软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2015,26(3):474−490 [doi: 10.13328/j.cnki.jos.004773]

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

基于 Petri 网的流程间元素映射方法*

曹 斌, 王佳星, 范 菁, 董天阳

(浙江工业大学 计算机科学与技术学院,浙江 杭州 310023)

通讯作者: 范菁, E-mail: fanjing@zjut.edu.cn

摘 要: 对流程进行比较,在业务流程管理中有着重要的应用价值,可用于流程版本控制、流程相似度计算、流程合并等应用场景当中.流程间的元素映射是流程比较的首要步骤.现有的流程比较方法仅考察了流程任务活动的映射,忽略了流程中其他元素的对应关系,进而无法保障流程比较结果的可靠性.为此,提出一种基于 Petri 网的流程间元素映射模型,侧重描述了库所映射时所需要满足的上下文环境;随后,在此基础上提出了基于上下文环境的流程间库所映射算法.对于任意给定的两个 Petri 网建模的流程模型,该算法通过双边和单边映射的策略,优先选择上下文环境相似程度最高的库所作为对应库所返回.大量基于真实数据集的实验展示了该方法在库所映射方面的有效性,也展示了其用于流程相似度计算方面的高效性.

关键词: 业务流程管理;流程比较;Petri 网建模;元素映射;上下文相似度

中图法分类号: TP18

中文引用格式: 曹斌,王佳星,范菁,董天阳.基于 Petri 网的流程间元素映射方法.软件学报,2015,26(3):474−490. http://www.jos.org.cn/1000-9825/4773.htm

英文引用格式: Cao B, Wang JX, Fan J, Dong TY. Mapping elements between process models based on Petri net. Ruan Jian Xue Bao/Journal of Software, 2015,26(3):474−490 (in Chinese). http://www.jos.org.cn/1000-9825/4773.htm

Mapping Elements Between Process Models Based on Petri Net

CAO Bin, WANG Jia-Xing, FAN Jing, DONG Tian-Yang

(School of Computer Science and Technology, Zhejiang University of Technology, Hangzhou 310023, China)

Abstract: Comparing processes has important application value in business process management, and it is common in scenarios of process version control, and process similarity calculation, process merging. Mapping elements between process models is the first step for process comparison. Current comparison solutions, however, only consider the mapping between tasks but ignore the correspondence of other elements in the process. Hence, the reliability of the results from the process comparison cannot be guaranteed. To address this issue, this paper first proposes a Petri net based model for element mapping between processes, and focuses on description of the context that is required for places mapping. Then, based on the place mapping context, a mapping algorithm is also presented. For any pair of processes modeled by Petri net, the algorithm adopts bilateral and unilateral mapping strategies where the pair of places that shares the most similar contexts is given high priority to be selected as the mapping result. The extensive experiments based on the real world data sets show the effectiveness of the proposed method as well as its efficiency in the scenario of process similarity search.

Key words: business process management; process comparison; Petri net modeling; element mapping; context similarity

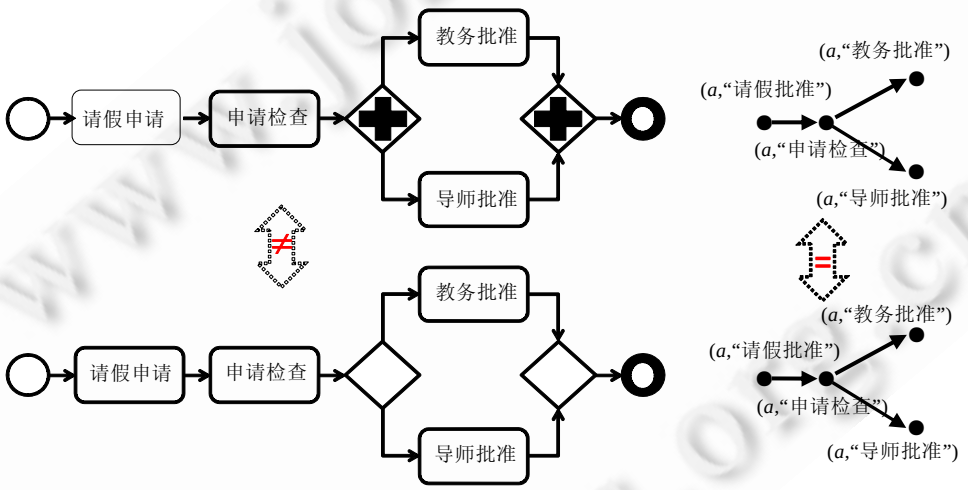
在业务流程管理(business process management,简称 BPM)过程中,很多应用场景需要对两个业务流程进行比较,如流程版本管理[1]中,管理人员要借助相关工具对同一流程的新旧版本进行比较,以确定流程在哪些环节发生了变化;流程相似度计算[2]时,需要对两个流程进行结构、活动、行为等方面的比较;流程一致性检测[3]中,需要将从流程执行日志中挖掘出的流程模型与原制定的模型进行比较,进而检查两者是否吻合;(4) 对来自不

* 基金项目: 国家自然科学基金(61173097); 浙江省重大科技专项(2013C01112); 杭州市重大科技创新专项(20132011A16)
收稿时间: 2014-07-08; 修改时间: 2014-09-30; 定稿时间: 2014-11-21

同企业组织的业务流程进行合并[4]时,需要通过流程比较找出业务的重叠部分.

考虑到业务流程图模型自身在路由结构、执行语义、资源配置等方面的复杂性,对它们进行比较并非易事.一般来说,流程比较的首要步骤是建立两个流程模型中各元素间的映射关系,即从流程 *B* 中找到流程 *A* 中相对应的任务、资源等.基于找到的映射关系,业务人员才能进一步根据具体应用目的对流程进行比较.但现有的流程比较方法均侧重于特定应用场景问题,而忽略或简化了流程间各元素的映射过程,仅对流程模型中表示实际业务的任务元素进行处理,因此,这些方法尚未能在实际中得到有效地应用.

例如,在基于图匹配的流程相似度计算[2]应用场景中,现有方法仅是将流程模型中不同的节点类型以一种统一的方式进行文本标注,并抽象转化成仅包含有活动元素的流程图模型.在图 1 中,左边是以 BPMN[5]方式建模的两个不同的原始业务流程图模型,除开始、结束节点外,它们包含有 BPMN 的两种基本元素,即活动(以矩形表示)、网关(以菱形表示).两个流程除了教务批准与导师批准两个活动元素的路由方式不同外(上面流程是并行,下面的是选择),其余流程元素均相同.右边的流程图则是去掉网关元素后的抽象效果图,最终对流程比较进行相似度计算时以右图为输入.然而从图中可以明显看出,右边两个抽象后的流程图完全相同.这是因为右图中网关元素的缺失使得该流程图模型无法正确表达活动教务批准与导师批准是并行路由还是选择路由,无法反映左边原有流程中的真实路由情况,因此,最终得到的流程比较结果的正确性也无法得到保障.



(a) 业务流程图模型 (b) 流程图(无网关)

Fig.1 The workflow model based on BPMN and its abstraction

图 1 BPMN 流程图模型及其抽象处理

本文针对上述流程比较过程中对流程模型内的元素映射不完整的问题展开研究,提出了基于 Petri 网[6]的流程间元素映射方法.选择 Petri 网的直接原因是,其模型中包含的元素类型只有变迁(transition)与库所(place)两种,而其他建模方法,如 BPMN,EPC,均含有更多的(>2)元素类型,使得需要考察和分析的映射情况更多.因此为了简化问题,我们选择 Petri 网进行相关映射方法的设计.

本文所提方法从流程图结构出发,主要面向基于 Petri 网建模的业务流程图模型,对流程模型内的元素,即变迁(transition)与库所(place),分别进行映射,以找到它们在不同流程中的对等关系.对于变迁间的映射,与现有常用方法类似,我们考察变迁的标识是否相等或计算其文本相似度[7].对于库所间的映射,由于其标识与变迁标识作用不同,在实际业务中并不代表具体的业务活动,因此不能简单地通过判断标识进行映射.事实上,既然库所表示变迁的条件和状态,那么库所与其相邻的变迁有着从属的关系:若两个库所其相邻的变迁完全相同,我们则认为它们是相同的.因此基于该假设,我们采用的库所匹配策略是:先记录待比较流程中各库所的相邻变迁,然后对流程 *A* 中的各库所与流程 *B* 中的各库所进行匹配,计算它们基于相邻变迁交集的上下文环境相似度并排

序,最后从所有库所对中按相仿度程度大小进行映射结果的选择.

本文主要贡献可归纳为如下 3 点:

- 提出了基于 Petri 网的流程间元素映射模型,并针对库所元素的映射情况,设计了指导库所间映射的模型,即基于相邻变迁交集的上下文环境,该模型弥补了现有流程比较方法无法对流程进行完整有效映射的问题;
- 在此基础上,提出了一种针对 Petri 网流程模型间库所元素的映射算法,该算法基于上下文环境相似程度进行映射选择,可以在两个流程间找出表示条件和状态的元素的对应关系;
- 基于真实数据集,并结合流程相似度计算场景,通过对所提算法的大量实验评估,证明其映射匹配效果和效率方面均能够适用于实际的业务流程管理当中.

