Mining Local Process Models and Their Correlations

Laura Genga, Niek Tax, Nicola Zannone

Mining Local Process Models and their Correlations

Laura Genga (✉)[0000−0001−8746−8826], Niek Tax, and Nicola Zannone[0000−0002−9081−5996]

Eindhoven University of Technology
5600 MB Eindhoven ,The Netherlands
{l.genga, n.tax, 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, which in general cannot be described by a single process model without overgeneralizing the allowed behavior. Several techniques for mining local patterns have been developed over the years, including Local Process Model (LPM) mining, episode mining, and the mining of frequent subtraces. These pattern mining techniques can be considered to be orthogonal, i.e., they provide different types of insights on the behavior observed in an event log. In this work, we demonstrate that the joint application of LPM mining and other patter mining techniques provides benefits over applying only one of them. First, we show how the output of a subtrace mining approach can be used to mine LPMs more efficiently. Secondly, we show how instances of LPMs can be correlated together to obtain larger LPMs, thus providing a more comprehensive overview of the overall process. We demonstrate both effects on a collection of real-life event logs.

1 Introduction

Process Mining [1] has emerged as a new field 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, and record for each event what was done, by whom, for whom, where, when, etc. One of the main challenges within process mining is *process discovery*, where the aim is to discover an interpretable and accurate model of the process based on an event log. The resulting process model provides insight into what is happening in the process and can be used as a starting point for more in-depth process analysis, e.g., bottleneck analysis [26], and checking compliance with rules and regulations [27].

In recent years, the scope of process discovery has broadened to novel application domains, such as software analysis and human behavior analysis. In some of those new application domains the logs have a *high degree of variability*, thereby making it difficult to represent the behavior observed in the log in a process model. High log variability significantly impacts the generation of insightful models; the process models obtained using process discovery techniques often do not provide useful insights into the process behavior, either because they overgeneralize, thus tending to allow for any sequence of events (e.g., [20,35]), or because, on the contrary, they represent exactly all (or most of) the behaviors recorded in the log, thus providing a spaghetti-like representation that is typically too complex to be exploited by a human analyst (e.g., [7]).

Several techniques aim to address this challenge of analyzing highly variable event logs. *Declarative process discovery* (e.g., [23,29]) focuses on the mining of binary relations between activities of the process. *Local Process Model (LPM) mining* (e.g., [32,33]) aims at the mining of a collection of process models instead of a single model, where each model captures a subset of the process behavior. *Subtrace mining* (e.g., [3,9,19]) mines subtraces that represent relevant sequential portions of process executions (i.e., subprocesses). In this work we will focus on subtrace 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 frequent patterns of sequential executions of process activities from event logs. Diamantini et al. [9] extend subtrace mining to discover partial order relations between process activities by either relying on a priori knowledge on concurrency relations or on concurrency detection mechanisms provided by process discovery techniques. However, subtrace mining techniques are not able to capture control-flow constructs other than sequential and concurrency relations between process activities. Rather, some approaches focus on relations *between patterns* instead of between activities from the process. For instance, the work in [9] constructs hierarchies of patterns where subtraces are ordered with respect to the inclusion relation. Genga et al. [13] apply frequent itemset mining techniques to mine partial order relations between subtraces.

Local Process Model (LPM) mining aims at mining process patterns that can describe any arbitrary combination of sequential ordering, 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 [32]. Therefore, a set of heuristics have been proposed in [32] to speed up the mining process. These heuristics discover subsets of process activities (called *projections*) that are strongly related and apply the LPM miner to each projection individually, aggregating the results by taking the union of the resulting LPMs. The downside of these heuristics is the loss of formal guarantees that all frequent local process models are found.

In this work, we explore the synergies between subtrace mining and LPM mining in two ways. First, we investigate the application of the patterns obtained using subtrace mining for LPM mining. This subtrace-based LPM mining approach generates *projections* based on the subtraces mined using the technique presented in [9] and furthermore extracts *ordering constraints* from the subtraces to reduce the search space of LPM mining. We conjecture that using activities from these subtraces as projections and ordering constraints can speed up the LPM mining procedure. Secondly, we explore the application of approaches to mine higher level relations *between* subtraces to generate larger LPMs. In particular, these approaches allow us to merge LPMs describing possibly unconnected portions of the process behavior, providing a more comprehensive overview of the overall process, which would otherwise be difficult to achieve using original LPM algorithms due to the large number of activities involved.

This paper is organized as follows. Sec. 2 introduces notation and basic concepts that are used throughout the paper. Sec. 3 presents a projection method and a constraint generation technique for LPM mining, while Sec. 4 presents a method to infer ordering relations

between mined LPMs. In Sec. 5 we evaluate both techniques on a collection of real-life event logs. Finally, Sec. 6 discusses related work and Sec. 7 concludes the paper.

2 Background

In this section we introduce notation and basic concepts used throughout the paper. We start by introducing event data and process models in Sec. 2.1 and then we introduce methods for mining subprocess models from event logs in Sec. 2.2.

2.1 Event Data & Process Models

Process models describe how processes should be carried out. Two process model notations that are commonly used in process mining are *process trees* [5] and *Petri nets* [28]. A process tree is a tree structure where leaf nodes represent process activities, while 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.

