On the Use of Hierarchical Subtrace Mining for Efficient Local Process Model Mining*

Niek Tax, Laura Genga, and Nicola Zannone

Eindhoven University of Technology
{n.tax, l.genga, n.zannone}@tue.nl

Abstract. Mining local patterns of process behavior is a vital tool for the analysis of event data that originates from flexible processes, for which it is generally not possible to describe the behavior of the process in a single process model without overgeneralizing the behavior allowed by the process. Several techniques for mining such local patterns have been developed throughout the years, including Local Process Model (LPM) mining and the hierarchical mining of frequent subtraces (i.e., subprocesses). These two techniques can be considered to be orthogonal, i.e., they provide different types of insights on the behavior observed in an event log. As a consequence, it is often useful to apply both techniques to the data. However, both techniques can be computationally intensive, hindering data analysis. In this work, we explore how the output of a subtrace mining approach can be used to mine LPMs more efficiently. We show on a collection of real-life event logs that exploiting the ordering constraints extracted from subtraces lowers the computation time needed for LPM mining compared to state-of-the-art techniques, while at the same time mining higher quality LPMs. Additionally, by mining LPMs from subtraces, we can obtain a more structured and meaningful representation of subprocesses allowing for classic process-flow constructs such as parallel ordering, choices, and loops, besides the precedence relations shown by subtraces.

1 Introduction

Process Mining [1] has emerged as a new research area that aims at business process improvement through the analysis of event logs recorded by information systems. Such event logs capture the different steps (events) that are recorded for each instance of the process (case), and record for each of those steps what was done, by whom, for whom, where, when, etc. *Process discovery*, one of the main area within the process mining field, is concerned with the discovery of an interpretable model from event logs able to accurately describe the process. The process models obtained give insight into what is happening in the process and can be used as a starting point for follow-up analysis, e.g., bottleneck analysis [23], and checking compliance with rules and regulations [24]. Many process discovery algorithms have been developed throughout the years, e.g., [3, 13, 17, 18, 20].

In recent years, the scope of process discovery has broadened to new application domains, including software analysis and human behavior analysis. In some of those new application domains, including human behavior analysis, logs recording observed behavior have a *high degree of variability*. Often, this log variability has a significant impact on the

* This work has been partially funded by the ITEA2 project M2MGrids (No. 13011).

generation of insightful models. In particular, traditional process discovery techniques tend to generate process models in which any sequence of events is allowed, often referred to as the *flower model*. Several techniques aim to address this challenge of analyzing highly variable event logs:

Declarative process discovery (e.g., [20, 25]) mines binary relations between activities of the process.

Local Process Model (LPM) mining (e.g., [27, 28]) mines a collection of process models instead of a single model, where each process model aims to model only a subset of the behavior.

Subtrace mining (e.g., [4, 10, 16]) mines subtraces that represent relevant sequential portions of process executions (i.e., subprocesses). Some techniques (e.g., [10]) have investigated the mining of *hierarchical* subtraces. Those techniques first infer the most relevant subtraces from the event log, and then they arrange subtraces in a hierarchical structure, according to inclusion relationships.

Our primary focus is on subtrace mining and LPM mining. These techniques share similar goals, i.e., the mining of relevant process execution patterns. However, they provide different insights on the process and have their advantages and disadvantages.

Subtrace mining techniques derive the set of activities that are frequently performed together, also describing in which order they are typically performed. Hierarchical subtrace mining enables the analysis of process behaviors at different levels of abstraction. Here, subtraces at the top of the hierarchy represent the most general process behaviors, i.e., the core subtraces starting from which all the other, more detailed, subtraces can be obtained by adding one or more activities. Most subtrace mining approaches, however, are only able to derive sequential subprocesses. A more structured output, involving classic process flow constructs, usually provides more meaningful insights of the process behaviors. An exception is represented by the approach in [10], which however requires either to have a priori knowledge on concurrency or to be able to infer concurrency from event logs, for instance by applying process discovery techniques. As discussed above, mining accurate process models, including concurrent relations, is particularly challenging for low-structured and highly variable processes. Moreover, even when concurrency can be derived, other process complex behaviors, like choice constructs or loops, are not considered by subtrace mining techniques.

Local models obtained with LPM mining can model portions of process behaviors, including sequential behaviors, concurrency, loops and choice construct. However, mining LPM patterns is computationally expensive, or even infeasible, for event logs with many activities. In practice, computational problems can already arise at seventeen activities [27]. Therefore, a set of heuristics have been proposed in [27] to speed up the mining process. These heuristics aim to discover subsets of process activities (called *projections*) that are strongly related by applying the LPM miner on each projection individually and aggregating the results by taking the union of the obtained local models. The downside of these heuristics is the loss of the formal guarantee that all process models that meet a certain support threshold will be found. The quality of the results is determined by the quality of the projections.

In this work, we explore the combined use of LPM mining and hierarchical subtrace mining to mine local processes. More precisely, we investigate the application of the subtraces inferred by the technique proposed in [10] as input for LPM mining, i.e., as

projections for the LPM mining. In addition, we can exploit the ordering defined by subtraces on the execution of activities to define *ordering constraints* on the local process models. We conjecture that using activities from these subtraces as projections and ordering constraints can speed up the LPM mining procedure. Moreover, applying LPM mining on the output of subtrace mining makes it possible to infer rich control-flow relations between activities in the mined subtraces. Hierarchical subtrace mining can mine hierarchical relations between patterns, which is computationally infeasible with LPMs [22]. Therefore, by combining subtrace mining and LPM mining, an analyst can explore process behaviors that are modeled in terms of classic process constructs at different levels of abstraction, which would be otherwise difficult to obtain by LPM and subtraces mining considered separately. To assess the effectiveness of our approach, we performed an evaluation using three real-life event logs. The results show how the use of subtraces to infer projections and ordering constraints for LPM mining actually leads to significant improvements in terms of both speed of the mining and quality of the obtained local processes.

This paper is organized as follows. Section 2 introduces notation and basic concepts that are used throughout the paper. Section 3 presents LPM and hierarchical subtrace mining. Section 4 introduces a projection method and a constraint generation technique for local process model mining that is based on subtrace mining results. In Section 5 we evaluate the speedup obtained with the projections and constraints from subtraces and the quality of the obtained results on a collection of real-life event logs. Finally, Section 6 discusses related work, and Section 7 concludes the paper.