本文第 1 节介绍基于 Petri 网的流程建模相关基础知识.第 2 节给出流程间元素映射模型以及库所映射时需要考虑的上下文环境.第 3 节提出基于上下文环境的库所元素映射算法,并分阶段对算法实现进行描述.第 4 节基于真实流程数据集进行大量的实验评估与分析.第 5 节介绍相关工作.最后总结并指出未来研究方向.

1 基于 Petri 网的流程建模

Petri 网图模型的两个基本元素库所和变迁由有向弧(directed arc)相连接,进而构成了 Petri 网内元素间的流关系.有向弧可以从变迁到库所或从库所到变迁,但相同两个节点之间不允许相连接.对于一个变迁,通过有向弧流入该变迁的库所集合称为输入库所(input place),而流出该变迁的库所集合称为输出库所(output place).在用 Petri 网对系统进行建模时,通常把库所看做系统中的资源,如状态、条件、媒介等,而变迁则看做是系统中的变化,如事件、操作、传输等.库所可以容纳一定数量的令牌(token),用来表示所代表资源的数量.若一个变迁的所有输入库所中存在足够数量的令牌使其使能,则该变迁可以被触发(firing);当变迁被触发后,它将消耗掉必要的令牌并在输出库所中生成新的令牌.上述元素的常用图形表示如图 2 所示.

Fig.2 Graph representation for basic Petri net elements

图 2 Petri 网元素图形表示

为达到特定的目的,流程模型通常被用来描述有哪些任务(task)需要被执行以及以什么次序执行.任务是流程模型的基本单元,它是不可分割的,必须完整地执行.任务间连接的组合构成了整个流程模型的控制结构,该结构体现了流程中任务路由情况,具体可分为顺序路由、并行路由、互斥选择路由以及循环路由[8].显然,Petri 网能够很好地表示上述路由方式[9].一般来说,用 Petri 网定义或表示一个流程时,任务由变迁表示,条件由库所表示,任务间的因果依赖关系可由库所和弧来表示,且该网只有一个入口(没有输入弧的库所)和一个出口(没有输出弧的库所).下面分别对业务流程管理过程中最为常用的顺序、并行以及互斥选择这 3 种路由方式进行 Petri 网建模说明:

- 顺序路由:如图 3(a)所示,通过在两个变迁间添加一个库所进行链接的方式来表示;
- 并行路由:如图 3(b)所示,AND-split 与 AND-join 结构分别对应于一个变迁有多个输出库所和一个变迁有多个输入库所,即通过使用 AND-split 变迁使得多个任务能被同时处理,而 AND-join 变迁只有当其之前的任务都完成才能被触发;
- 互斥选择路由:如图 3(c)所示,XOR-split 与 XOR-join 分别对应于一个库所有多个出线弧和有多个入线弧(连接变迁).库所中的令牌用于选择变迁.

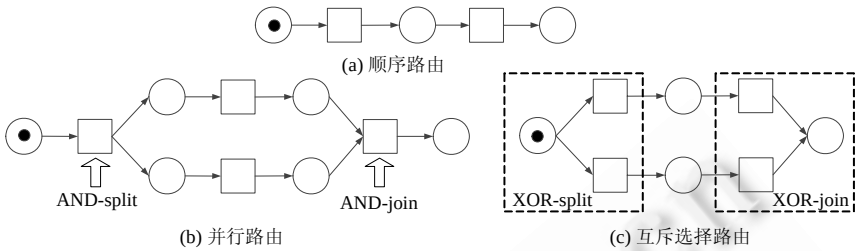


Fig.3 Basic workflow blocks modelled by Petri net

图 3 Petri 网建模流程基本结构

基于这 3 种基本结构,可以构造更加复杂的路由结构.如图 4 所示为一个多重选择(OR-split)、多重合并(OR-join)路由.构造多重选择路由需要为每条分支 $B_1,\ldots,B_n$ 都构造一条与其入口条件相反的任务$\sim B_1,\ldots,\sim B_n$,然后分别将 $B_1,\sim B_1,\ldots,B_n,\sim B_n$ 这 n 对任务通过 XOR-split 连接,最后再将这 n 个 XOR-split 节点通过 AND-split 连接.多重合并路由将所有需要合并的分支通过 XOR-join 连接,最后将所有的 XOR-join 用 AND-join 连接.

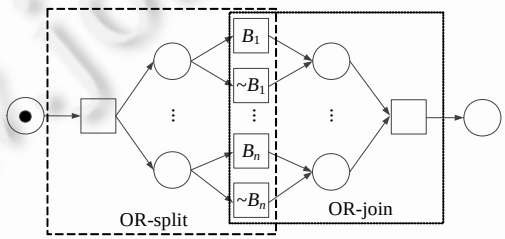


Fig.4 OR-split and OR-join modelled by Petri net

图 4 Petri 网建模多重选择与多重合并路由

2 基于 Petri 网的流程间元素映射模型

变迁元素代表了实际业务流程中的任务,库所代表了相关任务的条件或状态.对不同的 Petri 网流程图模型进行元素映射,主要是对变迁与库所节点展开.注意:由于令牌在不同时刻在各库所中的分布表示了流程模型具体执行时的实例情况,而本文则关注于流程模型定义本身,因此,对不同流程实例及其相应的运行情况或令牌分布不在本文讨论范围内.下面分别从变迁映射和库所映射两方面展开具体介绍.

2.1 变迁映射

在业务流程管理过程中,企业组织内的各任务活动并不专属于某个特定的业务流程,而是根据不同的业务需求,这些任务活动将可能在不同的流程中被重复使用,通过其他任务活动的配合,进而帮助相关流程完成特定的业务目的.因此,同一任务可以在多个不同的流程模型中出现.人们一般通过任务的标识,即可从不同流程中将相同的任务进行识别和对应.由于 Petri 网在业务流程的建模过程当中,变迁用来表示流程中的任务节点,因此变迁的标识一般可直接由任务的标识代替,进而,我们基于该标识对不同 Petri 网表示的流程模型的变迁进行映射.标识一般是一个字符串,因此,判断两个标识是否表示同一个变迁任务,可结合上下文执行语义或行为相似度[7],并通过判断两个字符串是否相等进行确定.在实际场景当中,可能不同的字符串也会代表相同的任务,在这种情况下,一些传统的文本和句法相似度计算方法可以被借鉴,如字符串编辑距离(string edit distance)[7]和基于 Wordnet 认知语言学词典相似度计算方法[10].基于上述文本相似度,现已有较成熟的方法[4]可以计算出两个任意变迁任务是否相似,如利用基于模式匹配技术[11,12]的计算方法,将两个变迁标识进行句法处理(去除标点、或英语词语后缀,如-ing 等),然后计算这两个标识内每一对单词的文本和句法相似度,接着选取两种相似度中的最大值,最终,两个标识的相似程度即是它们的每一对单词相似度之和平均值.

2.2 库所映射

由于库所在 Petri 网建模的流程模型中表示与其相连接变迁(即任务)的状态,与变迁标识作用不同,库所在流程模型中的标识仅用于区别其他库所元素,在实际业务背景中没有直接的具体意义.进而,我们不能像变迁映射一样简单地通过判断库所的标识进行映射关系的寻找.因此,如何对库所元素进行流程间的映射成为了本文的关键问题.接下来,我们通过一个样例进行说明.

如图 5 所示,左边有两个基于 Petri 网建模的不同流程:Process 1 与 Process 2.可以看出,两个流程包含有相同的变迁(任务),即{*A*,*B*,*C*,*D*,*E*}以及有相同数量的库所节点.两个流程的差别在于变迁 *C* 的位置不同.在 Process 1 中,变迁 *C* 与变迁 *B* 为选择关系,两者的输入库所均为 P_1,输出库所均为 P_2;而在 Process 2 中,变迁 *C* 与变迁 *D* 互为选择关系,它们共享 P_3' 为输入库所, P_4' 为输出库所.除去开始与结束节点外,Process 1 与 Process 2 中的库所元素间可能的映射如图 5 中间部分所示,共有 $A_4^4=24$ 种映射可能.以 P_1 为例,直观上,由于 P_1 与 P_1' 在各自的流程中两端变迁交集为 *A* 与 *B*,即,它们处在相同的上下文环境当中,因此我们认为,P_1 与 P_1' 在某种程度上可以作为一组映射关系.事实上,由于库所与变迁通过弧进行连接,它们之间存在一种从属关系,而这种从属关系造成的上下文环境恰是我们判断 P_1 与 P_1' 两者可能有对应关系的依据.基于这样的上下文关系,我们可以得到两个流程模型间各库所的多种映射可能.图 5 右边部分展示了左边两个流程虚线部分库所的 4 种可能映射.下面,我们对图 5 样例中潜在的库所映射机制进行一般化的定义.

