Mining Duplicate Tasks from Discovered Processes

Borja Vázquez-Barreiros, Manuel Mucientes, and Manuel Lama

Centro de Investigación en Tecnoloxías da Información (CiTIUS)
Universidade de Santiago de Compostela, Santiago de Compostela, Spain
{borja.vazquez,manuel.mucientes,manuel.lama}@usc.es

Abstract. Including duplicate tasks in the mining process is a challenge that hinders the process discovery as algorithms need an extra effort to find out which events of the log belong to which transitions. To face this problem, we propose an approach that uses the local information of the log to enhance an already mined model by performing a local search over the potential tasks to be duplicated. This proposal has been validated over 36 different solutions, improving the final model in 35 out of 36 of the cases.

Keywords: Process mining, process discovery, duplicate tasks.

1 Introduction

The notion of duplicate tasks —or activities— refers to situations in which multiple tasks in the process have the same label. This kind of behavior is useful when i) a particular task is used in different contexts in a process and ii) to enhance the comprehensibility of a model. Typically, duplicate tasks are recorded with the same label in the log and, hence, they hinders the discovery of the model that better fits the log, as algorithms need an extra effort to find out which events of the log belong to which transitions. There are several techniques allowing to mine duplicate tasks [2,3,4,5,6,7], however, or the heuristics rules used to detect the duplicate tasks are not sufficiently general for all the logs [7], or they have to deal with a large search space, increasing the time needed for these algorithms [3,5,6].

In this paper we present a novel proposal to tackle duplicate tasks. The proposal starts from an already mined model without duplicate tasks, and uses the local information of the log and the retrieved process to improve the model through a local search over the potential duplicate tasks.

2 Local search algorithm

Algorithm 1 describes the proposed approach to tackle duplicate tasks. The first step is the discovery of the potential duplicate activities. We used the heuristics defined in [5] to reduce the search space by stating that two tasks with the same

Algorithm 1: Local search Algorithm.

```
input: A log $L$
$ind_0 \leftarrow$ initial_solution(L)                    // Retrieved by a process discovery technique.
$potentialDuplicates \leftarrow \emptyset$
foreach activity $t$ in the log $L$ do
    if $\max(\min(|t >_L t'|, |t' >_L t|), 1) > 1$ then
        $potentialDuplicates \leftarrow potentialDuplicates \cup t$
$ind_0 \leftarrow$ localSearch ($ind_0$, $L$, $potentialDuplicates$, true)
Function localSearch( $ind_0$, $L$, $potentialDuplicates$, $firstExecution$)
    $ind_{best} \leftarrow ind_0$
    $potentialDuplicatesL2L \leftarrow \emptyset$
    foreach activity $t$ in $potentialDuplicates$ do
        combinations $\leftarrow$ calculateCombinations ($ind_0$, $L$, t)
        foreach combination $c$ in combinations do
            $t' \leftarrow$ activity $t$ from $ind_0$
            $t.inputs = (t.inputs \setminus c.inputs) \cup c.sharedInputs$
            $t'.inputs = c.inputs$
            $t.outputs = (t.outputs \setminus c.outputs) \cup c.sharedOutputs$
            $t'.outputs = c.outputs$
            if $(I(t') \neq \emptyset$ && $O(t') \neq \emptyset$ && $I(t) \neq \emptyset$ && $O(t) \neq \emptyset)$ then
                Add task $t'$ to individual $ind_0$ and update $t$ in $ind_0$
                Repair $ind_0$
                Post-prune unused arcs
                Evaluate $ind_0$
                if $ind_0 < ind_{best}$ then
                    $ind_0 \leftarrow ind_{best}$
                else
                    $ind_{best} \leftarrow ind_0$
                    $potentialDuplicatesL2L = potentialDuplicatesL2L \cup t''$ where
                    $t'' \notin potentialDuplicates$ and $t >_L t''$
            else
                $ind_0 \leftarrow ind_{best}$
    if $firstExecution$ then
        $ind_{best} \leftarrow$ localSearch ($ind_{best}$, $null$, $potentialDuplicatesL2L$, $false$)
    return $ind_{best}$
```