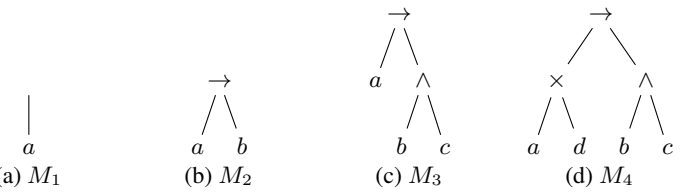
We formally define process trees recursively. Let Σ be the set of all process activities, $OP = \{\rightarrow, \times, \wedge, \circlearrowleft\}$ a set of operators and symbol $\tau \notin \Sigma$ denotes silent activities. We define a process tree pt as follows:

- $a \in \Sigma \cup \{\tau\}$ 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 model 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.

A *Petri net* $N = \langle P, T, F, \ell\rangle$ is a tuple where P is a finite set of *places*, T is a finite set of *transitions* such that $P \cap T = \emptyset$, $F \subseteq (P \times T) \cup (T \times P)$ is a set of directed arcs, called the flow relation, and $\ell : T \nrightarrow \Sigma$ is a labeling function that assigns process activities to transitions. Unlabeled transitions, i.e., $t{\in}T$ with $t{\notin}dom(l)$, are referred to as τ-transitions, or invisible transitions.

The state of a Petri net is defined by its *marking*. The marking assigns a finite number of tokens to each place. Transitions of the Petri net represent activities. The input places of a transition $t \in T$ are all places for which there is a directed edge to the transition, i.e. $\{p \in P|(p, t){\in}F\}$. The output places of a transition are defined similarly as $\{p \in P|(t, p){\in}F\}$. Executing a transition consumes one token from each of its input places and produces one token on each of its output places. A transition can only be executed when there is at least one token in each of its input places. Often we consider a Petri net in combination with an initial marking and a final marking, allowing us to define language $\mathfrak{L}(N)$, consisting of all possible sequences of visible transition labels (i.e., ignoring τ-transitions) that start in



(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.

the initial marking and end in the final marking. It is worth noting that process trees can trivially be transformed into Petri nets.

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 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., $L\restriction_{\{b,c\}} = [\langle b, c \rangle^5]$. $\#(\sigma, L)$ denotes the frequency of sequence $\sigma \in \Sigma^*$ as a subtrace within log L, e.g., $\#(\langle a, b \rangle, [\langle a, b, c \rangle^2, \langle a, b, d \rangle^3]) = 5$. $\sigma_1 \cdot \sigma_2$ denotes the concatenation of sequences σ_1 and σ_2, e.g., $\langle a, b \rangle \cdot \langle c, d, e \rangle = \langle a, b, c, d, e \rangle$.

2.2 Subprocess Mining

Two methods to mine subprocesses from event logs are *Local Process Models* (LPMs) [33] and *subtrace mining* [3,9,19]. LPMs are process models that describe frequent but partial behaviors seen in the event log, i.e., they model subsets of the process.

LPM Mining [33] is a technique to generate a ranked collection of LPMs through iterative expansion of candidate process trees. This technique encompasses four steps: 1) the *generation* of an initial set of process trees, consisting of one process tree for each activity; 2) the *evaluation* phase, where process tree quality is assessed by a set of tailored metrics; 3) the *selection* phase, where process trees that do not meet certain criteria are removed; 4) the *expansion* phase, where candidates selected at the previous step are expanded by replacing an activity node a by an operator node ($\rightarrow$, $\times$, $\wedge$ or $\circlearrowleft$), whose children are the replaced activity a and another activity $b \in \Sigma$ of the process. Steps 2 to 4 are repeated until no new candidate meets the criteria.

An LPM M can be expanded in many ways, as any one of its activity nodes can be replaced, using any of the operator nodes in combination with any other activity from the set of activities in the log. $Exp(M)$ denotes the set of expansions of M (described in more detail in [33]), and exp_max the maximum number of expansions allowed from an *initial LPM*, i.e., the LPMs generated in step 1.

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 operator node $\rightarrow$, with activity a as its left child node and b as 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

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

$\sigma = \langle a,d,b,c,d,c,a,c,b,d,a,b,a \rangle$

$\sigma\restriction_{\{a,b,c\}} = \langle a,b,c,c,a,c,b,a,b,a \rangle$

$\lambda_1\ \gamma_1\ \lambda_2\ \gamma_2\ \lambda_3$

$\Gamma_{\sigma,\text{LPM}} = \langle a,b,c,a,c,b \rangle$

(b) Segmentation of σ on M_3

Fig. 2: Example of segmentation in LPM mining.

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 as 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 activity a as its left child and activity d as its right child, 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*). LPMs are mined in process trees representation, but often their Petri net representation is used to visualize them.

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, $\sigma' = \lambda_1 \cdot \gamma_1 \cdot \lambda_2 \cdot \gamma_2 \ldots \lambda_n \cdot \gamma_n \cdot \lambda_{n+1}$ 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. $\Gamma_{\sigma,LPM} = \gamma_1 \cdot \gamma_2 \ldots \gamma_n$.

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 the 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 [33].