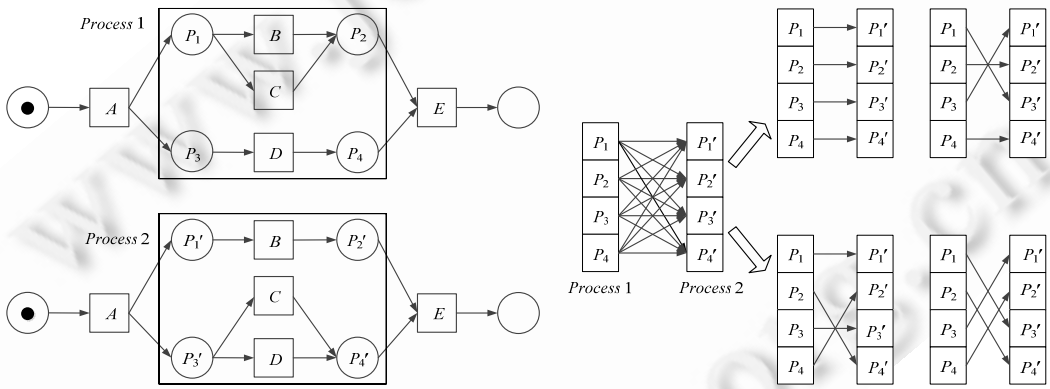


Fig.5 Two processes modelled by Petri net and their places mappings

图 5 两个 Petri 网流程样例及其库所映射

定义 1(左、右变迁集合). 在一个 Petri 网建模的流程模型中,对任意一个库所 P_i,将 P_i 作为输出库所的变迁集合称为 P_i 的左变迁集合,表示为 $LT(P_i)$;将 P_i 作为输入库所的变迁集合称为右变迁集合,表示为 $RT(P_i)$.

上述定义是针对流程内单个具体库所元素而言的.事实上,它描述了该库所在 Petri 网流程中的上下文环境,即,与该库所有着从属依赖关系的变迁.我们可以认为:该库所从属于右变迁集合(表示右变迁任务的开始),并依赖于其左变迁集合(表示左变迁任务的结束).

定义 2(库所上下文环境). 在一个 Petri 网建模的流程模型中,对任意一个库所 P_i,其上下文环境由左变迁集合 $LT(P_i)$与右变迁集合 $RT(P_i)$组成.

根据定义 2,我们给出本文在对库所进行映射时基于的假设:上下文环境一样的库所是相同的.但是,两个 Petri 网建模的流程模型比较时,很多情况下并不能找到两个上下文环境完全一样的库所,即不存在完全相等的库所映射.但存在大量上下文环境部分相同的库所,这时,我们就想确定哪些库所间的映射可能性最大,以及如何把那些最为可能映射的库所对找出来.

定义 3(左、右变迁交集). 给定两个 Petri 网建模的流程模型 Process 1 与 Process 2,考察 Process 1 中的库所 P_i 与 Process 2 中的库所 P_i' ,$LT(P_i)$与 $LT(P_i')$ 的交集称为 P_i 与 P_i' 的左变迁交集,表示为 $Intersect(LT(P_i)$,

$LT(P_i')$);而 $RT(P_i)$与 $RT(P_i')$ 的交集称为 P_i 与 P_i' 的右变迁交集 $Intersect(RT(P_i), RT(P_i'))$.

定义 4(笛卡尔积). 给定两个 Petri 网建模的流程模型 Process 1 与 Process 2,Process 1 中的库所 P_i 与 Process 2 中的库所 P_i' 的笛卡尔积:

$$DP(P_i,P_i')=\{\langle T_1,T_2\rangle \mid T_1\in(LT(P_i)\cap LT(P_i')) \wedge T_2\in(RT(P_i)\cap RT(P_i')), P_i\in \text{Process 1}, P_i'\in \text{Process 2}\}.$$

从定义 4 可以看出,Process 1 中的库所 P_i 与 Process 2 中的库所 P_i' 的笛卡尔积就是 P_i, P_i' 的左变迁交集中一个成员与 P_i, P_i' 的右变迁交集中一个成员的所有可能的有序对.

基于上述定义我们认为,两个库所上下文环境的相似程度大小与它们的左、右变迁交集正相关,即共享的变迁越多,则上下文环境越相似,最终成为对应关系的可能性越大.基于此,对于两个不同的流程,本文对它们的库所间的映射优先级策略如下:

1) 双边映射(two-side mapping):对于左变迁交集与右变迁交集均不为空集的库所对,按它们的笛卡尔积(定义 4)中的元素个数从大到小选择;
2) 单边映射(one-side mapping):对于左变迁交集与右变迁交集有一个不为空集的库所对,按不为空的变迁交集的个数从大到小选择;
3) 无映射:左变迁交集与右变迁交集均为空集.

可以看出,上述映射优先级是根据库所的上下文环境的完整程度来划分的,即双边映射的库所在其两边均共享有相同的变迁,单边映射仅在一边共享变迁.即,均在左变迁存在交集或均在右变迁存在交集.最后,若左、右变迁均无共享的变迁,则它们不能构成映射.在这种情况下,流程比较时一般将这样的库所看作是在另一个流程中增加或减少的节点元素.我们的最终目的是发现双边映射和单边映射的库所(剩余的则为无映射的库所).对于双边映射,左变迁交集与右变迁交集的笛卡尔积乘积可以用来表示上下文环境的相似程度,即对于该库所可能存在的实例路由组合情况;而单边映射情况下,上下文环境的相似程度取决于不为空集的单边变迁交集.我们在下一节将给出基于上下文环境的库所映射算法具体实现过程.

3 基于上下文环境的库所映射算法实现

本节基于第 2 节中描述的上下文环境相关定义以及库所映射优先级策略,提出了具体的库所映射算法.算法的输入为两个基于 Petri 网建模的流程模型,输出为经映射后两个流程的库所集合.

算法基于一个类似哈希表的流程间库所映射表,共分为 3 个阶段:

(1) 首先,在库所映射表结构中,初始化流程间所有库所映射;
(2) 接着,对初始化后的表结构执行双边映射过程;
(3) 最后,执行单边映射过程,匹配仅单边变迁交集不为空的库所.

对后两个阶段的结果进行并集操作,即可得到两个流程间最终的对应库所对.

我们还将介绍用到的主要数据结构,然后对算法各阶段展开详细介绍,并结合图 5 所示流程样例辅助对相关映射步骤的说明.

3.1 数据结构

算法在整个映射匹配过程当中要维护一张全局的表结构 *Place_Mapping_Table*,其原理与经典的哈希表数据结构类似,其定义如下:

定义 5(库所映射表). 给定两个 Petri 网建模的流程模型 Process 1 与 Process 2,将 Process 1 中的每个库所 $P_i\in(P_1,P_2,...,P_m)$都与 Process 2 中的所有库所 $P_1,P_2,...,P_n$ 进行匹配,并记录 P_i 与 Process 2 中的所有库所的左、右变迁交集,生成一张 $m\times n$ 的库所映射表.

如图 6 所示,左边的竖形列表是流程 Process 1 的所有库所(由 P_i 表示),右边每一行链表包含了流程 Process 2 的所有库所,由 P_j' 表示.左边每一个库所均会对应右边的一个完整链表.右边每一链表项中的内容 P_j',[{},{}]由两部分构成: P_j' 是表示与左边流程 Process 1 中同在一行的库所进行匹配操作;[{},{}]表示两

个当前匹配库所的左(第 1 个{})、右(第 2 个{})变迁交集.

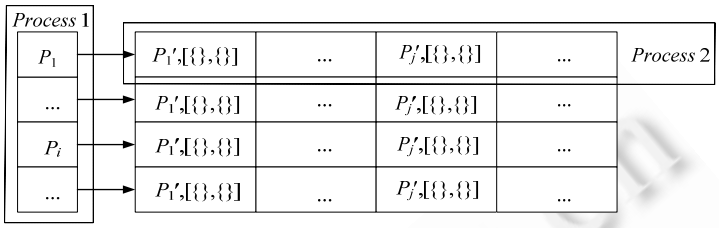


Fig.6 The example of place mapping table

图 6 库所映射表:*Place_Mapping_Table*

3.2 初始化映射

初始化映射的主要目的是初始化库所映射表 *Place_Mapping_Table*,进而建立两个待比较 Petri 网流程的所有库所间的上下文环境相似程度全局表,后面的阶段基于该度量可实施相关映射策略.

图 7 展示了初始化映射相关步骤的伪代码,其基于给定的两个 Petri 网流程(输入),生成库所映射表作为输出.首先遍历第 1 个流程 Process_1 中的所有库所,对于其中的每个库所,我们考察其与第 2 个流程 Process_2 中每个库所的上下文相似情况,该步骤通过一个嵌套的遍历完成(第 2 行~第 5 行).