label cannot share the same input and output dependencies. Within this context, the duplicate tasks are locally identified based on the *follows relation* ($>_L$), where the upper bound for an activity t is the minimum of the number of tasks that directly precede t in the log and the number of tasks that directly follow t. This definition can be formalized as [5]: $max(min(|t >_L t'|, |t' >_L t|), 1)$. If for a task t the upper bound is greater than 1, then t is considered as a potential task for being duplicated and, hence, it is added to *potentialDuplicates* (Alg.1:3-5).

After finding the potential duplicates, the algorithm splits the input and output dependencies of the activities of the model into multiple tasks with the same label through the function *localSearch* (Alg.1:7). In this step, the algorithm calculates the input and output combinations for each activity in *potentialDuplicates* (Alg. 1:10-11) through the function *CalculateCombinations* (Alg.2). Within this function, the algorithm first finds all the subsequences in the log L that match the pattern $t_1 t t_2$ where $t_1 \in I(t)$ and $t_2 \in O(t)$ in the model (Alg.2:2) —being $I(t)$ and $O(t)$ the inputs and outputs, respectively, of t. Then, based on these

Algorithm 2: Algorithm to compute the combinations of a task.

```
Function calculateCombinations($ind$, $L$, $t$)
    /* If the input parameter $L$ is null, retrieve the sequences from parsing $ind$ */
    Retrieve all the subsequences $t_1tt_2$ where $t_1 \in I(t)$ and $t_2 \in O(t)$
    $combinations \leftarrow \emptyset$
    forall the $subsequences\ t_1tt_2$ do
        $c \leftarrow \emptyset$
        Create a set $c.inputs$ with the combinations that share the same $t_1$ and add in
        $c.outputs$ their respective $t_2$
        Add $c$ to $combinations$
    foreach $c$ in $combinations$ do
        if $c.outputs = c'.outputs$ where $c' \in combinations$ then
            $c.inputs = c.inputs \cup c'.inputs$ and $c.outputs = c.outputs \cup c'.outputs$
            $combinations = combinations \setminus c'$
        if $c.outputs$ shares an element $e$ with another $c'.outputs$ then
            $c.sharedOutputs \leftarrow c.sharedOutputs \cup e$
        if $c.inputs$ shares an element $e$ with another $c'.inputs$ then
            $c.sharedInputs \leftarrow c.sharedInputs \cup e$
    return $combinations$
```

subsequences, the combinations are created following three rules (Alg.2:4-15). First, given two subsequences t_1tt_2 and t_3tt_4, if $t_1 = t_3$, then we merge both subsequences into a new combination (Alg.2:4-7). Later, given two different combinations c and c', if they share the same *output*, i.e., $c.output = c'.output$, these two combinations are merged (Alg.2:9-11). Finally, if the intersection between two combinations is not the empty set, we have to record which elements are shared by both combinations (Alg.2:12-15).

After creating all the possible combinations, for each combination c (Alg.1:12), the algorithm creates a new task t' equal to the original activity t of the current model (Alg.1:13). Then, it removes from $I(t)$ all the tasks shared with *c.inputs*, but keeping the tasks that are in *c.sharedInputs* (Alg.1:14). On the other hand, for the new task t', it retains only the elements in $I(t')$ that are contained in *c.inputs* (Alg.1:15). The same process is applied for the outputs of both t and t' but with *c.outputs* and *c.sharedOutputs* (Alg.1:16-1:17). If both the inputs and outputs of these tasks are not empty (Alg.1:18), they are included in ind_0 (Alg. 1:19). Otherwise the model goes back to its previous state and tries with a new combination. If the new task is included, the model is repaired (Alg.1:20) and the unused arcs are removed (Alg.1:21). In order to evaluate the models (Alg.1:22), we based the quality of a solution on three criteria: *fitness replay*, *precision* and *simplicity*. To measure these criteria we used the hierarchical metric defined in [10]. If the new model is better, the best individual ind_{best} is replaced with ind_0 (Alg.1:26). Otherwise the model goes back to its previous state and repeats the process with a new combination.

