitySet\cup\{\sigma_j\}$
9. } //if
10. } //if
11. } //for
12. $PostBinding(\sigma_i):=PostBinding(\sigma_i)+postActivitySet$; //此处+为袋集的求和运算
13. } //for
14. **FOR** (i=2; $i<\sigma.length$; ++i){ //计算当前案例运行轨迹中σ_i的前驱绑定信息
15. $preActivitySet:=\varnothing$
16. **FOR** ($j=i-1$; $j>=1$; --j){
17. if ($\sigma_j==\sigma_i$) **break**;
18. if ((σ_j,σ_i)$\in Causal|_T$){
19. if ($\forall k:(j<k<i)\wedge(\sigma_k\in preActivitySet)\rightarrow(\sigma_j,\sigma_k)\notin(Causal|_T)^*$){
20. $preActivitySet:=preActivitySet\cup\{\sigma_j\}$
21. } //if
22. } //if
23. } //for
24. $PreBinding(\sigma_i):=PreBinding(\sigma_i)+preActivitySet$;
25. } //for

//计算各个 1-循环结构所对应的任务的前驱和后继绑定信息

26. **FOR** (i=1; $i<\sigma.length$; ++i){
27. **IF** ($\sigma_i==\sigma_{i+1}\wedge(\sigma_i,\sigma_i)\in Causal|_T$){
28. $PreBinding(\sigma_i):=PreBinding(\sigma_i)+\{\sigma_i\}$;
29. $PostBinding(\sigma_i):=PostBinding(\sigma_i)+\{\sigma_i\}$;
30. } //if
31. } //for
32. } //foreach
33. 返回 $PreBinding$ 与 $PostBinding$

分别以任务集 T_1,T_5 所对应的任务依赖图与事件日志为输入执行算法 2,可得这两个案例模型中各个任务的前驱绑定与后继绑定信息,见表 4、表 5.表 4 中,任务 E 的前驱绑定[$\{D\}^{30}$,$\{E\}^{40}$]意味着 E 的执行有 30 次是被 D 使能的,另有 40 次是被 E 使能,两者构成一种互斥的合并结构.表 5 中任务 I 的后继绑定[$\{J,K\}^{20}$]意味着 I 执行完毕后有 20 次并发使能任务 J 与 K,从未单独使能其中之一,由此构成一种并行分支结构.

根据表 4 与表 5 所示的任务绑定信息,在任务依赖图上标注相应绑定以及流关系的执行次数,同时对并行分支与合并结构进行标注,如此可得图 5 所示模型.其中,有向弧上标记的数值为相应流关系的激活次数,实心圆

点表达绑定信息,用直线相连的多个实心圆点表示并行分支或并行合并结构,圆点或者连接圆点的直线上标注的数值为绑定的激活次数.文献[5]中将图 5 形式的模型称为 C-net,并给出了其形式化定义.而 C-net 实际上等同于表 4、表 5 所示任务绑定信息的一种形式化描述,限于篇幅,本文不再赘述.

Table 4 Task pre-binding and post-binding information in the case model corresponding to T_1

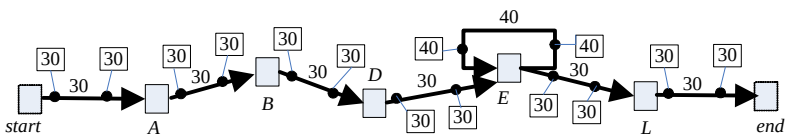
表 4 T_1 所对应案例模型的任务前驱与后继绑定

前驱绑定	任务	后继绑定
$[\{start\}^{30}]$	A	$[\{B\}^{30}]$
$[\{A\}^{30}]$	B	$[\{D\}^{30}]$
$[\{B\}^{30}]$	D	$[\{E\}^{30}]$
$[\{D\}^{30},\{E\}^{40}]$	E	$[\{E\}^{40},\{L\}^{30}]$
$[\{E\}^{30}]$	L	$[\{end\}^{30}]$

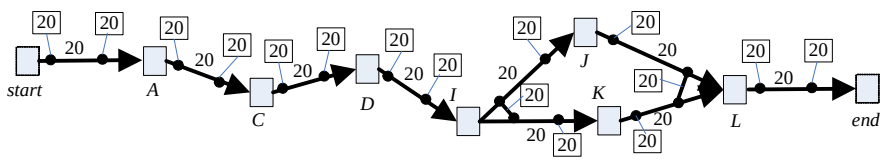
Table 5 Task pre-binding and post-binding information in the case model corresponding to T_5

表 5 T_5 所对应案例模型的任务前驱与后继绑定

前驱绑定	任务	后继绑定
$[\{start\}^{20}]$	A	$[\{B\}^{20}]$
$[\{A\}^{20}]$	C	$[\{D\}^{20}]$
$[\{C\}^{20}]$	D	$[\{I\}^{20}]$
$[\{D\}^{20}]$	I	$[\{J,K\}^{20}]$
$[\{I\}^{20}]$	J	$[\{L\}^{20}]$
$[\{I\}^{20}]$	K	$[\{L\}^{20}]$
$[\{J,K\}^{20}]$	L	$[\{end\}^{20}]$