2 Event Data and Process Models

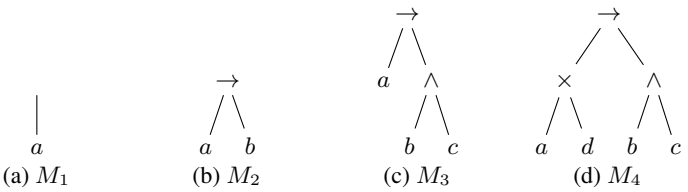
Process models describe how processes should be carried out. A process model notation that is commonly used in process mining is *process tree* [6]. A process tree is a tree structure where leaf nodes represent process activities. Non-leaf nodes represent *operators* that specify the allowed behavior over the activity nodes. Allowed operator nodes are the *sequence* operator ($\rightarrow$), which indicates that the first child is executed before the second, the *exclusive choice* operator ($\times$), which indicates that exactly one of the children can be executed, the *concurrency* operator ($\wedge$), which indicates that every child will be executed but allows for any ordering, and the *loop* operator ($\circlearrowleft$), which has one child node and allows for repeated execution of this node.

We formally define process trees recursively. Let Σ be the set of all process activities, $OP = \{\rightarrow, \times, \wedge, \circlearrowleft\}$ a set of operators and symbol $\tau \notin \Sigma$ denote silent activities. We define a process tree pt as follows:

- $a \in \Sigma \cup \{\tau\}$ is a process tree M;
- let $\{M_1, M_2, \ldots, M_n\}$ be a set of process trees. Then $\oplus(M_1, M_2, \ldots, M_n)$ with $\oplus \in OP$ is a process tree.

Hereafter, $\mathfrak{L}(M)$ denotes the language of a process tree M, i.e., the set of activity execution paths allowed by the model. Fig. 1d shows an example process tree M_4, with $\mathfrak{L}(M_4)$=$\{\langle a, b, c\rangle, \langle a, c, b\rangle, \langle d, b, c\rangle, \langle d, c, b\rangle\}$. Informally, it indicates that either activity a or d is executed first, followed by the execution of activities b and c in any order.

Process discovery aims to mine a process model from past process executions. An *event* e is the actual recording of the occurrence of an activity in Σ. A *trace* σ is a



(a) M_1 (b) M_2 (c) M_3 (d) M_4

Fig. 1: An initial LPM (M_1) and three LPMs built from successive expansions.

sequence of events, i.e., $\sigma = \langle e_1, e_2, \ldots, e_n \rangle \in \Sigma^*$. An *event log* $L \in \mathbb{N}^{\Sigma^*}$ is a finite multiset of traces. For example, event log $L = [\langle a, b, c \rangle^2, \langle b, a, c \rangle^3]$ consists of two occurrences of trace $\langle a, b, c \rangle$ and three occurrences of trace $\langle b, a, c \rangle$. $L\restriction_X$ represents the projection of log L on a subset of the activities $X \subseteq \Sigma$, e.g., $L\restriction_{\{b,c\}} = [\langle b, c \rangle^5]$. For an event log L, $\tilde{L} = \{\sigma \in \Sigma^* | L(\sigma) > 0\}$ is the *trace set* of L. For example, for event log $L = [\langle a, b, c \rangle^2, \langle b, a, c \rangle^3]$, $\tilde{L} = \{\langle a, b, c \rangle \langle b, a, c \rangle\}$. $\#(\sigma, L)$ denotes the frequency of sequence $\sigma \in \Sigma^*$ as a subtrace within log L, e.g., $\#(\langle a, b \rangle, [\langle a, b, c \rangle^2, \langle a, b, d \rangle^3]) = 5$.

3 Subprocess Mining

In this section, we provide an overview of LPM mining and hierarchial subtrace mining.

3.1 Local Process Models

Local Process Models (LPM) [28] are process models that describe frequent but partial behaviors seen in the event log, i.e., they model subsets of activities of the process. In this work, we use the process tree notation to represent LPMs.

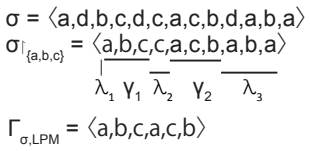
A technique to generate a ranked collection of LPMs through iterative expansion of candidate process trees is proposed in [28]. This technique involves four steps: 1) the *generation* of an initial (candidate) set of process trees, where a process tree is created for each single activity of the projection; 2) the *evaluation* phase, where process tree quality is assessed by a set of tailored metrics; 3) the *selection* phase, where process trees that scored poorly at the previous step are removed from the candidate set; 4) the *expansion* phase, where candidates selected at the previous step are expanded by replacing an activity node by an operator node, whose children are the replaced activity and another activity of the process. These steps are repeated until no candidate in the candidate set has the required support.

The *expansion procedure* consists of the replacement of a leaf activity node a in the process tree by an operator node (i.e., $\rightarrow$, $\times$, $\wedge$ or $\circlearrowleft$), where one of the child nodes is the replaced activity node a and the other is a new activity node $b \in \Sigma$. We denote $\mathcal{M}$ the LPM universe, i.e., the set of all possible LPMs. An LPM $M \in \mathcal{M}$ can be expanded in many ways, as it can be extended by replacing any one of its activity nodes, expanding it with any of the operator nodes, and with a new activity node that represents any of the activities in the event log. We define $Exp(M)$ as the set of expansions of M, and exp_max the maximum number of expansions allowed from an *initial LPM*, i.e., an LPM containing only one activity.