The main drawback of the heuristic followed to detect the possible duplicate tasks of the log (Alg.1:3 [5]) is that it does not cover all the search space, particularly with tasks involved in a length-two-loop situation, as it breaks the rule of two tasks sharing the same input and output dependencies. To solve this,

Table 1: Results for the 18 logs with the initial solutions of ProDiGen and HM.

		Logs																	
		Alpha	Folded	Loop	Fig6p25	betaSimpl.	flighCar	Fig5p19	Fig5p1AND	Fig5p1OR	Fig6p10	Fig6p31	Fig6p33	Fig6p34	Fig6p38	Fig6p39	Fig6p42	Fig6p9	RelProc
ProDiGen	C	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
	P	0.8	0.75	0.79	0.75	0.86	0.81	0.9	0.76	0.73	0.78	0.61	0.65	0.73	0.93	0.93	0.7	0.79	0.89
	S	0.3	0.3	0.29	0.29	0.3	0.29	0.3	0.29	0.29	0.29	0.26	0.28	0.29	0.31	0.3	0.28	0.3	0.30
ProDiGen + LS	C	1.0	*1.0*	*1.0*	*1.0*	*1.0*	*1.0*	*1.0*	*1.0*	*1.0*	*1.0*	*1.0*	*1.0*	*1.0*	*1.0*	1.0	*1.0*	*1.0*	*1.0*
	P	0.85	*1.0*	*0.99*	*0.97*	*0.94*	*1.0*	*1.0*	*1.0*	*1.0*	*0.96*	*1.0*	*1.0*	*0.93*	*1.0*	0.94	*1.0*	*1.0*	*0.95*
	S	0.31	*0.3*	*0.3*	*0.3*	*0.31*	*0.31*	*0.31*	*0.33*	*0.33*	*0.31*	*0.31*	*0.31*	*0.31*	*0.33*	0.3	*0.3*	*0.32*	*0.32*
HM	C	1.0	1.0	1.0	1.0	1.0	0.32	1.0	0.67	1.0	1.0	1.0	1.0	0.41	0.0	0.0	0.07	0.21	1.0
	P	0.72	0.75	0.79	0.73	0.86	0.81	0.9	0.75	0.67	0.78	0.56	0.6	0.76	0.57	0.6	0.64	0.95	0.89
	S	0.3	0.3	0.29	0.28	0.31	0.29	0.31	0.33	0.31	0.29	0.26	0.27	0.29	0.28	0.29	0.28	0.32	0.30
HM + LS	C	1.0	*1.0*	*1.0*	1.0	1.0	0.32	*1.0*	0.67	*1.0*	*1.0*	*1.0*	*1.0*	0.72	*1.0*	0.53	0.36	0.21	*1.0*
	P	0.81	*1.0*	*0.99*	0.94	0.93	0.82	*1.0*	1.0	*1.0*	*0.96*	*1.0*	*1.0*	0.98	*1.0*	0.94	0.95	0.95	*0.95*
	S	0.31	*0.3*	*0.3*	0.31	0.32	0.30	*0.31*	0.35	*0.33*	*0.31*	*0.31*	*0.31*	0.32	*0.33*	0.32	0.31	0.32	*0.32*

we have to make all the process iterative: when for a task t, $max(min(|t >_L t'|, |t' >_L t|), 1)$ is greater than 1, i.e, t is detected as a duplicate activity, the upper bound for all the tasks t' that directly follow t must be updated, because these tasks will now have multiple tasks with the same label as input. Hence, if a task t is correctly duplicated in the model (Alg.1:26), we add the tasks that directly follow t —and that weren't detected as possible duplicated tasks in the first step— into *potentialDuplicatesL2L* (Alg.1:27). Therefore, the last step of the algorithm (Alg.1:31) involves a new execution of the function *localSearch* (Alg.1:7) but with *potentialDuplicatesL2L* instead of *potentialDuplicates*. In this second and final execution, the subsequences are obtained from the process model —note that in the first execution the subsequences were extracted from the log. Therefore, the algorithm parses the solution, checking which one of the activities with the same label $t' \in I(t)$ were executed just before t and which activities $t'' \in O(t)$ were executed after t. Finally, it creates the combinations based on this information.