LPMs only contain a subset of the activities of a log L, and therefore, each LPM M can in principle be discovered on any projection on L containing the activities used in M. The computational complexity of LPM mining depends combinatorially on the number of activities in the log, and therefore, mining LPMs on projections of the log instead of on the

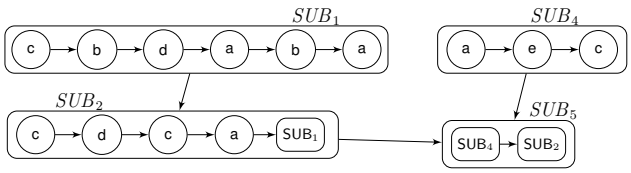


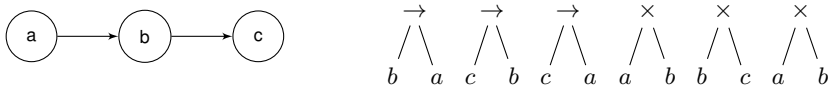
Fig. 3: Example of SUBDUE hierarchy

full significantly speeds up LPM mining. However, this results in a partial exploration of the LPM search space and does not guarantee that all LPMs meeting the support threshold are found. In principle, when the activities frequently following 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. This is desired, since interesting patterns can potentially exist in some activity set $\{a, b, c\}$, as well as in $\{a, b, d\}$, and discovering on both $L \restriction_{\{a,b,c\}}$ and $L \restriction_{\{a,b,d\}}$ and then merging the results is faster than discovering on $\{a, b, c\} \cup \{a, b, d\} = \{a, b, c, d\}$. The typical approach to generate the projection set for LPM mining is to apply Markov graph clustering [32] to a graph where vertices represent activities and edges represent the *connectedness* of two activities a and b based on following relations, i.e., $connectedness(a, b, L) = \sqrt{\frac{\#(\langle a,b\rangle,L)}{\#(\langle a\rangle,L)}^2 + \frac{\#(\langle a,b\rangle,L)}{\#(\langle b\rangle,L)}^2}$.

Subtrace Mining aims at finding frequent subsequences from logs. Diamantini et al. [9] apply *frequent subgraph mining* (FSM) to do so. In a first step, each trace $\sigma \in L$ is transformed into a directed graph $g = (V, E, \phi)$, with V the set of nodes that correspond to events in σ, E the set of the edges that show ordering relations between the events, and ϕ a labeling function associating nodes with the activities of the corresponding events. A node is created for each event in the trace and nodes representing subsequent events are connected with an edge. Once the set of graphs is obtained, an FSM algorithm is applied to derive frequent subgraphs from it, yielding the frequent subtraces in the event log. Diamantini et al. [9] use the SUBDUE algorithm [18] that adopts Description Length (DL) to iteratively select the most relevant 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 where each occurrence of s in G is replaced with a single node (i.e., compression). By doing so, SUBDUE relates the relevance of a subgraph with its compression capability.

At each iteration, 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 consists of a set of subtraces ordered according to their relevance. As an example, Figure 3 shows a portion of the SUBDUE output inferred from the set of graphs derived by the event log L=$\{\langle a, d, b, c, d, c, a, c, b, d, a, b, a\rangle^2, \langle a, e, c, c, d, c, a, c, b, d, a, b, a\rangle, \langle a, e, c, b, d\rangle\}$.

At the top level, we have subgraphs involving only elements of the original graphs set; while in the lower levels, we have subgraphs that involve upper level subgraphs in their



(a) Subtrace (b) Ordering constraints

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

definition. Since top-level subgraphs correspond to the most relevant subgraphs for the graphs set, we can reasonably expect to be able to capture most of the process behaviors by only considering the latter. Note that in this work we consider totally ordered traces, from which we construct sequential graphs; hence, the mined subgraphs are limited to sequential traces. However, SUBDUE can mine subgraphs that are more complex that just sequential traces, e.g., when partially ordered logs are analyzed.

3 Mining LPMs Using Subtrace Constraints & Projections

In this section, we present an approach to mine Local Process Models (LPMs) by exploiting subtrace mining results. This extends traditional LPM mining [33] by converting the set of subtraces (mined using the approach described in Sec. 2.2) into a set of *projections* (i.e., sets of activities) and a set of *ordering constraints* that are both used to restrict the set of possible expansions in the expansion phase of LPM mining. Projections restrict the possible extensions to the set of activities in the projection, while ordering constraints prohibit expansions into LPMs that violate ordering relations.

Each subtrace represents a frequent and connected portion of the process. Activities that do not co-occur together in a subtrace are unlikely to co-occur in a frequent LPM. Therefore, we extract one projection for each subtrace, consisting of the activities in the subtrace. Furthermore, LPMs that directly contradict the behavior of a subtrace can be extracted as ordering constraints, as they unlikely represent execution orders between activities. 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 $M = \oplus(a, b)$ with $a, b \in V$ and $\oplus \in \{\rightarrow, \times\}$.

Algorithm 1 describes the procedure to convert a subgraph into a set of ordering constraints. The algorithm generates a constraint for the exclusive choice tree between each pair of activities in the subtrace and adds it to the set of ordering constraints (line 4). Then, the algorithm checks for each pair of vertices in the subtrace whether there exists a path (possibly transitively) from vertex v_i to v_j (line 5). If so, a sequential constraint is added, thus prohibiting the reversed ordering (line 6). Fig. 4 shows the set of ordering constraints extracted for an example subtrace. Fig. 4a shows that most occurrences of activity *a* occurred before *b*, which, in turn, mostly occurred before *c*. Transitively, this means that *a* occurred before c. The three leftmost trees in Fig. 4b show the extracted ordering constraints that directly contradict those frequent orderings in the subtrace. Furthermore, the existence of the subtrace indicates that the activities tend to co-occur and do not tend to be mutually exclusive. Therefore, we can safely remove from the LPM search space the process trees that contain exclusive choices constructs between those activities.