Fig. 1 provides an example of the expansion procedure, starting from the initial LPM M_1 of Fig. 1a. The LPM of Fig. 1a is first expanded into a larger LPM by replacing a by

event id	activity	time
1	a	15-4-2016 12:23
2	d	16-4-2016 14:38
3	b	16-4-2016 14:46
4	c	16-4-2016 15:46
5	d	16-4-2016 16:53
6	c	16-4-2016 16:58
7	a	16-4-2016 17:11
8	c	16-4-2016 17:45
9	b	16-4-2016 18:03
10	d	17-4-2016 12:09
11	a	17-4-2016 18:24
12	b	17-4-2016 18:36
13	a	17-4-2016 18:37

(a) A trace σ of an event log L



(b) Segmentation of σ on M_3

Fig. 2: Example of segmentation in LPM mining.

operator node $\rightarrow$, with activity a as its left child node and b its right child node, resulting in the LPM of Fig. 1b. Note that M_1 can also be expanded using any other operator or any other activity from Σ, and LPM discovery recursively explores all possible process trees that meet a support threshold by iterative expansion. In a second expansion step, activity node b of the LPM of Fig. 1b is replaced by operator node $\wedge$, with activity b as its left child and c its right child, resulting in the LPM of Fig. 1c. Finally, activity node a of the LPM of Fig. 1c is replaced by operator node $\times$ with as left child activity a and as right child activity d, forming the LPM of Fig. 1d. In traditional LPM discovery the expansion procedure of an LPM stops when the behavior described by the LPM is not observed frequently enough in an event log L (i.e., with regard to some *support threshold*).

To evaluate a given LPM on a given event log L, its traces $\sigma \in L$ are first projected on the set of activities X in the LPM, i.e. $\sigma' = \sigma{\restriction_X}$. The projected trace σ' is then segmented into γ-segments, i.e., segments that fit the behavior of the LPM, and λ-segments, i.e. segments that do not fit the behavior of the LPM. Specifically, $\sigma' = \lambda_1\gamma_1\lambda_2\gamma_2 \cdots \lambda_n\gamma_n\lambda_{n+1}$ such that $\gamma_i \in \mathfrak{L}(LPM)$ and $\lambda_i \notin \mathfrak{L}(LPM)$. We define $\Gamma_{\sigma,LPM}$ to be a function that projects trace σ on the LPM activities and obtains its subsequences that fit the LPM, i.e. $\Gamma_{\sigma,LPM} = \gamma_1\gamma_2 \ldots \gamma_n$.

Let our LPM M_3 under evaluation be the process tree of Fig. 1c and σ the example trace shown in Fig. 2a. Function $Act(LPM)$ gives the set of process activities in the LPM, e.g. $Act(M_3) = \{a, b, c\}$. The projection on the activities of the LPM gives $\sigma{\restriction_{Act(M_3)}} = \langle a, b, c, c, a, c, b, a, b, a\rangle$. Fig. 2b shows the segmentation of the projected trace on the LPM, leading to $\Gamma_{\sigma,LPM} = \langle a, b, c, a, c, b\rangle$. The segmentation starts with an empty non-fitting segment λ_1, followed by a fitting segment $\gamma_1 = \langle a, b, c\rangle$, which completes one run through the process tree. The second event c in σ cannot be replayed on LPM, since it only allows for one c and γ_1 already contains a c. This results in a non-fitting segment $\lambda_2 = \langle c\rangle$. Segment $\gamma_2 = \langle a, c, b\rangle$ again represents a run through process tree; the segmentation ends with non-fitting segment $\lambda_3 = \langle a, b, a\rangle$. We lift segmentation function Γ to event logs, $\Gamma_{L,LPM} = \{\Gamma_{\sigma,LPM} | \sigma \in L\}$. An alignment-based [2] implementation of Γ, as well as a method to rank and select LPMs based on their support, i.e., the number of events in $\Gamma_{L,LPM}$, is described in [28].

Local Process Models (LPMs) only contain a subset of the activities of the log and each LPM can in principle be discovered on any projection of the log containing the activities used in this LPM. Since the computational complexity of LPM mining depends

combinatorially on the number of activities in the log, mining LPMs on projections of the log instead of on the full log can significantly lower the running time of LPM mining. However, this results in a partial exploration of the LPM search space, potentially leading to good LPMs not being found. In principle, when activities that often follow each other are in the same projection, the search space can be constrained almost without loss in quality of the mined LPMs. Such projection sets could potentially be overlapping, which is actually desired, since interesting patterns might exist within a set of activities $\{a, b, c, d\}$, as well as within a set of activities $\{a, b, c, e\}$, and discovering on both $L \restriction_{\{a,b,c,d\}}$ and $L \restriction_{\{a,b,c,e\}}$ and merging the results is faster than discovering on the full log with $\Sigma = \{a, b, c, d, e\}$. Markov clustering, a well-known graph clustering algorithm, is proposed as a method to generate projection sets in [27]. Markov clustering is applied to a graph where vertices represent activities and edges represent the *connectedness* of two activities, using directly-follows and directly-precedes ratios. It defines *connectedness* between activities a and b in $\log L$ as $conn(a, b, L) = \sqrt{\frac{\#(\langle a,b\rangle, L)}{\#(\langle a\rangle, L)}^2 + \frac{\#(\langle a,b\rangle, L)}{\#(\langle b\rangle, L)}^2}$.

3.2 Hierarchical Subtrace Mining

The approach presented in [10] proposes to apply *frequent subgraph mining* (FSM) algorithms to derive most relevant subprocesses. To this end, each trace $\sigma \in L$ is transformed into a directed graph $g = (V, E, \phi)$, where V is the set of nodes, each corresponding to an event in σ, E is the set of the edges, showing ordering relations between the events, and ϕ is a labeling function associating each node with the activity of the corresponding event. A trivial transformation consists in creating a node for each event in the trace and linking each pair of subsequent events through an edge. By doing so, every log trace is transformed into a sequence of events ordered according to their order in the log trace.

Once the set of graphs is obtained, an FSM algorithm is applied to derive frequent subgraphs (i.e., subtraces). FSM algorithms usually rely on a metric to evaluate the *relevance* of the subgraphs. The work in [10] uses the SUBDUE algorithm [15] that employs Description Length (DL) to assess the relevance of subgraphs. Given a graph set G and a subgraph s, SUBDUE uses an index based on DL, hereafter denoted by $\nu(s, G)$, which is computed as $\nu(s, G) = \frac{DL(G)}{DL(s)+DL(G|s)}$ where $DL(G)$ is the DL of G, $DL(s)$ is the DL of s and $DL(G|s)$ is the DL of G *compressed* using s (i.e., the graph set in which each occurrence of s is replaced with a single node). By doing so, SUBDUE relates the relevance of a subgraph with its compression capability. The higher the value of ν is, the lower the DL value of the graph dataset compressed by s is.