(a) 任务集 T_1 所对应案例模型的 C-net



(b) 任务集 T_5 所对应案例模型的 C-net

Fig.5 C-nets corresponding to case models

图 5 案例模型对应的 C-net

3 模型融合与长距离依赖关系的挖掘

3.1 模型融合

对于任务 t,假设其在任务集 T_i 所对应案例模型中的前驱绑定为 $\bullet t|_{T_i}=[T_{11\_i}^{k_{11}},...,T_{1m\_i}^{k_{1m}}]$,后继绑定为 $t\bullet|_{T_i}=[T_{21\_i}^{k_{21}},...,T_{2n\_i}^{k_{2n}}]$,则就整个事件日志而言,$t$ 的前驱绑定为 $\bullet t=\sum_{T_i}(\bullet t|_{T_i})$,后继绑定为 $t\bullet=\sum_{T_i}(t\bullet|_{T_i})$,例如:

$$\bullet A|_{T_1}=[\{start\}^{30}],\bullet A|_{T_2}=\bullet A|_{T_3}=\bullet A|_{T_6}=\bullet A|_{T_7}=[\{start\}^{10}],\bullet A|_{T_4}=\bullet A|_{T_5}=[\{start\}^{20}].$$

从而,就整个日志而言,$\bullet A=[\{start\}^{110}]$.由各案例模型所对应之任务绑定信息计算整个事件日志所对应任务绑定信息(亦即整个事件日志所对应的 C-net)的算法如下:

算法 3. 整个事件日志所对应 C-net 模型中任务前驱绑定与后继绑定信息的挖掘.

输入:各个案例模型所对应的任务的前驱绑定信息 $PreBinding|_T(t)$与后继绑定信息 $PostBinding|_T(t)$;

输出:整个事件日志所对应的各个任务 t 的前驱绑定 $PreBinding|_L(t)$与后继绑定 $PostBinding|_L(t)$.

步骤:

1. **FOREACH** (日志 L 中出现的任务 t){
2. $PreBinding|_L(t):=PostBinding|_L(t):=\varnothing$
3. **FOREACH** (任务集 T 对应的案例模型){
4. $PreBinding|_L(t):=PreBinding|_L(t)+PreBinding|_T(t)$; //此处+为袋集的求和运算
5. $PostBinding|_L(t):=PostBinding|_L(t)+PostBinding|_T(t)$;
6. }
7. }
8. 返回 $PreBinding|_L$ 与后继绑定 $PostBinding|_L$

以图 4 各案例模型对应的任务绑定为输入执行算法 3,可得整个事件日志对应的任务绑定信息见表 6,进而可得融合后的 C-net 模型如图 6 所示.

Table 6 Task pre-binding and post-binding information corresponding to the entire event log

表 6 整个事件日志所对应的任务前驱与后继绑定信息

前驱绑定	任务	后继绑定
$[\{start\}^{110}]$	A	$\mathbf{[\{B\}^{50},\{C\}^{60}]}$
$[\{A\}^{50}]$	B	$[\{D\}^{50}]$
$[\{A\}^{60}]$	C	$[\{D\}^{60}]$
$[\{B\}^{50},\{C\}^{60}]$	$\boldsymbol{D}$	$\mathbf{[\{E\}^{30},\{G\}^{40},\{H\}^{20},\{I\}^{20}]}$
$[\{D\}^{30},\{E\}^{40}]$	$\boldsymbol{E}$	$\mathbf{[\{E\}^{40},\{L\}^{30}]}$
$[\{G\}^{40}]$	F	$[\{G\}^{40}]$
$[\{D\}^{40},\{F\}^{40}]$	$\boldsymbol{G}$	$\mathbf{[\{F\}^{40},\{L\}^{40}]}$
$[\{D\}^{20}]$	H	$[\{J,K\}^{20}]$
$[\{D\}^{20}]$	I	$[\{J,K\}^{20}]$
$[\{H\}^{20},\{I\}^{20}]$	J	$[\{L\}^{40}]$
$[\{H\}^{20},\{I\}^{20}]$	K	$[\{L\}^{40}]$
$[\{E\}^{30},\{G\}^{40},\{J,K\}^{40}]$	L	$[\{end\}^{110}]$

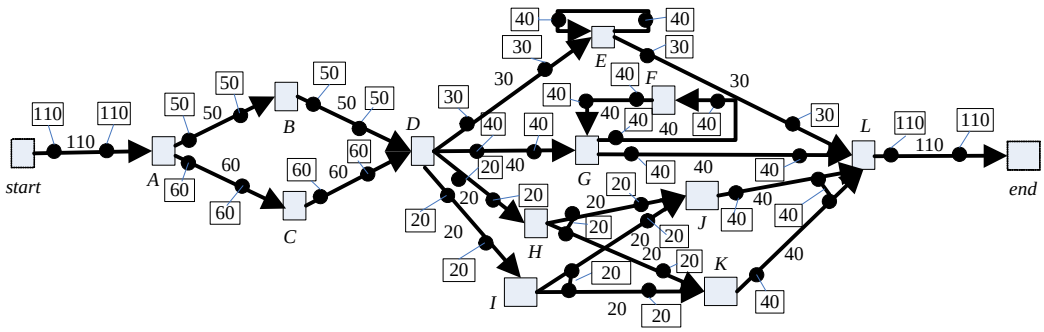


Fig.6 The integrated C-net model

图 6 融合后的 C-net 模型

3.2 长距离依赖关系挖掘

进行 C-net 模型融合时,可能会丢失原本存在的长距离依赖关系.如图 1 流程中,{B}到{E}、{C}到{H}、{C}到{I}的长距离依赖关系在图 6 所示 C-net 模型中无法体现.本节给出上述长距离依赖关系的挖掘方法.