Input: Two Petri-net based process models: Process_1, Process_2;
Output: The Initialized place mapping table: *placeMap*.
for each place P_i in Process_1
  for each place P_j' in Process_2
    $Intersect(LT(P_i),LT(P_j'))$ ← get intersection of $LT(P_i)$ and $LT(P_j')$ //定义 1、定义 2
    $Intersect(RT(P_i),RT(P_j'))$ ← get intersection of $RT(P_i)$ and $RT(P_j')$ //定义1、定义2
    Place_List←Add " P_j',[$Intersect(LT(P_i),LT(P_j'))$, $Intersect(RT(P_i),RT(P_j'))$],"
  placeMap←register P_i and *Place_List* to *Place_Mapping_Table*.

Fig.7 Initialization of the place mapping table

图 7 初始化 *Place_Mapping_Table*

在内层嵌套处理过程中,对于每个 Process_2 中的库所,我们计算内外两个库所的左、右变迁交集;然后,将其与该库所作为一个整体对象添加到一个链表中;最后,再将该链表与外层的库所一起初始化到库所映射表当中.当 Process_1 中的所有库所遍历完成时,库所映射表初始工作完成.

图 8 是对图 5 中样例的初始化映射结果,两个流程均含有 4 个库所,对于左边流程 Process 1 的每个库所,均有一个大小为 4 的链表在右边与之对应.其中,[{},{}]表示与相应的库所没有共享相同的变迁,因此我们认为,这样的库所对的上下文相似度为 0,最后不能将它们作为映射结果输出.

P_1	P_1',[{A},{B}]	P_2',[{},{}]	P_3',[{A},{C}]	P_4',[{},{}]
P_2	P_1',[{},{}]	P_2',[{B},{E}]	P_3',[{},{}]	P_4',[{C},{E}]
P_3	P_1',[{A},{}]	P_2',[{},{}]	P_3',[{A},{D}]	P_4',[{},{}]
P_4	P_1',[{},{}]	P_2',[{},{E}]	P_3',[{},{}]	P_4',[{D},{E}]

Fig.8 The initialized mapping based on Fig.5

图 8 基于图 5 样例的初始化映射

3.3 双边映射

双边映射的目的是:对那些左、右变迁交集均不为空的库所进行上下文环境相似度的排序,在这些库所中寻找上下文环境相似度最大的库所对作为映射结果返回.在具体实现中,我们对上下文环境相似度的排序是基于左变迁交集和右变迁交集的笛卡尔积,笛卡尔积中元素个数越多,说明这两个库所元素的共享变迁越多,其上下文环境越相似,它们被成功映射的可能性则越大.

双边映射过程的算法伪代码如图 9 所示,其输入是上阶段初始化好的全局库所映射表,输出则是左、右变迁交集均不为空的库所映射结果.

Input: The Initialized place mapping table: *placeMap*;
Output: Matched place pair after two-side mapping: *Matched_twoSide*.
Sort each list of Process_2 in *placeMap* by the cross join value of $Intersect(LT(P_i),LT(P'_j))$ and $Intersect(RT(P_i),RT(P'_j))$

```
for i=0 to Process_2.size
    maxValue←0
    for each place P_i in placeMap
        Place_List←get the place list corresponding to P_i
        First_Place ←get the first place of Place_List
        crossJoinValue←get the cross join value of First_Place and P_i
        if crossJoinValue>maxValue then
            maxValue←crossJoinValue
            Clear Matched_twoSide
            Matched_twoSide←add First_Place
    Delete Matched_twoSide from placeMap
```

Fig.9 The pseudo code for bilateral mapping

图 9 双边映射伪代码

在开始映射匹配前,先对全局库所映射表 *Place_Mapping_Table* 进行排序操作(第 1 行).具体方式为:对每一行流程 Process_2 列表中的元素,按照左、右变迁交集的笛卡尔积中的元素个数从大到小排序.这时,全局库所映射表 *Place_Mapping_Table* 的右边部分被更新为排序后的列表,且每个列表的第 1 个库所及相关的左、右变迁交集内容,表示其与左边对应库所能够进行映射的上下文环境最大相似度.至此,虽然在全局库所映射表 *Place_Mapping_Table* 中,我们横向上保证了特定两个库所映射的最大上下文环境相似度,但纵向上却不一定.例如,流程 A 中的库所 P_1 与流程 B 中的库所 P'_1 在同一行上具有最大的上下文环境相似度为 6,但此时,流程 A 中的库所 P_2 与 P'_1 在同一行上也是最大的上下文环境相似度,且为 9,比 P_1 与 P'_1 大.这种情况下,我们优先选择纵向上下文环境相似度最大的库所对为最终映射结果,即 P_2 与 P'_1 .上述纵向上的最大相似度检查对应图 9 中的第 3 行~第 11 行.最后,每当找到一对双边映射的库所,需要将它们从全局库所映射表 *Place_Mapping_Table* 中删除,即从两个流程的库所集合中同时删除(第 12 行).

图 10 展示了对图 5 样例的双边映射结果,其中,灰色标记的两对库所为本阶段的输出结果.

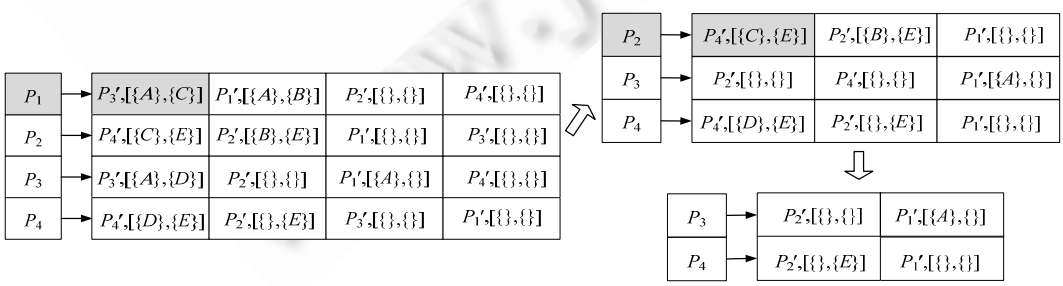


Fig.10 The two-side mapping based on the example in Fig.5

图 10 基于图 5 样例的双边映射

映射表 *Place_Mapping_Table* 一开始根据库所间左、右变迁交集的笛卡尔积的大小对每行列表进行了排

序,然后再纵向上比较每行的笛卡尔大小(虚线框表示).在本样例中,由于 4 个首位的库所与左边流程中的每个库所的上下文相似度大小相同(笛卡尔积均为 1),因此按照列表由上到下的顺序选择 P_1 与 P_3' 作为一对映射结果.接着,将它们从 *Place_Mapping_Table* 中删除.这时,*Place_Mapping_Table* 表变为图 10 右上角的结果,其尺寸被缩小.然后重复上述过程,我们又找到了 P_2 与 P_4' 这样一对映射结果.最终,*Place_Mapping_Table* 表变为右下角的内容.整个过程中,*Place_Mapping_Table* 的尺寸从 4×4 减小到 3×3,最后又减小到 2×2.因此在双边映射过程中,每找到一对映射结果,算法的搜索空间将以阶乘的形式降低,这样的方式在一定程度上保证了算法的高效性.

3.4 单边映射

单边映射阶段主要考虑左变迁交集与右变迁交集有一个不为空集的库所对,因此,该阶段的上下文环境相似度不能按照笛卡尔积来描述,因为它们均为 0.考虑到这些库所对存在一个不为空集的变迁交集集合,单边共享相同的变迁,因此具有一定的相似度.所以我们在进行映射时,将它们不为空的变迁交集个数大小作为上下文环境相似度的衡量,即按照不为空的变迁交集的个数从大到小进行选择.在算法实现上,该阶段与双边映射唯一的不同仅在于对全局库所映射表 *Place_Mapping_Table* 的排序依据不同,在此不再赘述.下面仅继续图 10 双边映射结果后的例子进行说明.

图 11 是图 10 结果的后续单边映射,可以看出,虚线框部分已与图 10 阶段结束不同.这是因为在单边映射开始时,我们先对流程 Process 2 中的 P_1',P_2' 根据它们各自与 P_3 或 P_4 的变迁交集大小进行了排序.例如,在 P_3 对应的右链表中,由于 P_1' 与 P_3 在左变迁上共享 A 变迁,而 P_2' 与 P_3 没有任何共享的变迁,因此, P_1' 被排在了 P_2' 前面.然后经过纵向比较,确定 P_3 与 P_1' 成为一对单边映射返回.剩余的 P_4 与 P_2' 为最后一组结果返回.

综上,通过对变迁和库所元素分别映射,我们为两个流程的比较建立起了元素对等关系.对于不同的应用场景,需要以不同的方式来利用这些对等结果.例如在相似性度量用途中,一个可能的使用方式是:不需要去进行流程图路由节点的抽象处理,可直接将路由节点作为传统经典图结构中的普通节点(即不区分变迁与库所类型节点,只有一种节点类型),但给予不同的标识,然后利用标识间的映射关系进行传统图的相似度计算方法即可.