Algorithm 1: Method *FindOrderingConstraints*

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

```
1 OC = {};
2 foreach v_i ∈ V do
3 |   foreach v_j ∈ V \ {v_i} do
4 |   |   OC = OC ∪ {×(φ(v_i), φ(v_j))};
5 |   |   if existsPath(v_i, v_j, E) then
6 |   |   |   OC = OC ∪ {→ (φ(v_j), φ(v_i))};
7 return OC;
```

Algorithm 2: Mining LPM using Subtraces

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

```
1  CP = ⟨⟩;
2  foreach s_i = (V_i, E_i, φ_i) ∈ S do
3  |   CP = CP · ⟨({φ_i(v_i) | v_i ∈ V_i}, findOrderingConstraints(s_i))⟩;
4  CP' = ∅;
5  foreach i ∈ {1, 2, ..., |CP}| do
6  |   (V_i, OC_i) = CP(i);
7  |   if ∃j ∈ {1, 2, ..., |CP|} : V_i ⊂ V_j ∨ (V_i = V_j ∧ j > i) then
8  |   |   continue;
9  |   foreach j ∈ {1, 2, ..., i − 1} do
10 |   |   if (V_i ⊆ V_j ∨ V_j ⊆ V_i) then
11 |   |   |   OC_i = OC_i ∪ OC_j
12 |   CP' = CP' ∪ {(V_i, OC_i)}
13 return MiningLPMwithProjectionsAndConstraints(L, CP');
```

Algorithm 2 describes how to mine LPMs from a log L given a set of subtraces S. For each subtrace in S, its set of activities and the ordering constraints are extracted (line 3), yielding set of projections and constraints CP. Algorithm *MiningLPMwithProjectionsAndConstraints* is invoked on the event log and the mined set of projections and constraints (line 13). This procedure mines LPMs in the traditional way, with an additional step in which every generated expansion $M_i \in Exp(M)$ of LPM M is first checked against the set of ordering constraints OC. If there exists a constraint $oc \in OC$ such that oc is a subtree of M_i, then M_i is discarded and not further expanded.

4 Deriving Partial Order Relations Over LPMs

In this section, we present an approach to discover partially ordered sets of Local Process Models (LPMs), which we will refer to as PO-LPMs. We adopt the approach in [13] for the mining of partial order relations between subtraces and adapt it to mine such relations between LPMs. We extract the following *ordering relations* between pairs of LPMs:

σ_1 :	⟨ a b c f l m g o n r ⟩
LPM_1^1	× × ×
LPM_2^1	× ×
LPM_3^1	× × × ×

(a)

σ_2 :	⟨ a b c f g a b c n r ⟩
LPM_1^1	× × ×
LPM_1^2	× × ×
LPM_2^1	× ×
LPM_3^1	

(b)

	LPM_1^1	LPM_1^2	LPM_2^1	LPM_3^1
σ_1	1	0	1	1
σ_2	1	1	1	0

(c)

Fig. 5: Building of the occurrence matrix for LPM_1, LPM_2 and LPM_3 and traces σ_1, σ_2.

(i) the *sequential* relation, denoted as $LPM_1 \twoheadrightarrow_{seq} LPM_2$, indicates that LPM_2 occurs immediately after LPM_1;
(ii) the *concurrent* relation, denoted as $LPM_1 \twoheadrightarrow_{conc} LPM_2$, indicates that the executions of the activities in the LPMs are interleaved;
(iii) the *eventually* relation, denoted as $LPM_1 \twoheadrightarrow_{ev} LPM_2$, indicates that LPM_2 occurs after LPM_1, but at least one other activity occurs between the two LPMs.

Given a set of LPMs $LPMS$ and log L, the approach first reduces $LPMS$ by *removing redundant ones* and then builds an *occurrence matrix* indicating in which traces each LPM occurs. Finally, it derives sets of the LPMs that frequently co-occur by applying frequent itemset mining to the occurrence matrix and then extracts the PO-LPM for each itemset by inferring the ordering relation on the log for each pair of LPMs in the itemset. We now explain each step in more detail.

Redundancy Reduction: First we apply existing techniques to remove *redundant* LPMs from the mined set of LPMs, i.e. LPMs that only describe behavior that is already represented by other LPMs in the set. This simplifies and speeds up the partial orders inferring step. We use the redundancy reduction technique of [30], which uses a greedy search approach to find a subset of LPMs that maximizes the number of events in the log covered while minimizing the number of LPMs used.

Occurrence Matrix: We build an occurrence matrix OM for event log L and the LPMs $LPMS'$ obtained using redundancy reduction, where each cell c_{ij} represents whether the $j\text{-}th$ LPM occurs in the $i\text{-}th$ trace. We build OM using segmentation function Γ. As shown in Sec. 2.2, function Γ can identify multiple instances of *the same LPM* in a single trace, therefore, in theory, multiple ordering relations can hold for a given pair of LPMs on a given trace. To deal with this property of Γ, we consider multiple instances of an LPM in a trace as *different* LPMs. Whenever we have more than one instance in a trace, we create a copy of the LPM for each of its occurrences and we set corresponding cells in the matrix to 1.