SUBDUE works iteratively. At each step, it extracts the subgraph with the highest compression capability, i.e., the subgraph corresponding to the maximum value of the ν index. This subgraph is then used to compress the graph set. The compressed graphs are presented to SUBDUE again. These steps are repeated until no more compression is possible or until a user-defined number of iterations is reached. The outcome of SUBDUE is a hierarchical structure, where mined subgraphs are ordered according to their relevance, showing the existing inclusion relationships among them. Top-level subgraphs are defined only through elements that belong to input graphs (i.e., nodes and arcs), while lower-level subgraphs additionally contain upper-level subgraphs as nodes. Descending the hierarchy,

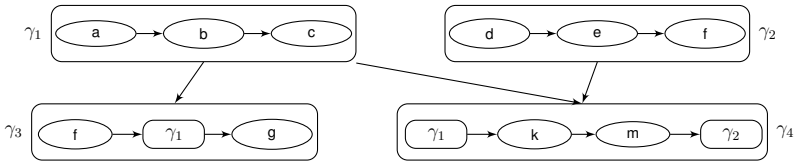
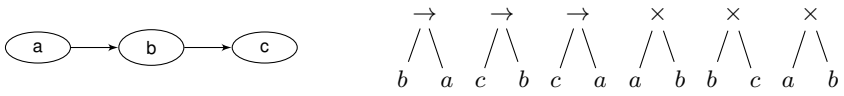


Fig. 3: Example of the SUBDUE hierarchy



(a) Subtrace (b) Ordering constraints

Fig. 4: An example of subtrace and the corresponding ordering constraints.

we pass from subgraphs that are very common in the input graph set to subgraphs occurring in a few input graphs. Therefore, we can reasonably expect to be able to capture most of the process behaviors by only considering top-level subgraphs.

Fig. 3 shows a simple example of the hierarchical structure mined by SUBDUE. We have two subgraphs at the top level, i.e., γ_1 and γ_2, together with two subgraphs at the lower level, i.e., γ_3 and γ_4. We can see that γ_3 is a child of γ_1, since it involves γ_1 in its definition. Similarly, γ_4 is a child of both γ_1 and γ_2.

4 Approach

In this section, we present an approach to mine local process models by exploiting subtrace mining. In particular, our approach extends an existing LPM mining algorithm [28] by converting the set of subgraphs mined using SUBDUE into a set of *projections* (i.e., set of activities) and a set of *ordering constraints*. Projections and constraints are both used in the expansion phase of the LPM mining algorithm. Projections are used to reduce the LPM mining search space by reducing the number of activities that can be used for expansion, while ordering constraints prohibit expansion into LPMs that do not meet certain ordering relations.

Ordering constraints represent impossible (or, at least, very unlikely) execution orders between activities of the subgraphs. Two types of constraints can be derived from subtraces: constraints on *exclusive choices* and constraints on *sequential executions*. Given a subgraph $s = (V, E, \phi)$, an ordering constraint on s is defined as a process tree $M = \oplus(a, b)$ such that $a, b \in V_i$ and $\oplus \in \{\rightarrow, \times\}$. Fig. 4 provides an example of a set of process trees representing ordering constraints mined from a subtrace.

These constraints allow us to remove from the space state those LPMs that we can reasonably expect to turn out to be poor quality models and, hence, would be discarded in any case. It is trivial to see that we can safely remove from the state space all process trees involving choices constructs among activities belonging to the same subgraph. Indeed, the fact that these activities belong to the same subgraph implies that they tend to occur together, which, in turn, excludes that they can be properly represented by a choice construct. Similarly, since subgraphs represent the most frequent ordering executions obtained for a

Algorithm 1: Subgraphs-based LPM

Input : event log L, set of subgraphs S
Output : set of local process models LPM

1 $LPM = \emptyset$;
2 $CP = \emptyset$;
3 **foreach** $s_i = (V_i, E_i, \phi_i) \in S$ **do**
4 $\quad OC_i = findOrderingConstraints(s_i)$
5 $\quad CP = CP \cup \{(V_i, OC_i)\}$;
6 $Checked\_p = \emptyset$
7 $CP' = \emptyset$
8 **foreach** $p_i = (V_i, OC_i) \in CP$ **do**
9 $\quad p'_i = p_i$
10 $\quad$ **if** $p_i \notin Checked\_p$ **then**
11 $\quad\quad$ **foreach** $p_j = (V_j, OC_j) \in CP$ **do**
12 $\quad\quad\quad$ **if** $p_j \notin Checked\_p \wedge j \neq i$ **then**
13 $\quad\quad\quad\quad$ **if** $(V_i \subseteq V_j \vee V_j \subseteq V_i)$ **then**
14 $\quad\quad\quad\quad\quad p'_i = (V_i \cup V_j, OC_i \cup OC_j)$
15 $\quad\quad\quad\quad\quad Checked\_p = Checked\_p \cup \{p_j\}$
16 $\quad Checked\_p = Checked\_p \cup \{p_i\}$
17 $\quad CP' = CP' \cup \{p'_i\}$
18 **foreach** $p_i \in CP'$ **do**
19 $\quad LPM = LPM \cup MiningLPMwithConstraints(L, p_i)$
20 **return** LPM

given set of activities, it is reasonable to not investigate those paths that explicitly contradict them when mining LPMs. For instance, the subprocess in Fig. 4a shows that in most of the cases activity *a* occurred before *b*, which, in turn, mostly occurred before *c* (which means that *a* occurred before *c* as well). Therefore, from this subgraph, we obtain the three sequential ordering constraints shown in Fig. 4b.

Algorithm 1 formalizes the approach. Given an event log L and a set of subgraphs S, ordering constraints are derived from each subgraph by invoking *findOrderingConstraints* (line 4), defined in Algorithm 2. Then, the algorithm generates the set of projections and constraints CP. Each element of CP represents a projection V_i together with the corresponding ordering constraints OC_i mined from subgraph s_i. After an element of CP has been created for each subgraph, the algorithm merges those elements whose sets of activities are identical or included among each other. Namely, it generates a new element p' whose projection corresponds to the union of their projections, and the set of ordering constraints corresponds to the union of their sets of ordering constraints. (lines 8-17). By merging them in a single projection, we ensure that the same set of activities are considered only once by LPM mining.