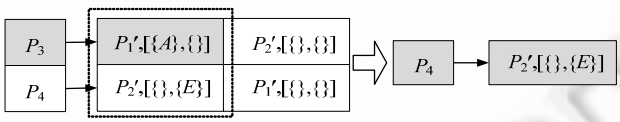


Fig.11 The one-side mapping based on the example in Fig.5

图 11 基于图 5 样例的单边映射

4 实验评估

由于本文所提映射方法是为流程比较服务的通用解决方案,对其映射结果优劣程度的考察需要结合具体的场景进行分析.由于目前尚缺乏统一的优劣评判标准,本文实验对基于真实数据的流程库进行人工去分支、去片段、去路由的改造,并人工找出两个流程的最佳映射的库所对,而变迁映射的实现是基于变迁的标识是否相同.将算法找出的映射库所对与人工找出的映射库所对进行比较,考察对流程进行去分支、去片段、去路由的改造后,本文映射算法的正确率变化情况.

此外,本节还将通过实验考察库所、变迁、边的变化对映射效率的影响.在效率评估实验中,我们展开流程间一对一的库所映射过程,分阶段考察基于上下文环境库所映射算法的性能;随后,我们将变迁映射加入,选取实际流程比较中对性能要求较高的流程相似度计算场景,将单个流程与流程库中的所有流程进行一对多的元素映射,看其性能是否满足实际应用的需要.

本文所有实验基于如下环境:处理器 Intel(R) Core(TM) i3-3240,3.40GHz,2.00GB 内存,Windows7 64 位操作系统,JDK 1.6.

4.1 数据介绍

实验数据来自于 IBM 公开的一组真实数据[13],包括 5 个流程库,总共 3 000 多个流程.这些流程大多是在对客户项目进行建模后获得的,其涉及保险、银行、自动化、通信、建筑、供应链、医疗保健和客户关系管理等多个领域.在这些流程中隐去了原来的详细信息,如重命名流程库名为 A,B_1,B_2,B_3,C,变迁名由 $T_1,T_2,T_3,...$来代替,库所名简化为 $P_1,P_2,P_3,...$匿名后的流程忽略了原先的实际含义而只考虑其结构信息.5 个流程库的复杂度都不相同,表 1 展示了它们各自的库所、变迁、边数的最大值、最小值、平均值信息,其中,B_1,B_2,B_3 代表的是相同领域中的同一个系列的流程,因此会有部分重叠的流程.对其中的一个库增加更多的流程模型并且细化其所有的流程,可以得到另一个流程库.本文实验中所用到的 Petri 网数据,均通过 UML2oWFN 工具[10]将 WebSphere 格式的流程定义模板转换而来.

Table 1 The dataset

表 1 数据集

	A	B_1	B_2	B_3	C
min/max/average place	9/174/50.93	5/217/46.91	5/271/50.03	7/276/53.23	2/585/14.42
min/max/average task	5/107/32.36	5/155/34.08	5/150/33.48	5/188/37.7	2/8431/20.53
min/max/average edge	15/362/102.21	9/452/94.73	9/558/99.81	11/585/107.48	3/61978/75.93
Size	284	288	363	421	1702

4.2 准确性评估

在准确性评估实验中,从第 4.1 节中的 5 个流程库中选取一定数量的流程,对其进行去分支、去片段、去路由的改造,并人工选出改造前和改造后两个流程的最佳库所映射对,考察用本文算法对改造前后的流程进行映射的准确性.

4.2.1 去分支

从第 4.1 节中的 5 个流程库中选取 100 个不同复杂度的流程,对其进行去分支的改造.如图 12(case_1-1)所示,流程 Process 1 包含一个选择结构,我们对其人工去掉其中的一个选择分支,即虚线框中的库所 P_2 和变迁 B,C,改造后变为流程 Process 2,则认为两个流程的最佳库所映射对为 P_1 与 P_1',P_3 与 P_3',P_4 与 P_4'.同理,如图 13(case 1-2)所示,我们对流程 Process 1 人工去掉其中的一个并行分支得到流程 Process 2,此时我们认为,其最佳的库所映射对为 P_3 与 P_3',P_4 与 P_4',P_5 与 P_5'.

用我们的算法对 100 组(改造前与改造后)流程进行映射后,得到的映射正确率为 100%.图 12 去掉一个选择分支后,使代表 XOR-Split 和 XOR-Join 的库所减少一个变迁,而去掉的分支中的库所连同左、右变迁也去掉了,因此,这些去掉的库所得不到映射,剩下的标识相同的库所的上下文相似度仍是最高的.图 13 去掉一个并行分支后,剩下的库所的左、右数量都没有改变,即,其上下文相似度与改造前相同,因此,剩下的标识相同的库所对映射起来.

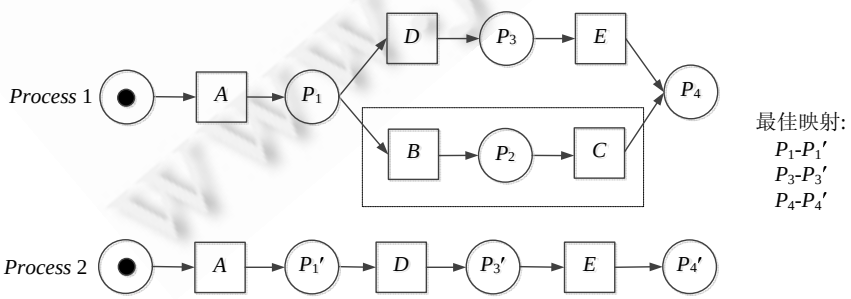


Fig.12 Case 1-1: removing the branch of XOR

图 12 Case 1-1:去选择分支

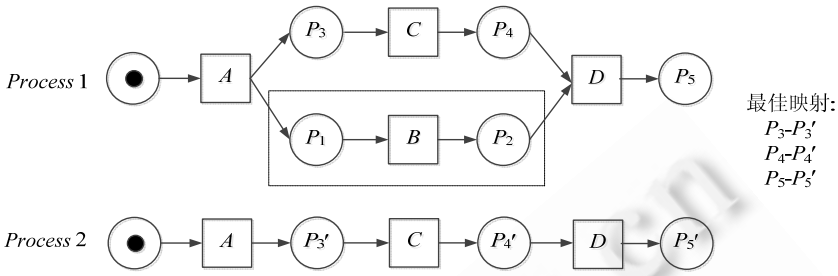


Fig.13 Case 1-2: Removing the branch of AND

图 13 Case 1-2:去并行分支

4.2.2 去片段

我们选取库所数量为 80,100,120 的流程,对其进行去片段(相连的一个库所和一个变迁)的改造.如图 14 (case 2)所示,流程 Process 1 是一个顺序结构,我们去掉其中虚线框中所示的一个片段即变迁 B 和库所 P_2,得到流程 Process 2.此时我们认为,最佳的库所映射对为 P_1 与 P_1' ,P_3 与 P_3' .

我们将库所数量为 80,100,120 的流程人工减少 5,10,15,20,25,30 个片段,然后用本文所提算法对改造前后的两个流程进行映射.如库所数量为 80 的流程,对其进行去片段的改造,得到库所数量为 75,70,65,60,55,50 的流程,分别用这 6 个改造后的流程与改造前的流程进行映射.在图 15 中我们可以看到:随着减少的库所数量的增加,映射算法在 3 个流程样例上的执行准确率呈线性下降.原因是,流程中去掉片段后会使得去掉片段的前面那个库所存在多种映射的可能性.如图 14 中,流程 Process 2 中的库所 P_1' 与流程 Process 1 中库所 P_1,P_2 都能进行映射,但在我们的算法中,我们没有采取策略对这种情况进行处理,因此算法为流程 Process 2 中的库所 P_1' 选择 Process 1 中库所 P_1 还是 P_2 来进行映射,我们无从得知.随着去掉的片段数增加,算法找出的映射库所对与我们人工选择的映射库所对相同的数目越来越少,即,正确率越来越低.

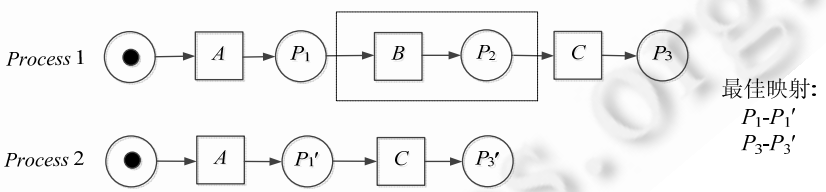


Fig.14 Case 2: Removing the sequence segment

图 14 Case 2:去顺序片段

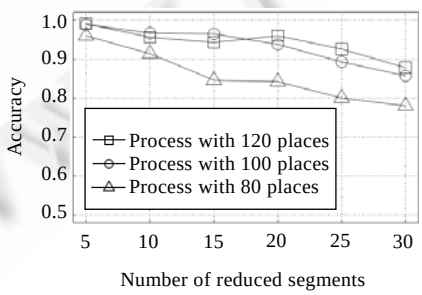


Fig.15 The accuracy after removing the sequence segment

图 15 Case 2 去顺序片段准确度结果

4.2.3 去路由

我们选取库所数量为 40 和 100 的流程,对其进行去路由的改造.图 16(case 3-1)表示去 XOR-Join 路由,我们去掉虚线框中的库所 P_3,P_4 和变迁 F,G,并且让变迁 E 与库所 P_2 相连、变迁 D 与库所 P_5 相连,得到流程 Process2.此时我们认为,流程 Process 1 与流程 Process 2 的最佳库所映射对为 P_1 与 P_1',P_2 与 P_2',P_5 与 P_5',P_6 与 P_6'.