文献[4,5]指出长距离依赖关系是两个决策点处任务选择间的非局部依赖关系,未给出严格定义.文献[8,9]认为两个任务只要具有间接依赖关系便具有长距离依赖关系,结合 WF-net 模型给出了长距离依赖关系的形式化定义.按此定义,图 1 流程中,任务 A 与其直接后继 B,C 以外的任何一个任务都存在长距离依赖关系.相比文献[4,5]将长距离依赖限定在两个决策点而言,这种定义会使模型过于复杂.为此,本文仍将长距离依赖限定为两个

决策点处的非局部依赖关系.同时,文献[4,5,8,9]将长距离依赖关系限定在两个单独的任务之间,而实际上,决策点处即使单个的选择分支也可能对应多个任务.为此,本文将长距离依赖关系约定为两任务集在决策点处的非局部依赖关系.下面结合 C-net 模型给出长距离依赖关系的定义.

定义 5(长距离依赖关系). 设 N 为一个 C-net 模型,S 与 T 是模型中的两个任务子集,$PreS$ 与 $PreT$ 分别是 S 与 T 中各任务前驱绑定中的一个任务集.如果下列条件满足,则称$(PreS,S)$到$(PreT,T)$存在长距离依赖关系,或者说 S 与 T 在决策点 $PreS$ 与 $PreT$ 处有长距离依赖关系,记作$(PreS,S)\prec(PreT,T)$:

(1) $S\neq T$;

(2) 存在任务集 $S'\neq S$,使得 $PreS$ 同时是 S 和 S'中各任务前驱绑定中的一个任务集,且 S 和 S'是 $PreS$ 中各任务后继绑定中的任务集之一;

(3) 存在任务集 $T'\neq T$,使得 $PreT$ 同时是 T 和 T'中各任务前驱绑定中的一个任务集,且 T 和 T'是 $PreT$ 中各任务后继绑定中的任务集之一;

(4) 在 $PreT$ 中的任务执行完毕后,T 中任务被选择执行的一个必要条件是:$PreS$ 中各任务执行完毕后,S 中的任务被选择执行.

图 6 所示的 C-net 模型中,令 $S=\{B\}$,$T=\{E\}$,$PreS:=\{A\}$,$PreT:=\{D\}$,$S'=\{C\}$,$T'=\{G\}$,则不难发现,定义 5 中的前 3 个关系均满足.而且,由该模型对应的原始流程定义(对应图 1 中 WF-net)可知,在$\{D\}$执行完毕后执行$\{E\}$的一个必要条件是:$\{A\}$中各任务执行完毕后,$\{B\}$中的任务被选择执行.由此可见,图 6 的 C-net 模型中,决策点$\{A\}$与$\{D\}$处应存在$\{B\}$到$\{E\}$的长距离依赖关系,即:$(\{A\},\{B\})\prec(\{D\},\{E\})$.

下面介绍长距离依赖关系的挖掘方法.根据定义 5,假设任务集 S 与 T 在决策点 $PreS$ 与 $PreT$ 处存在长距离依赖关系,则在没有噪音的情况下,对于 $PreT$ 执行完毕后选择执行 T 的案例而言,之前必然会在 $PreS$ 执行完毕后选择执行 S.因此,可通过如下指标对 S 与 T 在决策点 $PreS$ 与 $PreT$ 处的长距离依赖关系进行度量:

定义 6(长距离依赖因子). 设 L 为某流程对应的事件日志,$PreS$ 与 $PreT$ 是流程中的两个任务集,且 $PreS$ 与 $PreT$ 中任务的后继绑定由多个互斥的任务子集构成,S 与 T 分别是 $PreS$ 与 $PreT$ 后继绑定中的一个任务子集,且两者不等.记$|(PreS,S)\triangleright_L(PreT,T)|$为 L 中 $PreT$ 执行完毕后选择执行 T,而且之前在 $PreS$ 执行完毕后选择执行 S 的案例个数;记$|(PreS,S)\triangleleft_L(PreT,T)|$为 L 中 $PreT$ 执行完毕后选择执行 T,而之前在 $PreS$ 执行完毕后未选择执行 S 的案例个数.称

$$(PreS,S)\mapsto_L(PreT,T)=\frac{|(PreS,S)\triangleright_L(PreT,T)|-|(PreS,S)\triangleleft_L(PreT,T)|}{|(PreS,S)\triangleright_L(PreT,T)|+|(PreS,S)\triangleleft_L(PreT,T)|+1}$$

为 S 与 T 在决策点 $PreS$ 与 $PreT$ 处的长距离依赖因子.

对日志 L 而言,$(\{A\},\{B\})\mapsto_L(\{D\},\{E\})=\frac{30-0}{30+0+1}=0.968$.假设长距离依赖因子的阈值设置为 0.9,则可认定$\{B\}$与$\{E\}$在决策点$\{A\}$与$\{D\}$处存在长距离依赖关系,对应图 7 中由结点 B 指向结点 E 的点划线形状的有向弧.类似地,$(\{A\},\{B\})\mapsto_L(\{D\},\{G\})=\frac{20-20}{20+20+1}=0$,据此可以推定$\{B\}$与$\{G\}$在决策点$\{A\}$与$\{D\}$处不存在长距离依赖关系.