Combining ordering constraints can help reduce the number of process trees to explore. For example, let us assume that SUBDUE finds two subgraphs s_1, s_2 involving activities *a, b* but reported in opposite execution order such that $CP_{s_1} = (\{a, b\}, \{\rightarrow (b, a), \times(a, b)\})$ and $CP_{s_2} = (\{a, b\}, \{\rightarrow (a, b), \times(a, b)\})$. By merging these elements, we obtain $CP' = (\{a, b\}, \{\rightarrow (b, a), \rightarrow (a, b) \times (a, b)\})$. This means that in the expansion phase of LPM

Algorithm 2: FindOrderingConstraints

Input : subgraph $s = (V, E, \phi)$
Output : set of ordering constraints OC

1 $OC = \emptyset$;
2 **foreach** $v_i \in V$ **do**
3 **foreach** $v_j \in V$ **do**
4 $M_{xor} = \times(v_i, v_j)$;
5 $OC = OC \cup \{M_{xor}\}$;
6 $\rho(v_i, v_j) = extractPath(v_i, v_j)$
7 **if** $\rho(v_i, v_j) \neq \emptyset$ **then**
8 $M_{seq} = \rightarrow (v_j, v_i)$
9 $OC = OC \cup \{M_{seq}\}$;
10 **return** OC

mining, whenever a node involving *a* (*b*) has to be expanded adding another node *b* (*a*), only the parallel operator would be considered for the expansion. In fact, the expansions involving both the sequential operator and the choice operator are forbidden by the ordering constraints.

Finally, algorithm *MiningLPMwithConstraints* is invoked on the event log and the mined set of constraints (line 19). This algorithm is a refined version of the procedure described in Section 3.1, able to account for constraints during the expansion phase. More precisely, every time that a leaf activity node in the process tree M to expand is replaced by an operator node, the obtained process tree $M_i \in Exp(M)$ is checked against the set of ordering constraints OC_i. If there exists a constraint $oc_i \in OC_i$ such that oc_i is a subtree of M_i, then M_i is removed from the set of process trees that are considered for the following expansions.

Algorithm 2 describes the procedure to convert a subgraph into a set of ordering constraints. For each pair of activities in the subgraph, the algorithm generates a constraint representing an exclusive choice and updates the set of ordering constraints (lines 4 and 5). Then, the algorithm checks whether those activities are linked by a path (line 6). If a path exists, the algorithm generates a sequential constraint representing a sequential execution order that violates the detected path is added to the set of constraints (lines 7-9). The algorithm returns the set of inferred ordering constraints.

The hierarchical structure returned by SUBDUE can be exploited to determine which subgraphs have to be considered during the LPM mining. As explained in Section 3.2, low-levels subgraphs in the SUBDUE hierarchy correspond to detailed and often infrequent process variants. In this work, we let process analysts determine the level of detail to be used for LPM mining, i.e., the level of the hierarchy from which subgraphs can be selected.

5 Experiments

In this section, we explore the effect of exploiting subtrace mining to mine Local Process Models (LPM). To this end, we performed two sets of experiments. In the first, we investigated the effects of the only use of SUBDUE projections. Namely, we used the LPM mining procedure of [27] adopting the activities involved in SUBDUE subgraphs as

projections, thus neglecting their ordering. In the second set of experiments, we exploited both projections and ordering constraints, using the LPM mining procedure in Section 4.

We evaluated the LPM mining on two dimensions. First, we considered the reduction in the search space size, indicating the potential gain in computation time of the mining procedure. Then, we considered the quality of the output by comparing the ranking of LPMs mined using SUBDUE projections to the ranking of LPMs mined by exploring the full search space. When projections are exploited, some LPMs that are mined when exploring the full LPM search space might not be found [27]. Hence, by comparing the ranking of the models mined with/without projections, we can evaluate to what extent the use of projections affected the obtained outcome. We used the *Normalized Discounted Cumulative Gain* (NDCG) [7, 14], which is a widely used metrics to compare an obtained ranking with a ground truth ranking in the field of information retrieval [26]. Generally, NDCG@k is used, which only considers the top k elements of the ranking. NDCG consists of two components, *Discounted Cumulative Gain* (DCG) and *Ideal Discounted Cumulative Gain* (IDCG). DCG aggregates the relevance scores (i.e., the score obtained with respect to the quality metrics) of individual LPMs in the ranking in such a way that the graded relevance is discounted with logarithmic proportion to their position in the ranking. This results in more weight being put on the top of the ranking compared to lower parts of the ranking. Formally, DCG is defined as: $DCG@k = \sum_{i=1}^{k} \frac{2^{rel_i}-1}{log_2(i+1)}$, where rel_i is the relevance of the LPM at position i. Normalized Discounted Cumulative Gain (NDCG) is obtained by dividing the DCG value by the DCG on the ground truth ranking (called Ideal Discounted Cumulative Gain). Normalized Discounted Cumulative Gain (NDCG) is defined as: $NDCG@k = \frac{DCG@k}{IDCG@k}$.

As a baseline technique, we applied the Markov-based projection discovery technique presented in [27] iteratively until all projections contain at most seven activities. We compared the search space reduction and the NDCG obtained when mining with iterative Markov-based projections with the search space reduction and NDCG obtained when mining with projections and constraints extracted from SUBDUE subgraphs.

Dataset We evaluated our technique using three real-life event logs. The first event log contains execution traces from a financial loan application process at a large Dutch financial institution, commonly referred to as the *BPI 2012* log [11]. This log consists of 13087 traces (loan applications) for which a total of 164506 events have been executed, divided over 23 activities. The second event log contains traces from the receipt phase of an environmental permit application process at a Dutch municipality, to which we will refer as the *receipt phase WABO* log [5]. The receipt phase WABO log contains 1434 traces, 8577 events, and 27 activities. The third event log contains medical care pathways of sepsis patients from a medium size hospital, to which we will refer as the *SEPSIS* log [21]. The SEPSIS log contains 1050 traces, 15214 events, and 16 activities.

