Information Sciences 373 (2016) 369–387

Contents lists available at ScienceDirect

Information Sciences

journal homepage: www.elsevier.com/locate/ins

Enhancing discovered processes with duplicate tasks

Borja Vázquez-Barreiros*, Manuel Mucientes, Manuel Lama

Centro Singular de Investigación en Tecnoloxías da Información (CiTIUS), Universidade de Santiago de Compostela, Santiago de Compostela, Spain

ARTICLE INFO

Article history:
Received 15 September 2015
Revised 19 July 2016
Accepted 5 September 2016
Available online 5 September 2016

Keywords:
Duplicate tasks
Process discovery
Petri nets

ABSTRACT

Including duplicate tasks in the mining process is a challenge that hinders the process discovery, as it is also necessary to find out which events of the log belong to which transitions. To face this problem, we propose SLAD (Splitting Labels After Discovery), an algorithm that uses the local information of the log to enhance an already mined model, by performing a local search over the tasks that have more probability to be duplicated in the log. This proposal has been validated with 54 different mined models from three process discovery algorithms, improving the final solution in 45 of the cases. Furthermore, SLAD has been tested in a real scenario.

1. Introduction

In the recent years, a lot of work has been made for developing technologies to automate the execution of processes in different application domains such as industry, education or medicine [20]. In particular, for business processes, there has been an incredible growth on the amount of process-related data, i.e, execution traces of business activities. Within this context, process mining has emerged as a way to analyze the behavior of an organization based on these data —event logs— offering techniques to discover, monitor and enhance real processes, i.e., to understand *what is really happening in a business process* [25].

Based on this idea, and in order to model what is really happening in an organization, *process discovery* techniques aim to find the process model that *better* portraits the behavior recorded in an event log. There are four quality dimensions to measure *how good* is a model and, hence, identify which model is the *best*: fitness replay, precision, generalization and simplicity. *Fitness replay* measures how much of the behavior recorded in the log can be reproduced in the process model. On the other hand, *precision* and *generalization* measure if the model overfits —it disallows for new behavior not recorded in the log— or underfits —it allows additional behavior not recorded in the log— the data, respectively. Finally, *simplicity*, quantifies the complexity of the model, for instance, the number of arcs and tasks. Hence, the idea behind process discovery is to maximize these metrics in order to obtain an *optimal* solution that better describes the flow of the events that occur within the process.

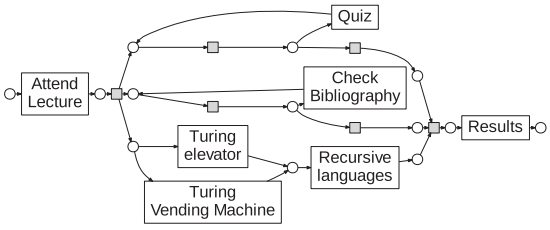
In order to discover the optimal solution, one key question is *how to describe the ordering and flow of events that occur in a process* [28]. From the control-flow perspective, a model can be represented with many different workflow patterns, such as *sequences*, *parallels*, *loops*, *choices*, etc. Furthermore, to improve the quality of the solution, models can be extended with

* Corresponding author. Tel.: +34 8818 16390.
E-mail addresses: borja.vazquez@usc.es (B. Vázquez-Barreiros), manuel.mucientes@usc.es (M. Mucientes), manuel.lama@usc.es (M. Lama).

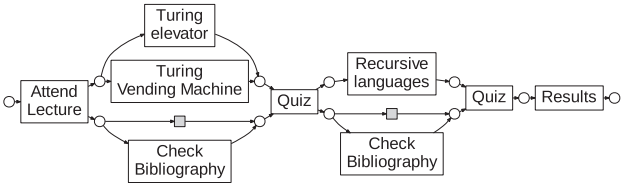
http://dx.doi.org/10.1016/j.ins.2016.09.008
0020-0255/

	Events
$case_1$	Attend lecture, Turing Elevator, Check Bibliography, Quiz, Recursive Languages, Quiz, Results.
$case_2$	Attend lecture, Turing Vending Machine, Check Bibliography, Quiz, Recursive Languages, Check Bibliography, Quiz, Results.
$case_3$	Attend lecture, Check Bibliography, Turing Vending Machine, Quiz, Check Bibliography, Recursive Languages, Quiz, Results.
$case_4$	Attend lecture, Turing Elevator, Check Bibliography, Quiz, Check Bibliography, Recursive Languages, Quiz, Results.
$case_n$	...