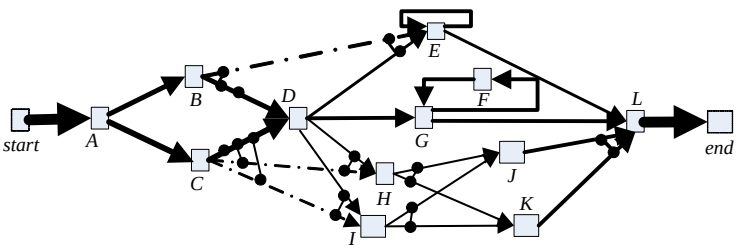


Fig.7 The C-net model combined with long distance dependencies

图 7 添加长距离依赖后的 C-net 模型

为挖掘 C-net 模型中的长距离依赖关系,可以先找出所有决策点及其对应的决策分支,将其存入集合 *DecisionBranchList*={(*PreT*,*T*)|任务集 *PreT* 中的每个任务都包含多个后继分支,且任务集 *T* 为这些任务的后继分支之一};之后,针对该集合中的任意两个决策分支(*PreS*,*S*)与(*PreT*,*T*),计算两者之间的长距离依赖因子(*PreS*,*S*)$\mapsto_L$(*PreT*,*T*),如果该因子取值大于阈值,则说明两者间存在长距离依赖关系;最后,每发现一个 *S* 与 *T* 在决策点 *PreS* 与 *PreT* 处的长距离依赖关系,都应在 *T* 的前驱绑定分支 *PreT* 中添加 *S* 中的任务,同时,根据 *S* 的后继分支在 *T* 之前被选择执行的情况更新 *S* 的后继绑定.如得到长距离依赖关系({*A*},{*B*})$\prec_L$({*D*},{*E*})后,一方面应将 *E* 的前驱绑定分支{*D*}^{30} 更新为{*B*,*D*}^{30};同时,得到|({*B*},{*D*})$\rhd_L$({*D*},{*E*})|的取值为 30 后,应将 *B* 的后继绑定更新为[{*D*,*E*}^{30},{*D*}^{20}].在上述过程中需注意:计算(*PreS*,*S*)$\mapsto_L$(*PreT*,*T*)时,可以分案例模型并行统计各个案例模型中|(*PreS*,*S*)$\rhd_L$(*PreT*,*T*)|与|(*PreS*,*S*)$\lhd_L$(*PreT*,*T*)|的取值,之后汇总这两个指标在各个案例模型中的取值,在此基础上求取(*PreS*,*S*)$\mapsto_L$(*PreT*,*T*).由此得长距离依赖并行挖掘算法如下:

算法 4. 长距离依赖关系的并行挖掘算法.

输入:事件日志 L,当中的任务集 *AllTasks* 及各任务的前驱绑定与后继绑定信息 *PreBinding*(*t*), *PostBinding*(*t*);

输出:长距离依赖关系集 *LongDistanceDepRelation* 及根据长距离依赖关系更新的任务前驱与后继绑定.

步骤:

1. 令 *DecisionBranchList*:=∅, *LongDistanceDepRelation*:=∅;
2. **FOREACH** (*pre_t*∈*AllTasks*){
3. **IF** (*PostBinding*(*pre_t*)含有多个互斥的分支)
4. **FOREACH** (*PostBinding*(*pre_t*)中的分支 *T*){
5. 令 $SetOfPreT := \bigcap_{t\in T} PreBinding(t)$;
6. **FOREACH** (*PreT*∈*SetOfPreT*)
7. **IF** (*pre_t*∈*PreT*) *DecisionBranchList*:=*DecisionBranchList*∪{(*PreT*,*T*)};
8. }
9. }
10. **FOREACH** (案例模型对应的事件日志子集 L_i) **IN PARALLEL**{
11. **FOREACH** ((*PreS*,*S*)∈*DecisionBranchList*)
12. **FOREACH** ((*PreT*,*T*)∈*DecisionBranchList*)
13. **IF** ((*PreS*,*S*)≠(*PreT*,*T*)) 计算$|(PreS,S)\rhd_{L_i}(PreT,T)|$与$|(PreS,S)\lhd_{L_i}(PreT,T)|$;
14. }
15. 汇总各案例模型中|(*PreS*,*S*)$\rhd$(*PreT*,*T*)|与|(*PreS*,*S*)$\lhd$(*PreT*,*T*)|的取值;
16. **FOREACH** ((*PreS*,*S*)∈*DecisionBranchList*)
17. **FOREACH** ((*PreT*,*T*)∈*DecisionBranchList*)
18. **IF** ((*PreS*,*S*)≠(*PreT*,*T*)∧(*PreS*,*S*)$\mapsto_L$(*PreT*,*T*)>$\delta_{longDisDep}$)
19. *LongDistanceDepRelation*:=*LongDistanceDepRelation*∪{(*PreS*,*S*)$\prec$(*PreT*,*T*)};
20. **FOREACH** ((*PreS*,*S*)$\prec$(*PreT*,*T*)∈*LongDistanceDepRelation*){
21. **FOREACH** (*t*∈*T*)
22. *PreBinding*(*t*):=*PreBinding*(*t*)–[{*Pret_T*}^{x}]+[{*Pre_T*∪*T*}^{x}]; //–为袋集的差运算,*x* 为 *t* 的 //前驱绑定中 *Pre_T* 的执行次数
23. 令 $SetOfPostS := \bigcap_{s\in S} PostBinding(s)$;