Settings We use the implementation of the iterative Markov LPM mining algorithm available in the *LocalProcessModelDiscovery* package[1] of the ProM framework [29]. For the LPM mining approach that uses SUBDUE projections and constraints, we made an implementation available in ProM package *LocalProcessModelDiscoveryWithSubdueCon-*

[1] https://svn.win.tue.nl/repos/prom/Packages/LocalProcessModelDiscovery/

straints[2]. Both for Markov-based LPM mining and SUBDUE-based LPM mining we use the standard configuration in ProM. For SUBDUE we used the implementation provided in http://ailab.wsu.edu/subdue/, varying the number of iterations. Note that, roughly speaking, a high number of SUBDUE iterations is expected to have benefits in terms of quality of LPM mining. Higher numbers of iterations lead to a higher number of process behaviors captured by SUBDUE subgraphs and, thus, taken into account during LPM mining. However, this has a negative impact on the speedup of the LPM mining procedure. Moreover, it is worth noting that, by construction, SUBDUE extracts the largest among the more frequent subgraphs in the earlier iterations. Hence, in practice, we expect the subtraces obtained using SUBDUE even for a low number of iterations to be able to represent most of the process behaviors. According to this observation, we used 1, 5 and 10 iterations for our experiments. Nonetheless, to verify our assumption, we also performed for each dataset an experiment with a high number of iterations (i.e., 10000). Note that for these experiments we derived the LPMs only for the subgraphs of the top-level of SUBDUE hierarchy, since they represent the most relevant ones, as explained in Section 3. All the experiments were performed on an Intel i7 CPU.

Results Table 1 shows the results obtained on the three event logs. The results obtained when exploring the full search space, i.e., without applying any projections or constraints, are considered to be the ground truth LPM ranking and therefore have NDCG@k values of 1.0 by definition. For the BPI 2012 log and the receipt phase log the full search space consists of approximate 1.5 million LPMs, while for the smaller SEPSIS log the search space consists of 315k LPMs. The results obtained by the best heuristic configuration(s) are reported in bold.

When using projections mined with iterative Markov [27] projection mining, the search space of LPM mining is reduced by a factor between 43.50x (BPI 2012 log) and 120.21x (Receipt phase log), while the majority of the top 20 LPMs of the ground truth can still be found, resulting in high NDCG values. The table shows that using the constraints extracted from SUBDUE subgraphs after running the SUBDUE algorithm for 10k iterations together with the iterative Markov projections further increases the speedup of LPM mining on all three logs while resulting in identical LPM rankings.

The number of iterations used in the SUBDUE algorithms when using projections extracted from SUBDUE results is indicated between parentheses. The search space size of LPM mining when using SUBDUE projections depends on the number of iterations used in the SUBDUE mining phase: when a larger number of iterations is used, then a larger number of unique sets of activities will be found to be used a projections, leading to a larger LPM search space. At the same time, increasing the number of iterations used in the SUBDUE mining phase increases the quality of the mined LPMs in terms of NDCG. Notice that the quality of the LPM mining results is not very stable when we run the SUBDUE algorithm for only one iteration: on the receipt phase log and the BPI 2012 log it leads to results comparable with or better than the results obtained with iterative Markov projections, but on the SEPSIS log it results in a very small search space and NDCG values. It is worth noting that running the SUBDUE algorithm for 10 iterations leads to a higher speedup than iterative Markov projections on all logs, even when no SUBDUE constraints are used, while at the same time the quality of the obtained rankings in terms of NDCG is higher. This

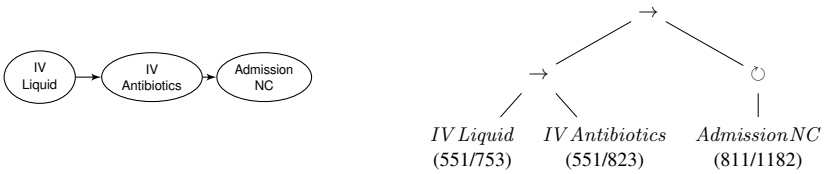
[2] https://svn.win.tue.nl/repos/prom/Packages/LocalProcessModelDiscoveryWithSubdueConstraints

Table 1: Experimental results of LPM mining with and without SUBDUE projections and constraints in terms of search space size and quality of the ranking.

Event Log	**Projections** (iterations)	**Constraints** (iterations)	**Search Space Size**	**Speedup**	**NDCG@5**	**NDCG@10**	**NDCG@20**
BPI 2012	None	None	1567250	-	1.0000	1.0000	1.0000
BPI 2012	Iterative Markov	None	36032	43.50x	0.9993	0.9987	0.9865
BPI 2012	Iterative Markov	SUBDUE (10k)	21084	74.33x	0.9993	0.9987	0.9865
BPI 2012	SUBDUE (1)	None	10608	147.74x	0.9993	0.9987	0.9830
BPI 2012	SUBDUE (5)	None	10740	145.93x	**1.0000**	**0.9994**	0.9870
BPI 2012	SUBDUE (10)	None	10904	143.73x	**1.0000**	**0.9994**	**0.9903**
BPI 2012	SUBDUE (10k)	None	12666	123.74x	**1.0000**	**0.9994**	**0.9903**
BPI 2012	SUBDUE (1)	SUBDUE (1)	2718	576.62x	0.9993	0.9987	0.9830
BPI 2012	SUBDUE (5)	SUBDUE (5)	**2620**	**598.19x**	**1.0000**	**0.9994**	0.9870
BPI 2012	SUBDUE (10)	SUBDUE (10)	2874	545.32x	**1.0000**	**0.9994**	**0.9903**
BPI 2012	SUBDUE (10k)	SUBDUE (10k)	4012	390.64x	**1.0000**	**0.9994**	**0.9903**
Receipt phase	None	None	1451450	-	1.0000	1.0000	1.0000
Receipt phase	Iterative Markov	None	12074	120.21x	0.9418	0.8986	0.8238
Receipt phase	Iterative Markov	SUBDUE (10k)	10610	136.80x	0.9418	0.8986	0.8238
Receipt phase	SUBDUE (1)	None	8176	177.53x	**1.0000**	**0.9994**	0.9903
Receipt phase	SUBDUE (5)	None	8256	175.81x	**1.0000**	**0.9994**	0.9903
Receipt phase	SUBDUE (10)	None	8264	175.64x	**1.0000**	**0.9994**	**0.9958**
Receipt phase	SUBDUE (10k)	None	8504	170.68x	**1.0000**	**0.9994**	**0.9958**
Receipt phase	SUBDUE (1)	SUBDUE (1)	4012	390.64x	**1.0000**	**0.9994**	0.9903
Receipt phase	SUBDUE (5)	SUBDUE (5)	**1862**	**779.51x**	**1.0000**	**0.9994**	0.9903
Receipt phase	SUBDUE (10)	SUBDUE (10)	2170	668.71x	**1.0000**	**0.9994**	**0.9958**
Receipt phase	SUBDUE (10k)	SUBDUE (10k)	2178	666.41x	**1.0000**	**0.9994**	**0.9958**
SEPSIS	None	None	315451	-	1.0000	1.0000	1.0000
SEPSIS	Iterative Markov	None	6304	50.04x	0.9332	0.9148	0.8613
SEPSIS	Iterative Markov	SUBDUE (10k)	3768	83.72x	0.9332	0.9148	0.8613
SEPSIS	SUBDUE (1)	None	12	26287.58x	0.5763	0.3771	0.2489
SEPSIS	SUBDUE (5)	None	334	994.46x	0.9916	0.9671	0.9472
SEPSIS	SUBDUE (10)	None	394	800.64x	**0.9923**	**0.9692**	**0.9534**
SEPSIS	SUBDUE (10k)	None	1034	305.08x	**0.9923**	**0.9692**	**0.9534**
SEPSIS	SUBDUE (1)	SUBDUE (1)	**10**	**31545.10x**	0.5763	0.3771	0.2489
SEPSIS	SUBDUE (5)	SUBDUE (5)	174	1812.94x	0.9916	0.9671	0.9472
SEPSIS	SUBDUE (10)	SUBDUE (10)	144	2190.63x	**0.9923**	**0.9692**	**0.9534**
SEPSIS	SUBDUE (10k)	SUBDUE (10k)	470	671.17x	**0.9923**	**0.9692**	**0.9534**