For example, consider the set of LPMs consisting of $LPM_1 = \{\rightarrow (\rightarrow (\mathsf{a}, \mathsf{b}), \mathsf{c})\}$, $LPM_2 = \{\wedge(\mathsf{f}, \mathsf{g})\}$ and $LPM_3 = \{\wedge((\rightarrow (\mathsf{l}, \mathsf{m}), (\rightarrow (\mathsf{o}, \mathsf{n}))\}$. Fig. 5 shows the occurrence matrix for these LPMs on traces σ_1 (Fig. 5a) and σ_2 (Fig. 5b), marking the events in the trace that belong to each LPM with $\times$. All three LPMs occur exactly once in σ_1, resulting in "1" values for all three LPMs in the occurrence matrix of Fig. 5c. In contrast, σ_2 contains *two* instances of LPM_1 (i.e., LPM_1^1 and LPM_1^2) and one of LPM_2.

	LPM_1	LPM_2	LPM_3
LPM_1	2	174	0
LPM_2	0	0	0
LPM_3	0	0	0

M_{seq}

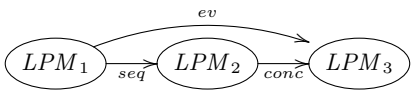
	LPM_1	LPM_2	LPM_3
LPM_1	0	0	0
LPM_2	0	0	199
LPM_3	0	0	0

M_{conc}

	LPM_1	LPM_2	LPM_3
LPM_1	0	25	199
LPM_2	0	0	0
LPM_3	0	0	0

M_{ev}

(a) Ordering relation matrices



(b) PO-LPM

Fig. 6: Deriving PO-LPMs for itemset $\{LPM_1, LPM_2, LPM_3\}$.

Deriving PO-LPMs: We infer sets of LPMs that frequently co-occur in the same trace by applying any *frequent itemset mining* algorithm (see [12] for an overview) using a support threshold ρ. Then, we determine the ordering relations between the LPMs per set of frequently co-occurring LPMs. For each set of frequently co-occurring LPMs we extract the traces from the log in which these LPMs co-occur. Using Γ we obtain the instances of the LPMs in these traces, from which we can extract the starting and ending position of each instance and determine whether $LPM_i \rightarrowtriangle_{seq} LPM_j$, $LPM_i \rightarrowtriangle_{conc} LPM_j$, or $LPM_i \rightarrowtriangle_{ev} LPM_j$ holds with LPM_i, LPM_j two co-occurring LPMs. For each pair of LPMs occurring in the same itemset we store the number of traces for which these relations between the LPMs hold respectively in matrices M_{seq}, M_{conc} and M_{ev}. Based on these matrices we extract as ordering relations between LPMs those relations that exceed a user-defined support threshold η, resulting in the PO-LPMs. Note that with ρ and η there are two distinct support thresholds. This is motivated by the fact that a pair of LPMs can occur in different order in different traces, and therefore the support of an ordering relation can be smaller than the support of the itemset. Note that η can be considered as the *confidence* of the ordering relations.

As an example, consider again LPMs LPM_1, LPM_2, LPM_3 and trace σ_1, in which all three LPMs occur. Analyzing the positions of the events belonging to each LPM in Fig. 5a we observe that LPM_2 occurs immediately after LPM_1 (i.e., $LPM_1 \rightarrowtriangle_{seq} LPM_2$), that LPM_2 is interleaved with LPM_3 (i.e., $LPM_2 \rightarrowtriangle_{conc} LPM_3$) and LPM_3 eventually occurs after LPM_1 (i.e., $LPM_1 \rightarrowtriangle_{ev} LPM_3$). Suppose that for some set of LPMs and log L that among others contains σ_1 , itemset $\{LPM_1, LPM_2, LPM_3\}$ is extracted as a set of frequently co-occurring LPMs, and M_{seq}, M_{conc} and M_{ev} are as shown in Fig. 6a, then Fig. 6b shows the PO-LPM for this itemset for $\eta = 50\%$ of the traces. Note that the use of the thresholds ρ and η ensures us to infer PO-LPMs that meet minimum support requirements, as in the case of single LPMs.

5 Evaluation

In this section, we describe two sets of experiments. First, we evaluate the speedup in Local Process Model (LPM) mining that is obtained by applying the technique of Sec. 3.

Secondly, we explore the resulting PO-LPMs obtained by applying the technique of Sec. 4. We evaluate both on the same collection of real-life event logs, which is described below.

Datasets: We evaluated our technique using four 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'12* log [10]. 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 [4]. 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 [24]. The SEPSIS log contains 1050 traces, 15214 events, and 16 activities. Finally, as fourth event log we use a dataset from the lighting system of a smart office environment, which was gathered in [36]. This dataset contains continuous values for the color temperature and the light intensity of the lighting in four different areas in the office space. Events correspond to interactions with the lighting interface that result in changes in the color temperature and intensity of the lighting in one or more areas and each case is a working day. The event names are converted from continuous values to symbolic activities using the well-known technique SAX [21], resulting in eight categories for each event representing the color temperature and intensity in each of the four areas. We refer to this log as *Laplace* and it contains 92 traces, 1557 events and 218 activities.