24. **FOREACH** ($s\in S$)
25. **FOREACH** (*PostBinding*(s)的分支 *PostS*)
26. **IF** (*PostS*$\in$*SetOfPostS*){
27. 令 k:=|(S,*PostS*)$\triangleright$(*PreT*,*T*)|;
28. *PostBinding*(s):=*PostBinding*(s)−[{*PostS*}^{k}]+[{*PostS*$\cup$*T*}^{k}];
29. }
30. }
31. **RETURN** *LongDistanceDepRelation* 与更新后的各任务前驱与后继绑定信息

根据算法 4,可得如图 6 所示的 C-net 模型中存在 3 个长距离依赖关系,分别是($\{A\},\{B\}$)$\prec_L$($\{D\},\{E\}$),($\{A\},\{C\}$)$\prec_L$($\{D\},\{H\}$)和($\{A\},\{C\}$)$\prec_L$($\{D\},\{I\}$).相应地,表 6 中任务 E,H 与 I 的前驱绑定分别更新为[{B,D}^{30},{E}^{40}],[{C,D}^{20}]和[{C,D}^{20}],任务 B 与 C 的后继绑定分别更新为[{D,E}^{30},{D}^{20}]和[{D,H}^{20},{D,I}^{20},{D}^{20}].最终可得添加长距离依赖关系后的 C-net 模型,如图 7 所示.为清晰起见,图中省略了绑定的执行次数信息,通过流关系所对应有向弧的粗细表达流关系执行次数的多少.显然,按本节所定义的长距离依赖因子正确挖掘出了图 1 中原本存在的长距离依赖关系.然而由于引言中所述的原因,文献[4,5]中的启发式流程挖掘算法无法得到上述长距离依赖关系.

4 模型挖掘算法与性能评价

4.1 模型挖掘的整体框架

第 2 节给出了案例模型所对应 C-net 的挖掘方法,第 3 节给出了案例模型对应 C-net 的融合算法以及模型中长距离依赖关系的并行挖掘算法.在上述方法的基础上,可以得到完整的并行化启发式流程挖掘算法,其整体框架如图 8 所示:

- 首先,根据各个案例的执行任务集将原始事件日志中的案例进行切分,该步骤可以通过多线程技术或者 MapReduce 框架进行并行化,切分后得到各个案例模型对应的事件日志子集;
- 其次,并行地对各个案例模型对应的事件日志子集执行算法 1 与算法 2,由此可以得到各个案例模型所对应的 C-net;
- 接下来,通过算法 3 对各个案例模型对应的 C-net 进行融合;
- 最后,执行算法 4 挖掘模型中的长距离依赖关系,由此得到最终的挖掘结果.

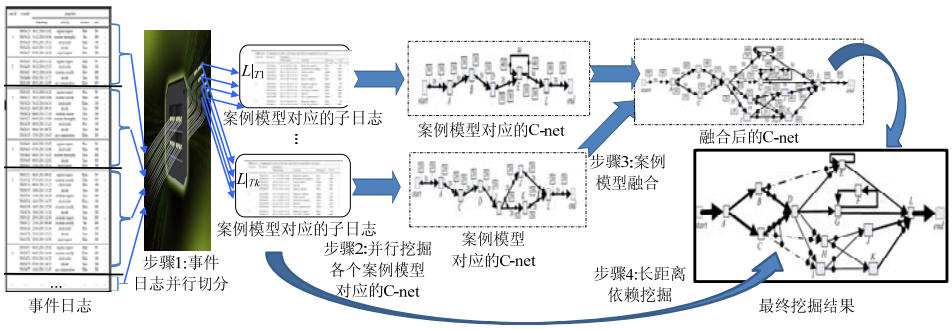


Fig.8 The framework of the parallelized heuristics process mining algorithm

图 8 并行化启发式流程挖掘算法整体框架

4.2 模型挖掘的并行化算法

根据前述并行化启发式流程挖掘算法的整体框架,可得算法的伪代码描述如下:

算法 5. 基于案例模型的并行化启发式流程挖掘算法.

输入:由案例运行轨迹构成的简单事件日志 L;

输出:包含长距离依赖关系的 C-net 模型.

步骤:

1. 初始化集合 *CaseModelSet* 为∅,用以记录各个案例模型对应的任务子集;
2. 初始化日志中出现的所有任务构成的集合 *AllTasks* 为∅;
3. **FOREACH** ($\sigma\in L$) **IN PARALLEL**{
4. 令 T 为σ中出现的任务构成的任务子集;
5. 令 $CaseModelSet:=CaseModelSet\cup\{T\}$; $AllTasks:=AllTasks\cup T$;
6. 令 $L|_T:=L|_T+\{\sigma\}$,用以记录任务集 T 所对应案例模型的事件日志子集;
7. }
8. **FOREACH** ($T\in CaseModelSet$) **IN PARALLEL**{ /*并行计算各案例模型所对应的 C-net 模型*/
9. 以 $L|_T$ 为输入执行算法 1,记得到的依赖关系集为 $Causal|_T$;
10. 以 $L|_T$ 与 $Causal|_T$ 为输入执行算法 2,计算 T 中各任务 t 的前驱绑定 $PreBinding(t)|_T$ 与后继绑定 $PostBinding(t)|_T$
11. }
12. 以所有案例模型对应的任务前驱绑定 $PreBinding|_T(t)$与后继绑定 $PostBinding|_T(t)$为输入执行算法 3,计算整个事件日志所对应的各任务 t 的前驱绑定 $PreBinding|_L(t)$与后继绑定 $PostBinding|_L(t)$,并记 *Causal_all* 为各案例模型对应任务依赖图中任务邻接关系的并集;
13. 以事件日志中各任务的前驱绑定 $PreBinding(t)|_L$ 与后继绑定 $PostBinding(t)|_L$ 为输入执行并行算法 4,挖掘长距离依赖关系并更新任务绑定信息;
14. **FOREACH** (($PreS,S$)$\prec$($PreT,T$)){
15. **FOREACH** ($t_1\in S$)
16. **FOREACH** ($t_2\in T$)
17. **IF** ($t_1\neq t_2$)
18. $Causal\_all:=Causal\_all\cup\{(t_1,t_2)\}$;
19. 返回 C-net:(*AllTasks*,*start*,*end*,*Causal_all*,*PreBinding*,*PostBinding*).

具体而言:

- 第 1 步~第 7 步并行处理原始事件日志中的各个案例,计算案例的执行任务集,并将案例归入相应的案例模型子日志.当采用多线程技术进行并行化时,由于不同线程处理的案例可能被归入同一个案例模型日志,这会引起共享对象的写冲突.因此,这一步中需要在各个线程之间针对案例模型对应的事件日志进行同步;
- 第 8 步~第 11 步并行地对各个案例模型事件日志执行算法 1 与算法 2 计算案例模型对应的任务依赖图与任务绑定信息.这一步中,各个线程之间不存在共享对象写冲突,由此可以进行充分的并行化;
- 第 12 步对各案例模型对应的任务绑定信息进行汇总,这一步无需并行化;
- 第 13 步调用算法 4 计算融合后 C-net 模型中的长距离依赖关系;
- 第 14 步~第 19 步针对挖掘到的长距离依赖关系在 C-net 中添加长距离依赖对应的流关系.

4.3 算法性能评价

若忽略线程同步消耗的时间,并假设各案例模型中的案例个数相等,则算法 5 的时间复杂度如下:

$$O(|L|/K\times TraceLength^3+|L|/K\times TraceLength^2\times|T|\times|Split|^2+|T|^3\times|Split|^2+|T|^3\times|Length2Loop|+$$
$$|T|^2\times|Split|^2\times|LongDistanceDepRel|^2).$$

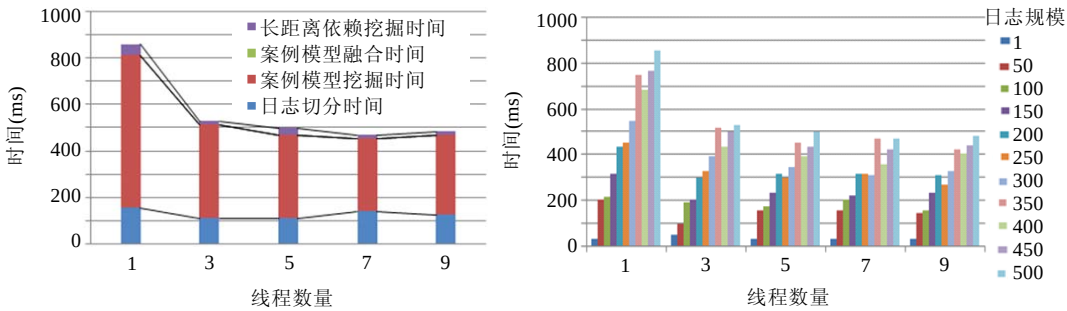
其中,$|L|$为日志中的案例个数,K 为处理机个数与案例模型数量的最小值,*TraceLength* 为案例运行轨迹的最大长

度,|*T*|为任务个数,|*Split*|为任务的最大分支个数,|*Length*2*Loop*|为 2-循环的个数,|*LongDistanceDepRel*|表示长距离依赖关系的数量.一般而言,任务数量、任务后继分支的个数、2-循环以及长距离依赖关系的数量会远小于日志数量.此时,可将这些因素对应的分量从并行化启发式流程挖掘算法的时间复杂度中忽略,从而可得时间复杂度的近似公式:

$$O(|L|/K\times TraceLength^3+|L|/K\times TraceLength^2\times|T|\times|Split|^2).$$

按照该近似公式,并行化启发式流程挖掘算法的时间复杂度可近似降低为串行算法复杂度的 1/*K*.

采用多线程技术对文中提出的并行化启发式挖掘算法进行了实现,并分别以表 1 对应的日志 *L* 以及 L^{50}(表示将 *L* 中的运行轨迹重复 50 次得到的日志),L^{100},L^{150},…,L^{500} 为输入,在处理器为 Intel Xeon E7-4830 八核 CPU、内存为 6G 的运行环境下,采用线程数量为 1,3,5,7,9 时,算法各主要步骤的耗用时间如图 9 所示.