seems to indicate that using the activity sets of SUBDUE subgraphs is a more effective approach to mine clusters of activities for projection based LPM mining than the iterative Markov based approach. No increase in the quality of the mined LPMs has been found by increasing the number of SUBDUE iterations over 10 iterations; on all three event logs the quality when using 10k iterations is identical to 10 iterations.

Using constraints extracted from SUBDUE subtrace mining in combination with SUBDUE-based projections has resulted in considerably higher speedup on all three logs without any loss in quality of the mined LPMs. On all event logs, the highest quality LPMs are mined when using projections and the constraints extracted from the subsequence mined with at least 10 iterations of the SUBDUE algorithm. Additionally, as expected, the speedup obtained using these projections and constraints is considerably higher than the speedup obtained when mining LPMs without constraints and with iterative Markov projections, which was the current state-of-the-art approach in LPM mining. Compared to LPM mining



(a) Subgraph mined with SUBDUE (b) Local Process Model

Fig. 5: Subgraph mined with SUBDUE from the SEPSIS log and the Local Process Model showing the true relation between activities *IV Liquid*, *IV Antibiotics*, and *Admission NC*.

with iterative Markov projections, mining with SUBDUE (10) projections and constraints results in an increase in speedup from 43.50x to 545.32x on the BPI 2012 log, from 120.21x to 668.71x on the receipt phase log, and from 50.04x to 6190.63x on the SEPSIS log. To put these results into perspective: this brings down the mining time on the BPI 2012 log from 24 minutes to less than two minutes.

Before concluding this section, we provide an example of LPM mined from a SUBDUE subgraph, to provide some insights on the benefits of introducing a higher level of structure on subtrace mining output. Fig. 5a shows the subtrace mined with SUBDUE from the SEPSIS log from which the projection on activity set $\{IV\,Liquid, IV\,Antibiotics, AdmissionNC\}$ was extracted and Fig. 5b shows one of the LPM mined from that set of activities. While SUBDUE can only mine sequential relations, as in Fig. 5a, the LPM shows the true relation between the three activities: 551 out of 753 occurrences of *IV Liquid* are followed by *IV Antibiotics* and then one or more occurrences of *Admission NC*. However, 811 out of 1182 instances of *Admission NC* are preceded by the sequence $\langle IV\,Liquid, IV\,Antibiotics\rangle$, showing that there are on average almost 1.5 admissions for each liquid and antibiotics.

6 Related Work

Several approaches have been proposed to extract the most relevant subprocesses (intended as subgraphs) from a set of process execution traces. Some approaches propose to extract subprocesses from sequential traces. For instance, Bose et al. [4] mine subprocesses by identifying sequences of events that fit a priori defined templates. Leemans et al. [16] introduce an approach to derive *episodes*, i.e., directed graphs where nodes correspond to activities and edges to *eventually-follow* precedence relations that, given a pair of activities, indicate which one occurs later in the process. Compared to these approaches, our approach does not require any predefined template and extracts subprocesses that are the most relevant according to their description length, thus taking into account both frequency and size in determining the relevance of each subprocess.

Several other techniques, like LPMs, focus on the mining of more complex patterns that allow for control-flow constructs other than sequential ordering. Chapela-Campa et al. [8] developed a technique called WoMine-i to mine subprocess patterns with multiple control-flow constructs that are *infrequent*, in contrast to most techniques that focus on the mining of *frequent* subprocess patterns. Lu et al. [19] recently proposed an interactive subprocess

exploration tool, which allows for the discovery of subprocess patterns and then allows the process analyst to modify these discovered subprocesses based on domain knowledge. For such a modified subprocess the tool provides the analyst with information regarding how frequent the modified pattern is, and where in the log it occurred. Dalmas et al. [9] developed a version of the Local Process Model miner, called the High-Utility LPM miner, that aims at mining subprocess patterns that optimize some optimization function that is provided by the process analyst, instead of mining simply frequent LPMs. Greco et al. [12] propose a Frequent Subgraph Mining (FSM) algorithm that exploits knowledge about relationships among activities (e.g., an AND/OR split) to drive subgraphs mining. Graphs are generated by replaying traces over the process model; however, this algorithm requires a model properly representing the event log, which may not be available for many real-world processes.

7 Conclusions and Future Work

In this work, we investigated the use of subtrace mining in the generation of projections and constraints to aid the discovery of Local Process Models (LPMs). Specifically, we extended the LPM algorithm in [27] to account for ordering constraints mined using SUBDUE. We evaluated our approach on three real-world event logs. The results show that mining LPMs with SUBDUE projections and constraints outperforms the current state-of-the-art techniques for LPM mining both in quality as well as in computation time. In particular, on two out of the three event logs, the speedup in computation time compared to the current state-of-the-art technique is more than a factor 10.