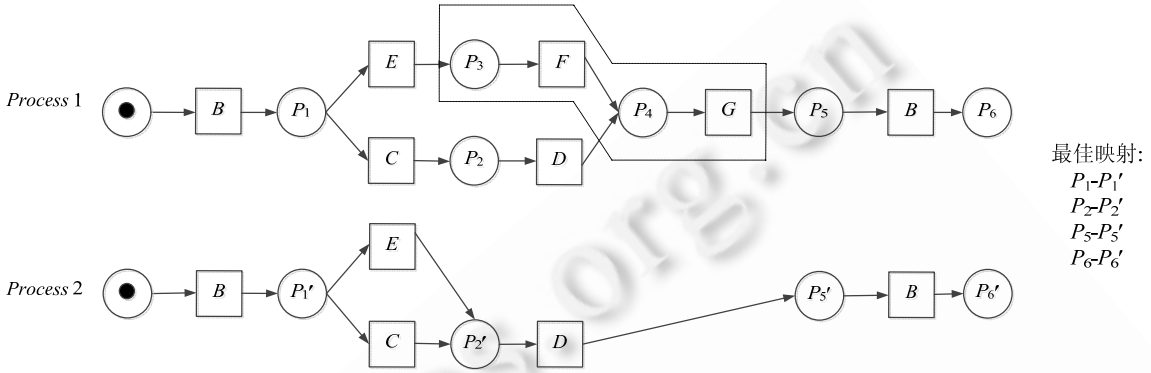


Fig.16 Case 3-1: Removing XOR-Join

图 16 Case 3-1:去 XOR-Join 路由

图 17(case 3-2)表示去 XOR-Split 路由,我们去掉虚线框中的库所 P_2,P_3 和变迁 B,C,连接库所 P_1 与变迁 E 以及库所 P_4 与变迁 D,得到流程“Process 2”.我们认为,它们的最佳库所映射对为 P_1 与 P_1',P_4 与 P_4',P_5 与 P_5',P_6 与 P_6'.

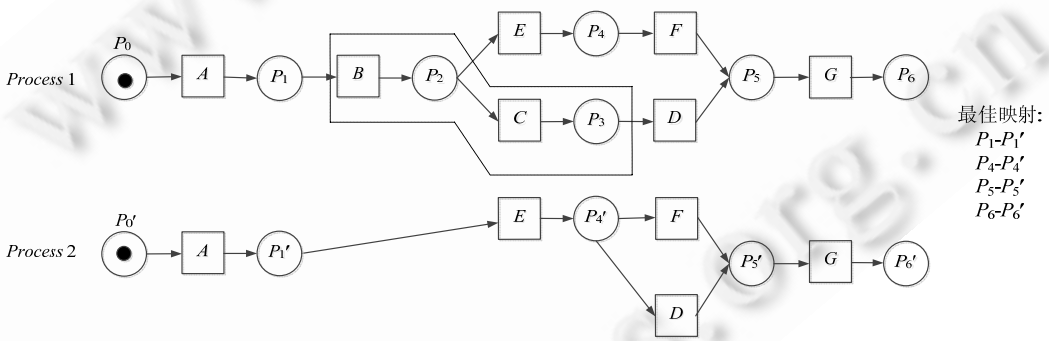


Fig.17 Case 3-2: Removing XOR-Split

图 17 Case3-2:去 XOR-Split 路由

图 18 展示了去路由场景下的准确性实验结果.我们选择库所数量为 40,100 的两个流程样例,通过减少包含路由结构片断的个数构造新的流程,然后对它们进行映射.

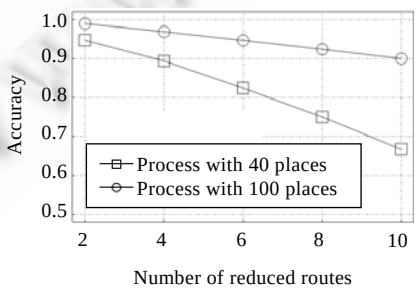


Fig.18 The accuracy after removing the routes

图 18 去路由结果

图 18 显示了随着减少的路由结构数量的增加,算法的映射准确率呈线性下降的趋势.原因是经过改造后,部分库所的映射可能有多种,即,它们的上下文相似度相同,但最终的最佳映射组合只有一组,因此准确度会下降.如图 16 中,流程 Process 2 中的库所 P_5' 可以与流程 Process 1 中的库所 P_4(因其左变迁交集均有 D)或 P_5 映射,因为它们的上下文相似度一样,均为单边映射,P_5 与 P_5' 右变迁交集元素个数为 1,P_4 与 P_5' 左变迁交集元素个数也为 1.但本文所提算法对于上下文环境相似的库所未制定进一步的匹配策略,因此对于 Process 2 中的库所 P_5',选择哪个库所进行映射是未知的.

可以看出:虽然随着改造流程的复杂度变高,我们的映射算法准确度会随之降低,但对于流程大小相差不大的流程比较场景,本文所提算法能够保持较高的准确度.如图 18 中,对于含有 40 个库所的流程与改造后去掉 10 个路由结构片断(包含有 20 个库所,即,只剩余 50%的库所)的流程进行映射,准确度在 60%以上.

4.3 效率评估

4.3.1 一对一映射

本节实验中,我们选取两个复杂度相近(库所数 257、变迁数 185、边数为 538)的流程 A 与 B 进行映射比较,分 3 个阶段考察库所、变迁以及边的数量 3 个因素对本文所提库所映射算法的性能影响,并结合其内部的映射策略实验进行分析.各组实验的具体实施方式是选择流程 B 作为参考,将流程 A 的库所、变迁和边进行人为的增加或删除等操作,然后观察各因素对映射耗时以及映射成功库所比例的影响.注意,映射成功库所的比例定义为:A 与 B 映射成功后的库所数目/流程 A 的库所总数.

如图 19、图 20 所示实验中,我们均固定流程 A 的变迁以及边数目,改变流程 A 的库所数目,并分别观察映射时间以及映射成功的库所比例随库所数的变化情况.从图 19 可以明显看出:随着库所数的增加,总映射时间不断增加的同时,除阶段 2(phase 2)双边映射外,其他阶段均有较显著地增加.阶段 1 与阶段 3 增加是因为库所的增多导致初始化映射和单边映射过程中需要进行库所间比较的次数增加,进而时间也相应的增加.阶段 2 时间增加不明显的原因可由图 20 解释,在所有的映射测试实例中可以看出:双边映射后的结果只有 1%左右,相对于其他单边映射以及无映射成功的库所数目来说,该映射结果数的变化反应到总时间中便不明显.图 20 中随着库所的增加,映射成功的库所比例值下降是因为增加的库所数并未产生新的映射,而流程 A 的总库所数目却在不断增加,所以比例会下降.

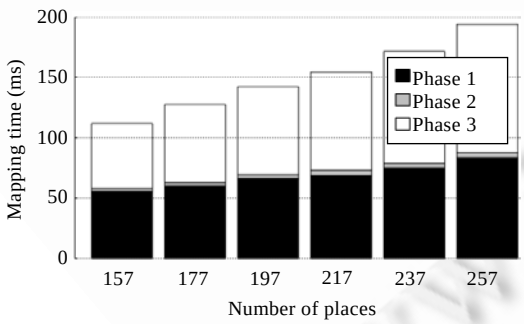


Fig.19 Mapping time with different place numbers

图 19 耗时随库所数变化的关系

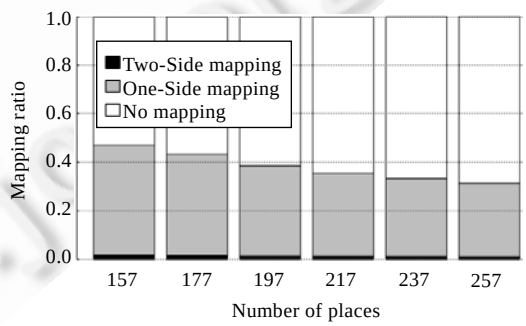


Fig.20 Mapping ratio with different place numbers

图 20 映射成功的库所比例随库所数的变化

图 21 与图 22 表示改变变迁数,固定其他因素的实验结果.随着变迁数的增加,库所的映射总时间也随着增加.但阶段 1 和阶段 2 的映射时间几乎不变,而增加的只是阶段 3 的映射时间.这是因为初始化映射的时间只与待比较的两个流程的库所数有关,若库所数不变,初始化映射时间也不会有大的波动.虽然阶段 2 包括了按笛卡尔积由大到小进行排序的过程,但根据图 22 所示阶段 2 产生的双边映射成功库所比例,其结果数对应的比较时间与总时间相比所占比例很小.同时,由图 22 可知:单边映射成功的库所比例增大,说明库所数的增加会增大单边映射的成功率,因此阶段 3 的时间会随之变大.