3 Experimentation

The validation of the presented approach has been done with several synthetic logs from [5,7]. We used ProDiGen [10] and HM [11] over these set of logs to retrieve the initial solutions. On the other hand, the quality of the models was measured taking into account three metrics: fitness replay (C) [8], precision (P) [1] and simplicity (S) [9] . Table 1 shows the results retrieved before applying the presented approach —the raw solutions mined with ProDiGen and HM— and after the local search. Moreover, they show information about which algorithm retrieves better results for each metric —highlighted in grey— and which solutions are equal to the original model —highlighted in *italics*.

After applying our approach over the solutions, the proposed local search was able to enhance the results in 35 out of 36 of the cases. More specifically, the algorithm was able to i) *significantly* improve the precision, and ii) to reduce the complexity of the different models by splitting the behavior of the overly connected nodes. Furthermore, our approach was able to retrieve the original model in 25 out of 36 cases.

4 Conclusions

We have presented an approach to tackle duplicate tasks in an already discovered model. Our proposal takes as starting point a model without duplicate tasks and its respective log, and based on the local information of the log and the causal dependencies of the input mined model, it improves the comprehensibility of the solution. The presented approach has been validated with 36 different models with duplicate tasks. Results conclude that this local search is able to detect all the potential duplicate tasks in the log, and enhance the comprehensibility of the final model, by improving its fitness replay, precision and simplicity.

Acknowledgments

This work was supported by the Spanish Ministry of Economy and Competitiveness under project TIN2014-56633-C3-1-R, and the Galician Ministry of Education under the projects EM2014/012 and CN2012/151.

References

1. Adriansyah, A., Munoz-Gama, J., Carmona, J., van Dongen, B.F., van der Aalst, W.M.P.: Alignment based precision checking. In: BPM. (2012) 137–149
2. Broucke, S.K.V.: Advances in Process Mining. PhD thesis
3. Buijs, J.C.A.M., van Dongen, B.F., van der Aalst, W.M.P.: Quality dimensions in process discovery: The importance of fitness, precision, generalization and simplicity. International Journal of Cooperative Information Systems **23**(1) (2014)
4. Carmona, J., Cortadella, J., Kishinevsky, M.: A region-based algorithm for discovering petri nets from event logs. In: BPM. Springer (2008) 358–373
5. de Medeiros, A.: Genetic Process Mining. PhD thesis, TU/e (2006)
6. Goedertier, S., Martens, D., Vanthienen, J., Baesens, B.: Robust process discovery with artificial negative events. The Journal of Machine Learning Research **10** (2009) 1305–1340
7. Li, J., Liu, D., Yang, B.: Process mining: Extending α-algorithm to mine duplicate tasks in process logs. In: Advances in Web and Network Technologies, and Information Management. (2007) 396–407
8. Rozinat, A., van der Aalst, W.M.P.: Conformance checking of processes based on monitoring real behavior. Information Systems **33**(1) (2008) 64–95
9. Sánchez-González, L., Garca, F., Mendling, J., Ruiz, F., M.Piattini: Prediction of business process model quality based on structural metrics. In: Conceptual Modeling ER 2010. Volume 6412. (2010) 458–463
10. Vázquez-Barreiros, B., Mucientes, M., Lama, M.: ProDiGen: Mining complete, precise and minimal structure process models with a genetic algorithm. Information Sciences **294** (2015) 315–333
11. Weijters, A., van der Aalst, W.M.P., de Medeiros, A.: Process mining with the heuristics miner-algorithm. Technische Universiteit Eindhoven **166** (2006)