As future work, we plan to explore the use of other subtraces mining techniques to derive local process models. Moreover, we intend to investigate the mining of ordering relations between projections that are not involved in hierarchical inclusion relations, and their use in the combined set of LPMs, to obtain models describing possible correlations between behaviors occurring in different portions of the process.

References

1. van der Aalst, W.M.P.: Process mining: data science in action. Springer (2016)
2. van der Aalst, W.M.P., Adriansyah, A., van Dongen, B.F.: Replaying history on process models for conformance checking and performance analysis. Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery 2(2), 182–192 (2012)
3. Bergenthum, R., Desel, J., Lorenz, R., Mauser, S.: Process mining based on regions of languages. In: Business Process Management. pp. 375–383. Springer (2007)
4. Bose, R.P.J.C., van der Aalst, W.M.P.: Abstractions in process mining: A taxonomy of patterns. In: Business Process Management, LNCS, vol. 5701, pp. 159–175. Springer (2009)
5. Buijs, J.C.A.M.: Receipt phase of an environmental permit application process (WABO), CoSeLoG project (2014), doi:10.4121/uuid:a07386a5-7be3-4367-9535-70bc9e77dbe6
6. Buijs, J.C.A.M., van Dongen, B.F., van der Aalst, W.M.P.: A genetic algorithm for discovering process trees. In: Proc. of IEEE Congress on Evolutionary Computation. pp. 1–8. IEEE (2012)
7. Burges, C., Shaked, T., Renshaw, E., Lazier, A., Deeds, M., Hamilton, N., Hullender, G.: Learning to rank using gradient descent. In: Proceedings of International Conference on Machine Learning. pp. 89–96 (2005)

8. Chapela-Campa, D., Mucientes, M., Lama, M.: Discovering infrequent behavioral patterns in process models. In: Business Process Management. pp. 324–340. Springer (2017)
9. Dalmas, B., Tax, N., Norre, S.: Heuristics for high-utility local process model mining. In: Proceedings of the International Workshop on Algorithms & Theories for the Analysis of Event Data. pp. 106–121. CEUR (2017)
10. Diamantini, C., Genga, L., Potena, D.: Behavioral process mining for unstructured processes. Journal of Intelligent Information Systems 47(1), 5–32 (2016)
11. van Dongen, B.F.: BPI challenge 2012 (2012), doi:10.4121/uuid:3926db30-f712-4394-aebc-75976070e91f
12. Greco, G., Guzzo, A., Manco, G., Saccà, D.: Mining and reasoning on workflows. IEEE Transactions on Knowledge and Data Engineering 17(4), 519–534 (2005)
13. Günther, C.W., van der Aalst, W.M.P.: Fuzzy mining–adaptive process simplification based on multi-perspective metrics. In: Business Process Management. pp. 328–343. Springer (2007)
14. Järvelin, K., Kekäläinen, J.: Cumulated gain-based evaluation of ir techniques. ACM Transactions on Information Systems 20(4), 422–446 (2002)
15. Jonyer, I., Cook, D., Holder, L.: Graph-based Hierarchical Conceptual Clustering. Journal of Machine Learning Research 2, 19–43 (2002)
16. Leemans, M., van der Aalst, W.: Discovery of frequent episodes in event logs. In: Proc. of Int. Symposium on Data-driven Process Discovery and Analysis. pp. 1–31. CEUR-WS.org (2014)
17. Leemans, S.J.J., Fahland, D., van der Aalst, W.M.P.: Discovering block-structured process models from event logs containing infrequent behaviour. In: BPM. pp. 66–78. Springer (2013)
18. Liesaputra, V., Yongchareon, S., Chaisiri, S.: Efficient process model discovery using maximal pattern mining. In: Business Process Management. pp. 441–456. Springer (2015)
19. Lu, X., Fahland, D., Andrews, R., Suriadi, S., Wynn, M.T., ter Hofstede, A.H.M., van der Aalst, W.M.P.: Semi-supervised log pattern detection and exploration using event concurrence and contextual information. In: Proceedings of International Conference on Cooperative Information Systems. Springer (2018)
20. Maggi, F.M., Mooij, A.J., van der Aalst, W.M.P.: User-guided discovery of declarative process models. In: Computational Intelligence and Data Mining. pp. 192–199. IEEE (2011)
21. Mannhardt, F.: SEPSIS Cases – Event Log (2016), doi:10.4121/uuid:915d2bfb-7e84-49ad-a286-dc35f063a460
22. Mannhardt, F., Tax, N.: Unsupervised event abstraction using pattern abstraction and local process models. In: Proceedings of the International Working Conference on Business Process Modeling, Development and Support. pp. 89–96. CEUR-WS.org (2017)
23. Măruşter, L., van Beest, N.R.T.P.: Redesigning business processes: a methodology based on simulation and process mining techniques. Knowl. Inf. Syst. 21(3), 267 (2009)
24. Ramezani, E., Fahland, D., van der Aalst, W.M.P.: Where did I misbehave? diagnostic information in compliance checking. In: BPM. pp. 262–278. Springer (2012)
25. Schönig, S., Cabanillas, C., Jablonski, S., Mendling, J.: Mining the organisational perspective in agile business processes. In: International Conference on Enterprise, Business-Process and Information Systems Modeling. pp. 37–52. Springer (2015)
26. Tax, N., Bockting, S., Hiemstra, D.: A cross-benchmark comparison of 87 learning to rank methods. Information processing & management 51(6), 757–772 (2015)
27. Tax, N., Sidorova, N., van der Aalst, W.M.P., Haakma, R.: Heuristic approaches for generating local process models through log projections. In: Proceedings of IEEE Symposium Series on Computational Intelligence. pp. 1–8. IEEE (2016)
28. Tax, N., Sidorova, N., Haakma, R., van der Aalst, W.M.P.: Mining local process models. Journal of Innovation in Digital Ecosystems 3(2), 183–196 (2016)
29. Verbeek, H.M.W., Buijs, J.C.A., Van Dongen, B.F., van der Aalst, W.M.P.: ProM 6: The process mining toolkit. In: Proceedings of BPM Demo Track. vol. 615, pp. 34–39. CEUR-WS.org (2010)