5.1 Mining LPMs Using Subtraces

We now explore the effect of using subtraces to the efficiency of LPM mining, for which we perform two sets of experiments. First, we investigate the effects of only using SUBDUE projections, i.e., using the projection-based LPM mining procedure of [32] while using the activities in SUBDUE subtraces as projections. Then, we exploit both projections and ordering constraints as described in Sec. 3.

Tools & Configurations: We use the iterative Markov LPM mining algorithm implemented in the *LocalProcessModelDiscovery* package[1] of the ProM framework [34]. We have implemented the novel LPM mining approach based on SUBDUE projections and constraints in the ProM package *LocalProcessModelDiscoveryWithSubdueConstraints*[2]. For both Markov-based LPM and subtrace-based LPM mining we use the standard ProM configurations. For SUBDUE we use the standard implementation[3], in which we varied the number of iterations. Note that a high number of SUBDUE iterations is expected to be beneficial for the quality of the LPM results: more iterations lead to a more process behavior being captured in subtraces. However, this negatively impacts the speedup of LPM mining. Moreover, by construction, SUBDUE extracts the largest frequent subtraces in the first iterations. Hence, we expect the obtained subtraces to be able to represent most of the process behaviors even using only a few iterations. Therefore, we explore using 1,

[1] https://svn.win.tue.nl/repos/prom/Packages/LocalProcessModelDiscovery/
[2] https://svn.win.tue.nl/repos/prom/Packages/LocalProcessModelDiscoveryWithSubdueConstraints
[3] http://ailab.wsu.edu/subdue/

5 and 10 iterations, and to verify our assumption we additionally use 10000 iterations. All experiments are performed on an 2.4 GHz Intel i7 machine, equipped with 16 Giga of RAM.

Methodology: We evaluate our approach using two dimensions. First, we consider the reduction in the search space size, which represents how much speedup is obtained in the mining procedure. Secondly, we consider the quality of the mined LPMs by comparing the LPMs mined using SUBDUE projections and constraints to those mined when using the full search space. This second dimension is relevant since using projections with LPM mining might lead to not all LPMs being found [32]. By comparing the LPM rankings obtained by mining with and without projections we can assess to what extent the use of projections affects the results. We compare the ranking using *Normalized Discounted Cumulative Gain* (NDCG) [6,17], which is a widely used metrics to evaluate ranked results in information retrieval. 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 we apply the Markov-based projection technique from [32] iteratively until all projections contain at most seven activities, and compare the search space reduction and the NDCG obtained when using this approach with the search space reduction and NDCG obtained when using projections and constraints from SUBDUE subtraces.

Results: Table 1 shows the results for the four logs. The results obtained without using projections or constraints are considered to be the ground truth LPM ranking and therefore have NDCG@k values of 1.0 by definition. The results obtained by the best heuristic configuration(s) are reported in bold and between parenthesis is the number of SUBDUE iterations.

Iterative Markov [32] projections result in a reduction of the search space by a factor between 43.50x (BPI'12) and 4365.48x (Laplace), while the high NDCG values indicate that the majority of the top 20 LPMs of the ground truth are still found. Using the constraints extracted from SUBDUE subtraces obtained with 10k SUBDUE iterations together with iterative Markov projections further increases the speedup of LPM mining on all four logs while resulting in identical LPM rankings.

The search space size of LPM mining with SUBDUE projections depends on the number of iterations performed by SUBDUE: more iterations result in a larger number of unique sets of activities, leading to more projections and a larger LPM search space, but at the same time increasing the quality of the mined LPMs in terms of NDCG. Note that the quality of the LPM mining results differs between the logs when one SUBDUE iteration is used. This is because for logs with few activities a single subtrace can already capture most relevant process behavior, while for logs with many activities it can only capture a small part. The use of SUBDUE projections leads to a higher speedup than iterative Markov

Table 1: The search space size and NDCG results for mining LPMs with and without SUBDUE projections and constraints.