(a) 日志 L^{500} 挖掘耗时统计图 (b) 不同线程数量与日志规模下算法总耗时统计图

Fig.9 The running time statistical chart of the proposed algorithm

图 9 算法运行时间统计图

由图 9(a)可见:多线程程序的耗时明显少于单线程程序的耗时,其中,案例模型挖掘耗用的时间减少尤其明显.这是因为线程在挖掘案例模型时无需任何同步,算法可完全并行.此外,随线程数量的增加,算法耗时先是明显减少,当线程数量达到 7 时耗时最少,之后反而有所上升.这是因为本文采用的日志中包含 7 个案例模型,采用 7 个线程既能发挥各个案例模型并行处理的优势,又不会因为线程过多增加线程操作的开销.图 9(b)给出了不同线程数量下日志规模分别为 *L* 的 1 倍、50 倍、100 倍直至 500 倍时并行化启发式流程挖掘算法的总耗时,由该图可见:随线程数量的增加,算法耗时随日志规模增长而增长的速度有所减缓.这也反映了算法并行化之后带来的好处.

在时间性能方面,由于文献[1,4,5]中未给出现有启发式流程挖掘算法的一些技术细节,我们无法准确还原已有算法,而且嵌入到 Prom6.1[10]中的现有启发式流程挖掘算法无法统计运行时间,因此,此处无法定量地将本文算法与已有算法进行直接对比.但是由于两者在任务依赖图挖掘步骤上的处理方式基本一致,在分支与合并结构以及长距离依赖关系的挖掘方面,本文方法在案例模型的基础上进行操作,相比以往的启发式流程挖掘算法会降低算法复杂度,本文方法新增的案例模型融合步骤仅需进行简单的袋集求和运算,时间复杂度较低,所以,图 9 中本算法单线程运行时的时间消耗应该低于已有启发式流程挖掘算法的耗时.即使用其近似已有启发式流程挖掘算法,其该耗时明显高于多线程并行化后的算法耗时.在忽略线程同步时间、假设各案例模型案例数量基本相同的理想情况,本文算法复杂度最好可将为原算法的 1/*K*.但实际情况中,一般会低于这个改进的量.例如图 9(b)中,串行程序(单线程)的挖掘时间为 875ms,采用 7 个线程时的挖掘时间降到了 374ms,时间缩短到原来的 43%.

最后,在噪音处理方面,除了可以通过设置各种因子阈值排除噪音外,还可在本文算法运行前统计各任务的执行次数以及各案例模型包含的案例数,并设置一定的阈值来排除不常执行的任务或者案例模型.

5 相关工作比较

文献[1]对现有流程发现算法进行了综述,指出了启发式流程挖掘系列算法在日志噪音和不完备日志处理方面的优势.本文方法是对启发式流程挖掘算法的完善和并行化,在日志噪音和不完备日志的处理方面,相比Alpha系列算法[1]、基于状态区域的两阶段流程挖掘算法[11]等流程挖掘方法有明显优势.

此外,在长距离依赖关系的挖掘方面,文献[8,9]曾对长距离依赖关系进行了分类,给出了几类长距离依赖的典型特征,并提出了从日志中挖掘包含长距离依赖关系的合理的WF-net的alpha++算法.文献[12]将整数线性规划方法运用于流程发现,提出了ILPMiner挖掘算法,其基本思想是:通过尽可能多的挖掘库所来限制所得Petri网模型的行为,使其尽量与日志中的案例行为一致.上述两个算法均能正确发现某些特定类型的长距离依赖关系,但是两者对日志中的噪音较为敏感,而对噪音日志的处理,是本文算法的主要优势之一.文献[13]将遗传算法应用到流程发现算法中,尝试用统一方法综合解决长距离依赖导致的非自由选择结构以及不可见任务和重名任务的挖掘问题,而且具有一定噪声处理能力,但是遗传式挖掘算法的时间效率较差.

图1中的流程模型是专门针对现有启发式流程挖掘算法在长距离依赖关系挖掘方面的局限而设计的,能很好地说明已有方法的不足与本文方法在这一方面的改进.而且,模型中包含了1-循环、2-循环以及并发、互斥、长距离依赖各种结构,模型具有很好的典型性和代表性.表1中的事件日志L包含了该模型所有不同的案例运行轨迹,以此为日志实例将本文挖掘算法与已有算法进行对比有一定的合理性.所以,作者以L为实例对嵌入到Prom6.1中的传统启发式流程挖掘算法、ILPMiner、遗传式挖掘算法以及alpha挖掘算法进行了测试.测试结果表明:上述各种已有算法均不能挖掘出原始模型中存在的长距离依赖关系;此外,ILPMiner算法无法正确得到原始模型中的循环结构,遗传式挖掘算法与alpha算法无法正确得到原始流程中存在的1-循环结构以及个别任务之间正确的因果依赖关系.此外,以文献[13]中针长距离依赖关系专门设计的几组第三方事件日志为输入,对本文挖掘算法进行测试发现,符合本文定义的两决策点间的长距离依赖关系均正确挖掘得到.

实际上,基于案例模型降低业务流程分析复杂度的思想在文献[14]中已经提出,其中主要是进行复杂业务流程的建模和验证;文献[15]按照这种思想提出了基于流程案例簇的任务关系挖掘算法,但要求日志中不含噪音,而且对日志完备性要求过强,本文通过借鉴启发式流程挖掘算法的思想在这一方面作了针对性的改进.