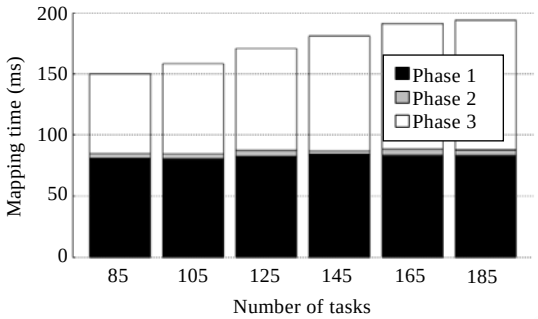


Fig.21 Mapping time with different task numbers

图 21 变迁数变化与映射时间的关系

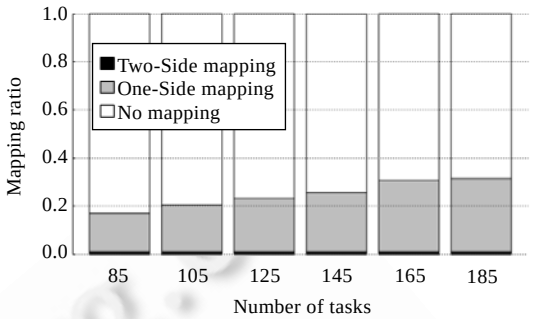


Fig.22 Mapping ratio with different task numbers

图 22 映射成功的库所比例随变迁数的变化

图 23 表示改变边数,固定其他元素数量的实验结果.随着边数的增加,库所映射的总时间随之增加,但趋势缓慢.阶段 1 和阶段 2 的映射时间几乎没有改变,而阶段 3 的映射时间略有增加.由图 24 可知:增加边数会增加双边、单边映射成功的库所比例,而单边映射的成功率比双边较为明显些,有可见的上升趋势,但趋势很缓慢,所以阶段 3 映射时间的增大速度很慢.

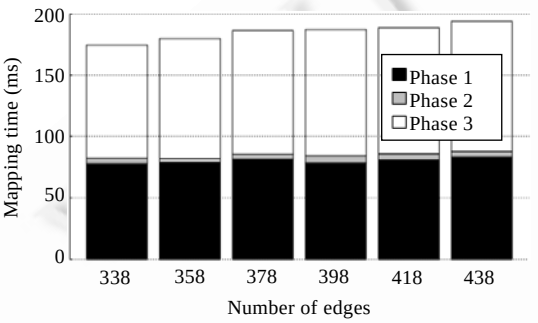


Fig.23 Mapping time with different edge numbers

图 23 边数变化与映射时间的关系

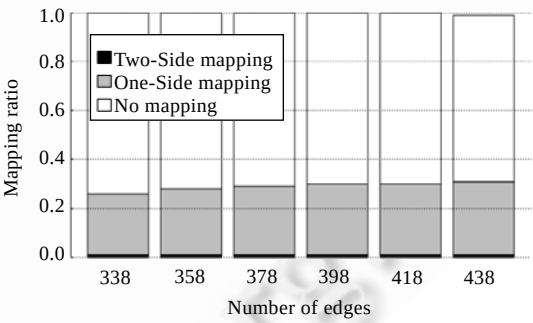


Fig.24 Mapping ratio with different edge numbers

图 24 映射成功的库所比例随边数的变化

4.3.2 一对多映射

在诸多需要对流程进行比较的应用场景当中,流程相似度计算不仅有着计算复杂的流程图匹配环节,还要在较短的时间内从包含有大量流程的流程库中检索出最为相似的流程,相对于其他应用场景,相似流程的搜索较为苛刻.因此在本节实验中,为了考察流程元素映射是否能应用到流程相似度计算场景当中,我们加入相对简单的变迁映射,考察单个流程与流程库间多个流程的一对多映射效率.

我们选取A,B_3,C这3个具有不同复杂度的流程库,并在各自的流程库中挑选复杂度相近的一些流程进行一对多映射的实验,映射总时间包括变迁映射的总时间和库所映射的总时间.与之前实验类似,我们分别考察单个流程的库所、变迁、边的数量变化与流程库间映射总时间的影响.

图 25 表示其他因素不变的情况下,与流程库总映射时间随库所数变化情况.可以看出:随着库所数的增加,流程与 3 个流程库的映射总时间均线性增加.这是因为根据上一节的实验分析,库所元素的映射时间占了流程间元素映射总时间的绝大部分,因此对于同一个流程库来说,库所的增加将导致总映射时间的增加.3 个流程库中,由于 C 库的流程数量最多(1 702 个流程),所以其映射时间所有测试情况最长,最大为 7s 左右,其次是 B(421 个流程)为 5.7s 左右,A(284 个流程)为 3.2s 左右.

图 26 是映射时间随变迁数变化的实验结果.随着变迁数的增加,流程与 3 个流程库的映射总时间均增加.这是因为增大变迁数,一方面增加了变迁映射的时间,另一方面也增大了双边、单边映射库所数的比例,从而增

加了库所映射的时间.在变迁数较小时,变迁数的增加,使得其对流程与 *C* 库的映射总时间影响最大,其次是 *B* 库、*A* 库;而在变迁数较大时,变迁数的增加使得其对流程与 *B* 库的映射总时间影响最大,其次是 *C* 库、*A* 库.其原因是:在变迁数较小时,流程与流程库之间的映射时间由流程库的大小决定,流程库越大,映射时间越大;*C* 库拥有众多复杂度低的流程,而 *B* 库拥有众多复杂度较高的流程,当单个流程的变迁数变大后,可以使得单个流程与 *B* 库中较高复杂度的流程之间映射的变迁数增加,双边、单边映射的库所数增加,从而流程与 *B* 库的映射总时间超过了其与 *C* 库的映射总时间.而 *A* 库的变迁数量小的流程、变迁数量多的流程的数量均比 *B*,*C* 库中的少,因此发生映射的可能性比 *B*,*C* 库小.

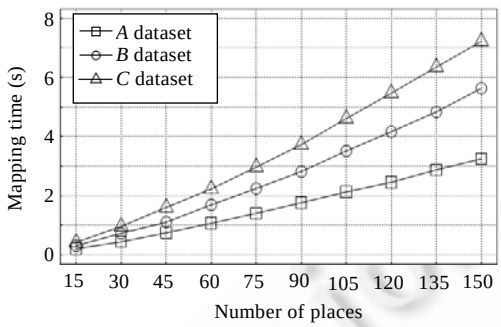


Fig.25 Mapping time with different place numbers

图 25 库所数变化与映射时间的关系

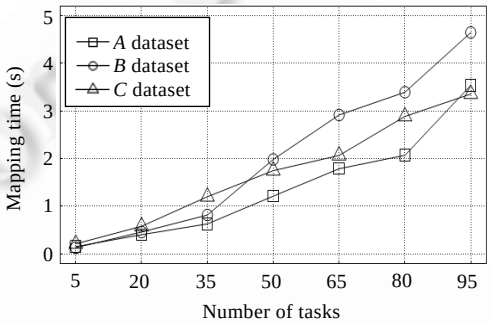


Fig.26 Mapping time with different task numbers

图 26 变迁数变化与映射时间的关系

图 27 展示的是边数对映射时间的影响.随着边数的增加,3 个流程库的映射总时间均增加,因为边数的增加会增大双边、单边映射库所的比例,从而增加库所映射的时间.边的增加,对流程与 *C* 库的映射总时间的影响最大,其次是 *B* 库、*A* 库,这也是与流程库中流程的数目决定的.

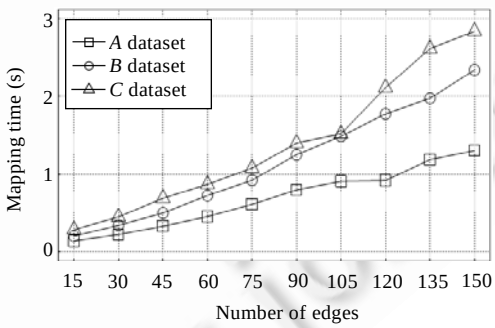


Fig.27 Mapping time with different edge numbers

图 27 边数变化与映射时间的关系

综上可以看出:流程与流程库进行映射的总时间均随着库所、变迁、边的数目增加而增加,其中,库所数目的增加使得时间的变化最为明显,如图 25 中由 1s 增加到 8s.对于较复杂的流程来说,若在流程相似度计算时先进行元素间的映射,会使得相似流程的搜索速度变慢(增加了映射的时间).但就本文实验的结果结合相似流程搜索在实际中的时间需求来看,由于元素映射额外增加的时间是可以接受的.此外,在大型流程库和复杂流程映射匹配时(如包含 150 个库所的流程与含有 1 702 个流程的 *C* 库进行映射),其总映射时间在秒级别,因此可以认为,本文所提的流程元素间映射方法可以满足实际应用的性能需求.