Event Log	**Projections** (iterations)	**Constraints** (iterations)	**Search Space Size**	**Speedup**	**NDCG@5**	**NDCG@10**	**NDCG@20**
BPI'12	None	None	1567250	-	1.0000	1.0000	1.0000
	Iterative Markov	None	36032	43.50x	0.9993	0.9987	0.9865
	Iterative Markov	SUBDUE (10k)	21084	74.33x	0.9993	0.9987	0.9865
	SUBDUE (1)	None	10608	147.74x	0.9993	0.9987	0.9830
	SUBDUE (5)	None	10740	145.93x	**1.0000**	**0.9994**	0.9870
	SUBDUE (10)	None	10904	143.73x	**1.0000**	**0.9994**	**0.9903**
	SUBDUE (10k)	None	12666	123.74x	**1.0000**	**0.9994**	**0.9903**
	SUBDUE (1)	SUBDUE (1)	2718	576.62x	0.9993	0.9987	0.9830
	SUBDUE (5)	SUBDUE (5)	**2620**	**598.19**x	**1.0000**	**0.9994**	0.9870
	SUBDUE (10)	SUBDUE (10)	2874	545.32x	**1.0000**	**0.9994**	**0.9903**
	SUBDUE (10k)	SUBDUE (10k)	4012	390.64x	**1.0000**	**0.9994**	**0.9903**
Receipt phase	None	None	1451450	-	1.0000	1.0000	1.0000
	Iterative Markov	None	12074	120.21x	0.9418	0.8986	0.8238
	Iterative Markov	SUBDUE (10k)	10610	136.80x	0.9418	0.8986	0.8238
	SUBDUE (1)	None	8176	177.53x	**1.0000**	**0.9994**	0.9903
	SUBDUE (5)	None	8256	175.81x	**1.0000**	**0.9994**	0.9903
	SUBDUE (10)	None	8264	175.64x	**1.0000**	**0.9994**	**0.9958**
	SUBDUE (10k)	None	8504	170.68x	**1.0000**	**0.9994**	**0.9958**
	SUBDUE (1)	SUBDUE (1)	4012	390.64x	**1.0000**	**0.9994**	0.9903
	SUBDUE (5)	SUBDUE (5)	**1862**	**779.51x**	**1.0000**	**0.9994**	0.9903
	SUBDUE (10)	SUBDUE (10)	2170	668.71x	**1.0000**	**0.9994**	**0.9958**
	SUBDUE (10k)	SUBDUE (10k)	2178	666.41x	**1.0000**	**0.9994**	**0.9958**
SEPSIS	None	None	315451	-	1.0000	1.0000	1.0000
	Iterative Markov	None	6304	50.04x	0.9332	0.9148	0.8613
	Iterative Markov	SUBDUE (10k)	3768	83.72x	0.9332	0.9148	0.8613
	SUBDUE (1)	None	12	26287.58x	0.5763	0.3771	0.2489
	SUBDUE (5)	None	334	994.46x	0.9916	0.9671	0.9472
	SUBDUE (10)	None	394	800.64x	**0.9923**	**0.9692**	**0.9534**
	SUBDUE (10k)	None	1034	305.08x	**0.9923**	**0.9692**	**0.9534**
	SUBDUE (1)	SUBDUE (1)	**10**	**31545.10x**	0.5763	0.3771	0.2489
	SUBDUE (5)	SUBDUE (5)	174	1812.94x	0.9916	0.9671	0.9472
	SUBDUE (10)	SUBDUE (10)	144	2190.63x	**0.9923**	**0.9692**	**0.9534**
	SUBDUE (10k)	SUBDUE (10k)	470	671.17x	**0.9923**	**0.9692**	**0.9534**
Laplace	None	None	4784569	-	1.0000	1.0000	1.0000
	Iterative Markov	None	1096	4365.48x	0.8261	0.7942	0.7635
	Iterative Markov	SUBDUE (10k)	746	6413.63x	0.8261	0.7942	0.7635
	SUBDUE (1)	None	12	398714.08x	0.4354	0.2836	0.1841
	SUBDUE (5)	None	42	113918.31x	0.5061	0.3296	0.2139
	SUBDUE (10)	None	72	66542.35x	0.7909	0.5683	0.3689
	SUBDUE (10k)	None	730	6554.20x	**0.9096**	**0.8690**	**0.7928**
	SUBDUE (1)	SUBDUE (1)	**10**	**478456.90x**	0.4354	0.2836	0.1841
	SUBDUE (5)	SUBDUE (5)	34	140722.62x	0.5061	0.3296	0.2139
	SUBDUE (10)	SUBDUE (10)	56	85438.73x	0.7909	0.5683	0.3689
	SUBDUE (10k)	SUBDUE (10k)	582	8220.91x	**0.9096**	**0.8690**	**0.7928**

projections on all logs, even SUBDUE constraints are not used. At the same time, when enough SUBDUE iterations are used, SUBDUE projections result in higher NDCG. This shows that SUBDUE subtraces are more effective in finding related sets of activities for use as projections in LPM mining compared to Markov clustering.

The constraints extracted from SUBDUE subtraces in combination with SUBDUE-based projections results in considerably higher speedup on all logs without resulting in lower NDCG. On three logs, using 10 SUBDUE iterations is sufficient to achieve the highest quality LPMs, while only on the Laplace log more iterations are needed. Using SUBDUE projections and constraints we have found speedups between 598.19x (BPI'12) and 478456.90x (Laplace). To put these results into perspective: this brought down the mining time on the BPI'12 log from 24 minutes to less than two minutes. This shows that subtrace mining results can be used to speed up LPM mining. Additionally, in [31] we showed that the mined LPMs provide additional process insights in comparison to subtraces, meaning that it is actually useful to perform LPM mining after subtrace mining.

5.2 Mining Ordering Relations Over LPMs

In this section, we evaluate our approach to discover PO-LPMs from a set of LPMs. We propose a set of measures to assess the quality of PO-LPMs and we discuss the results that we obtained for the four logs. Note that the notion of quality exploited in these experiments differs from the one used before. In the previous experiments, the quality of the different LPMs set was intended as their similarity with the set of LPMs discovered by the exhaustive search, to evaluate the impact of the pruning of the search state. Here, we focus on exploring the benefits of considering larger and possible *unconnected* portions of process behaviors. Therefore, we evaluate the balance between the loss in support and the gaining in size of PO-LPMs with respect to single LPMs sets. Additionally, we show how PO-LPMs can be used to merge LPMs resulting in higher-level LPMs that describe a larger fragment of the process.