6 总 结

基于案例的执行任务集将流程模型划分为案例模型,并结合启发式流程发现算法给出了从事件日志出发并行挖掘各个案例模型所对应任务依赖图的方法;之后,对各个案例模型对应的任务依赖图进行融合得到完整流程模型对应的C-net模型;最后,将长距离依赖关系扩展为两个任务子集在决策点处的非局部依赖关系,给出了更为准确的对长距离依赖关系度量指标,结合实例说明了本文方法的可靠性并对算法运行时间进行了统计分析.后续工作将结合更多实例对本文方法进行改进和完善.

References:

[1] Van der Aalst WMP. Process Mining-Discovery, Conformance and Enhancement of Business Processes. Heidelberg: Springer-Verlag, 2011. 124−187.

[2] Turner CJ, Tiwari A, Olaiya R, Xu YC. Business process mining: From theory to practice. Business Process Management Journal, 2012,18(3):493−512. [doi: 10.1108/14637151211232669]

[3] Van der Aalst W, Adriansyah A, De Medeiros AKA, *et al*. Process mining manifesto. In: Daniel F, Barkaoui K, Dustdar S, eds. Proc. of the BPM 2011 Int'l Workshops. Heidelberg: Springer-Verlag, 2011. 169−194. [doi: 10.1007/978-3-642-28108-2_19]

[4] Weijters AJMM, van der Aalst WMP, De Medeiros AKA. Process Mining with the HeuristicsMiner Algorithm. BETA Working Paper Series 166. Eindhoven: BETA Publisher in Eindhoven University of Technology, 2006. 1−34.

[5] Weijters AJMM, Ribeiro JTS. Flexible Heuristics Miner (FHM). BETA Working Paper Series 334. Eindhoven: BETA Publisher in Eindhoven University of Technology, 2010. 1−29.

[6] Van der Aalst WMP, van Hee KM. Workflow Management: Models, Methods, and Systems. The MIT Press Cambridge, 2004. 31−68.

[7] Wen LJ, Wang JM, van der Aalst WMP, Huang BQ, Sun JG. Mining process models with prime invisible tasks. IEEE Data Knowledge Engineering, 2010,69(10):999−1021. [doi: 10.1016/j.datak.2010.06.001]

[8] Wen LJ, van der Aalst WMP, Wang JM, Sun JG. Mining process models with non-free-choice constructs. Data Mining and Knowledge Discovery, 2007,15(2):145−180. [doi: 10.1007/s10618-007-0065-y]

[9] Wen LJ, Wang JM, Sun JG. Detecting implicit dependencies between tasks from event logs. In: Zhou X, *et al*., eds. Proc. of the Frontiers of WWW Research and Development. Heidelberg: Springer-Verlag, 2006. 591−603. [doi: 10.1007/11610113_52]

[10] Van Dongen BF, de Medeiros AKA, Verbeek HMW, Weijters AJMM, van der Aalst WMP. The ProM framework: A new era in process mining tool support. In: Ciardo G, Darondeau P, eds. Proc. of the Applications and Theory of Petri Nets 2005. Heidelberg: Springer-Verlag, 2005. 444−454. [doi: 10.1007/11494744_25]

[11] Cortadella J, Kishinevsky M, Lavagno L, Yakovlev A. Deriving Petri nets from finite transition systems. IEEE Trans. on Computers, 1998,47(8):859−882. [doi: 10.1109/12.707587]

[12] Van der Werf JMEM, van Dongen BF, van Hee KM, Hurkens CAJ, Serebrenik A. Process discovery using integer linear programming. In: van Hee KM, Valk R, eds. Proc. of the Applications and Theory of Petri Nets 2008. Heidelberg: Springer-Verlag, 2008. 368−387. [doi: 10.1007/978-3-540-68746-7_24]

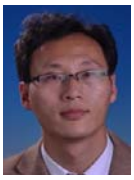
[13] De Medeiros AKA, Weijters AJMM, van der Aalst WMP. Genetic process mining: An experimental evaluation. Data Mining and Knowledge Discovery, 2007,14(2):245−304. [doi: 10.1007/s10618-006-0061-7]

[14] Lu FM, Zeng QT, Bao YX, Duan H. Hierarchy modeling and formal verification of emergency treatment processes. IEEE Trans. on Systems, Man and Cybernetics: Systems, 2014,44(2):220−234. [doi: 10.1109/TSMC.2013.2242465]

[15] Lu FM, Zeng QT, Bao YX, Duan H, Zhang H. Mining algorithm of task dependencies based on process case clusters. Computer Integrated Manufacturing Systems, 2013,19(8):1771−1783 (in Chinese with English abstract).

附中文参考文献:

[15] 鲁法明,曾庆田,包云霞,段华,张昊.基于流程案例簇的任务关系挖掘算法.计算机集成制造系统,2013,19(8):1771−1783.

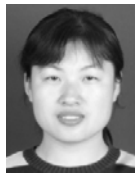


鲁法明(1981－),男,山东新泰人,博士,讲师,CCF 会员,主要研究领域为 Petri 网,流程挖掘.

曾庆田(1976－),男,博士,教授,博士生导师,CCF 高级会员,主要研究领域为流程挖掘,Petri 网.

段华(1976－),女,博士,副教授,主要研究领域为机器学习,数据挖掘.

程久军(1976－),男,博士,副教授,主要研究领域为移动计算.



包云霞(1979－),女,讲师,主要研究领域为统计学习.