5 相关工作

据我们所知,目前专门针对流程间元素映射的研究很少.在大多数涉及流程比较过程的相关研究中,对流程间元素(即任务、资源等)映射考虑均做了简化处理.Jochen 等人[14]在进行流程模型差别检测过程中,利用 SESE (single-entry-single-exit)[15]方法将流程分解为不同的片断,然后对这些片断进行了映射处理.但遗憾的是,他们并未对连接 SESE 片断的路由判断等节点提出相应的映射算法,而只是人为的对其进行了映射.Dijkman 等人[2]在流程相似度计算[16]场景中,将不同建模语言表达的流程模型逐步抽象成了经典的点线图,使得流程中除任务本身外的信息丢失,如路由情况等,无法保证后续结果的可靠性.此外,Dijkman 在流程差别检测方面也做了一些研究:首先,对相似流程间的差别进行了分类[17],这些差别类型包括业务流程中的业务活动差别、业务活动间的控制流差别以及执行这些活动的执行者权力差别;随后在该分类基础上,利用相等性概念,设计了用来检测业务流程间差别的方法[18].但其并未在这些差别分类和检测方法中考虑数据流、判断条件差别等情况,即没有考虑如何对这些元素在流程间进行映射,使得其方法无法检测出实际存在的差别.流程版本控制的应用场景当中,也需要对流程进行比较.Weber 等人[19]对流程发生改变的模式进行了归类,但其中的一些模式仍需要人工的判断,如哪些非任务节点的元素随着相关任务的改变而发生变化等.Aalst 等人[20]同样利用 Petri 网建模,提出了一种基于流程执行行为的流程模型比较方法,但该方法只考虑任务活动在模型中的可能执行次序,忽略了库所在不同流程模型间的映射.

6 总结与未来工作

本文针对现有流程比较方法中无法有效进行流程间元素映射的问题,提出了一种基于 Petri 网的流程间元素映射方法.该方法侧重于 Petri 网流程模型中的库所映射机制,其出发点为基于库所左、右变迁交集的上下文环境相似度.利用该上下文相似度,我们给出了针对库所映射的算法实现,即:先将 Petri 网转换成映射模型,再对两个模型进行初始化映射,接着对库所按照双向、单向的优先级进行映射,最终得到映射结果.在本文的实验环节,我们基于真实的数据集,从准确性和效率两方面对所提方法进行了评估.在准确性实验中,通过 3 种不同测试情况的考察,基于上下文相似的库所映射能够在流程图结构中节点个数相同不大时具有较高的准确性;在效率测试方面,分阶段对流程模型间一对一的映射进行了考察和分析,还借鉴流程相似度计算场景,对模型与模型库间一对多映射进行了的整体映射效率的评估.实验结果表明,对本文所提方法在准确性和性能上能够满足实际的应用场景.

在未来工作中,我们拟考虑如下几个方面的工作:

(1) 利用本文进行库所映射时,目前只能找到一组映射.但事实上,基于图 5 的讨论,有多组映射方式可选.如何找到所有的或多个映射组合,是我们在将来需要进一步考虑的问题;

(2) 在多个潜在的映射组合中,造成不同组合差别的原因是什么、对实际业务流程管理应用场景的影响如何、哪些映射组合是最优的等问题,也需要在未来深入研究;

(3) 基于本文对流程元素映射的思想,如何将其进一步与具体的应用场景结合,如流程间差别计算、流程相似度计算等,并针对这些场景进行效率和准确度等方面的研究,也是我们未来的工作.

References:

[1] Weber B, Rinderle S, Reichert M. Change patterns and change support features in process-aware information systems. In: Proc. of the Advanced Information Systems Engineering. 2007. 574−588. [doi: 10.1007/978-3-540-72988-4_40]

[2] Dijkman R, Dumas M, Garcia-Banuelos L. Graph matching algorithms for business process model similarity search. In: Proc. of the Business Process Management. 2009. 48−63. [doi: 10.1007/978-3-642-03848-8_5]

[3] Van Der Aalst W, Arya A, Van Dongen B. Replaying history on process models for conformance checking and performance analysis. Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery, 2012,2(2):182−192. [doi: 10.1002/widm.1045]

[4] La Rosa M, Dumas M, Uba R, Dijkman R. Merging business process models. In: Proc. of the Move to Meaningful Internet Systems. 2010. 96−113. [doi: 10.1007/978-3-642-16934-2_10]

[5] White SA. Introduction to BPMN. IBM Cooperation 2.0. 2004.

[6] Yuan CY.Theory and Application of Petri Nets. Beijing: Publishing House of Electronics Industry, 2005 (in Chinese).

[7] Dijkman R, Dumas M, Van Dongen B, Käärik R, Mendling J. Similarity of business process models: Metrics and evaluation. Information Systems, 2011,36(2):498−516. [doi: 10.1016/j.is.2010.09.006]

[8] Van Der Aalst WMP, Ter Hofstede AH, Kiepuszewski B, Barros AP. Workflow patterns. Distributed and Parallel Databases, 2003, 14(1):5−51. [doi: 10.1023/A:1022883727209]

[9] Van Der Aalst WMP. Three good reasons for using a Petri-net-based workflow management system. In: Proc. of the Information and Process Integration in Enterprises. 1996. 179−201.

[10] Soria C, Monachini M, Vossen P. Wordnet-LMF: Fleshing out a standardized format for wordnet interoperability. In: Proc. of the 2009 Int'l Workshop on Intercultural Collaboration. ACM Press, 2009. 139−146. [doi: 10.1145/1499224.1499246]

[11] Cai Q, Yates A. Large-Scale semantic parsing via schema matching and lexicon extension. In: Proc. of the ACL. 2013. 423−433.

[12] Zhang CJ, Zhao Z, Chen L, Jagadish HV, Cao CC. CrowdMatcher: Crowd-assisted schema matching. In: Proc. of the 2014 ACM SIGMOD Int'l Conf. on Management of Data. 2014. 721−724. [doi: 10.1145/2588555.2594515]

[13] Fahland D, Favre C, Jobstmann B, Koehler J, Lohmann N, Volzer H, Wolf K. Instantaneous soundness checking of industrial business process models. In: Proc. of the Business Process Management. 2009. 278−293. [doi: 10.1007/978-3-642-03848-8_19]

[14] Küster JM, Gerth C, Förster A, Engels G. Detecting and resolving process model differences in the absence of a change log. In: Proc. of the Business Process Management. 2008. 244−260. [doi: 10.1007/978-3-540-85758-7_19]

[15] Vanhatalo J, Völzer H, Leymann F. Faster and more focused control-flow analysis for business process models through sese decomposition. In: Proc. of the ICSOC. 2007. 43−55. [doi: 10.1007/978-3-540-74974-5_4]

[16] Yan Z, Dijkman R, Grefen P. Fast business process similarity search. Distributed and Parallel Databases, 2012,30(2):105−144. [doi: 10.1007/s10619-012-7089-z]

[17] Dijkman R. A classification of differences between similar BusinessProcesses. In: Proc. of the 11th IEEE Int'l Enterprise Distributed Object Computing Conf. 2007. 37−47. [doi: 10.1109/EDOC.2007.24]

[18] Dijkman R. Diagnosing differences between business process models. In: Proc. of the Business Process Management. Berlin, Heidelberg: Springer-Verlag, 2008. 261−277. [doi: 10.1007/978-3-540-85758-7_20]

[19] Weber B, Reichert M, Rinderle-Ma S. Change patterns and change support features—Enhancing flexibility in process-aware information systems. Data & Knowledge Engineering, 2008,66(3):438−466. [doi: 10.1016/j.datak.2008.05.001]

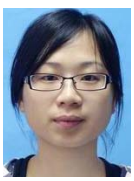
[20] Van Der Aalst WMP, De Medeiros AA, Weijters AJMM. Process equivalence: Comparing two process models based on observed behavior. In: Proc. of the Int'l Conf. on Business Process Management. 2006. 129−144. [doi: 10.1007/11841760_10]

附中文参考文献:

[6] 袁崇义.Petri 网原理与应用.北京:电子工业出版社,2005.

曹斌(1985－),男,山西孟县人,博士,讲师,CCF 会员,主要研究领域为业务流程管理,大数据.

范菁(1969－),女,博士,教授,博士生导师,CCF 高级会员,主要研究领域为服务计算,虚拟现实.



王佳星(1990－),女,博士生,CCF 学生会员,主要研究领域为工作流.

董天阳(1977－),男,博士,副教授,CCF 会员,主要研究领域为虚拟现实,数据挖掘.