(a) *Log*.



(b) *Model without duplicate tasks*.



(c) *Model with duplicate tasks*.

Fig. 1. A log and two process models —Petri nets— exemplifying a lecture of automata theory and formal languages.

more behavior: *duplicate activities*, *non-free-choice constructs*[1], etc. Within the scope of this paper, the notion of duplicate tasks —or activities— [30] refers to situations in which multiple tasks in the process have the same label, i.e., they can appear more than once in the process. As previously said, the inclusion of duplicate tasks is useful to improve the precision and simplicity of a model, and, hence, enhance its comprehensibility [3,10]. Fig. 1 shows an example on how the addition of duplicate tasks to a model improves its understandability and structural clarity. In this example, considering the sample log of Fig. 1a, the events *Quiz* and *Check Bibliography*, are executed multiple times —twice in each trace. Between the multiple possibilities of modeling the behavior of the log, we can assume i) an injective relation between the events in the log and the activities in the model (Fig. 1b); or ii) that multiple activities can share the same label, i.e. a model with duplicate activities (Fig. 1c). In this example, although both models perfectly reproduce all the behavior recorded in the log —both have a perfect replay fitness—, the model depicted in Fig. 1b allows to execute both *Quiz* and *Check Bibliography* as many times as we want at any time in the process, hence, this model is not a precise picture of the recorded behavior of the log —its precision is lower. On the other hand, if both activities are duplicated, the resulting model (Fig. 1c) is more suitable, i.e., more precise with respect to the recorded behavior in the log, as it does not allow, for example, to check the bibliography —*Check Bibliography*— during the exam —*Quiz*. Hence, the ability to discover these duplicate tasks *may* greatly enhance the comprehensibility of the final solution, and create a more specific process model.

From the perspective of process discovery, including duplicate tasks in the mining process is a well known challenge [29,30] as, usually, duplicate tasks are recorded with the same label in the log, hindering the discovery of the model that better fits the log —algorithms need to find out which events of the log belong to which tasks. To face this issue, handling duplicate tasks is usually considered as a pre-mining step, i.e., the potential duplicate tasks are identified and accordingly labeled *before* mining the log, or as part of the process discovery algorithm. Within this context, there are several techniques in the state of the art that allow to mine duplicate tasks [3,6,8,9,13–15,19]. However, some approaches can retrieve worse solutions than without duplicate activities [3], others can duplicate any activity of the log without imposing any limit to the number of activities that may be duplicated [6,8,13], or they have problems when dealing with duplicate activities involved in certain constructs, such as loops [9,19].

In this paper we analyze the possibility of tackling duplicate tasks *after* mining a process model, in particular, after mining a causal net [27] or heuristic net [38], without adversely affecting the quality of the initial solution. Hence, we present SLAD (Splitting Labels After Discovery), a novel algorithm that enhances an already discovered process model by splitting the behavior of its activities. Firstly, a process discovery technique mines a log without considering duplicate tasks, generating a causal net or a heuristic net. Then, SLAD, using the local information of the log and the retrieved model, tries to improve the quality of the model by performing a local search over the tasks that have more probability to be duplicated in the log. The contributions of this proposal are: i) the discovering of the duplicate activities is performed *after* the discovery process, in order to *unfold* the overly connected nodes than may introduce extra behavior not recorded in the log; ii) new

[1] A non-free choice (NFC) construct is a special kind of choice, where the selection of a task depends on what has been executed before in the process model.