Tools & Configurations: For each log we use the set of LPMs that we obtained in the experiments of Sec. 5.1 for projections using 10k SUBDUE iterations and use the implementation of the technique to reduce redundancy in LPM results [30] as available in ProM package *LocalProcessModelConformance*[4]. We implemented the PO-LPM mining approach of Sec. 4 in PHP[5] and use the implementation of the FP-Growth itemset mining algorithm in the SPMF pattern mining library [11] to obtain the frequent itemsets (FI hereafter), i.e. sets of frequently co-occurring LPMs.

Methodology: We test our technique with three types of sets of FI: 1) the entire set of FI; 2) the set of *closed* FI, i.e. the subset of FI where for each itemset i there exists no other itemset j such that $i \subset j$ with identical support to i; 3) the set of *maximal* FI, i.e. FI where for each itemset i there exists no other itemset j with $i \subset j$ where the support of j exceeds ρ. We vary ρ from 1% to 100% increasing it in steps of 1%. We set η=50% since, as a rule of thumb, it is reasonable to consider only ordering relations occurring at least in more than half of the cases in which the LPMs occur together. Lower values for η would likely result in PO-LPMs involving multiple and infrequent relations between pairs of LPMs, thus affecting the understandability and the representative capability of the output. We evaluate the quality of the discovered PO-LPMs along two dimensions: 1) the amount of information provided

[4] https://svn.win.tue.nl/repos/prom/Packages/LocalProcessModelConformance
[5] https://surfdrive.surf.nl/files/index.php/s/PeD64m5xr5hxcqi

Table 2: LPMs set statistics inferred from the three event logs.

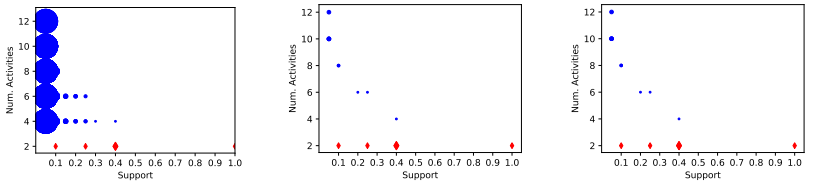
Log	FI	#LPMs	Avg. #Act	Avg. Supp (%)	Avg. IR
BPI'12	-	5	2	41.6	0.036
	All	210	7.05	3,16	0.006
	Closed	9	7.6	20.5	0.035
	Maximal	9	7.6	20.5	0.035
SEPSIS	-	5	2.4	53.8	0.090
	All	136	7.61	3.7	0.015
	Closed	46	7.85	7.6	0.037
	Maximal	26	8.69	8,8	0.038
Laplace	-	18	2	9	0.0009
	All	65758	15.97	1	0.0007
	Closed	21	8.19	1	0.0004
	Maximal	21	8.19	1	0.0004

on the process (i.e., pattern size) and 2) the portion of process behaviors they represent (i.e., their support). It is easy to see that there is a trade-off between these dimensions: larger patterns typically have lower support. What is the optimal trade-off between the two dimensions depends on the process analysis task at hand and needs to be decided by the process analyst. Here, we investigate the trade-off between the dimensions as a result of ρ. To capture both dimensions in a single measure we also define *Information Ratio* (IR) measure as follows: $IR = \frac{\#activitiesLPM}{\#activitiesProcess} \times \frac{\#occurrences}{\#traces}$. Function IR yields values in interval $[0, 1]$ with 0 corresponding to an empty set of LPMs and 1 corresponding to a set of LPMs that involves all process activities and occurs in all traces.

Results: Tab. 2 reports statistics on the set of PO-LPMs that are inferred from three of the four logs for the three different itemset mining approaches, as well as for the original set of LPMs ("-" in column *FI*). Columns *#LPMs*, *Avg. #Act*, *Avg. Supp (%)*, and *Avg. IR* respectively indicate the number of the LPMs in the set, the average number of activities per LPM, the average support of the LPMs, and the average information ratio. The receipt phase log is missing in the table, as only two LPMs remained after the redundancy reduction step, between which no ordering relation could be found.

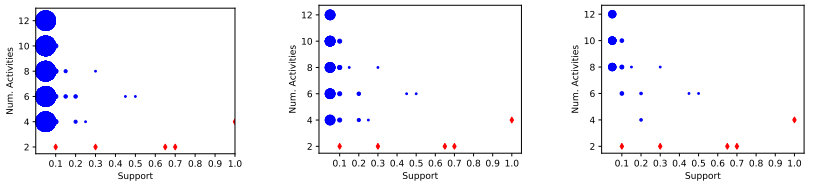
It should be noted that by combining the LPMs inferred from each log, for the BPI'12 log we derived 10 activities out of 23 activities in the process. Respectively for the SEPSIS and Laplace logs we derived 10 out of 16 and 30 activities out of 218 activities. This suggests that the processes under analysis involve many infrequent activities that do not need to be modeled to capture most of the structure in the process, and therefore, they are not in the LPM set. In turn, this implies that most of the LPMs involve only a small fraction of the process activities. This explains why the IR values overall are very small, regardless of log and settings. However, this does not affect our analysis, since we investigate how IR values vary on the same log between different PO-LPMs sets instead of considering their absolute value.

BPI'12 All FI configurations led to PO-LPMs involving over three times the amount of activities of the initial LPMs. Using all PO-LPMs results in a large support drop, while it does



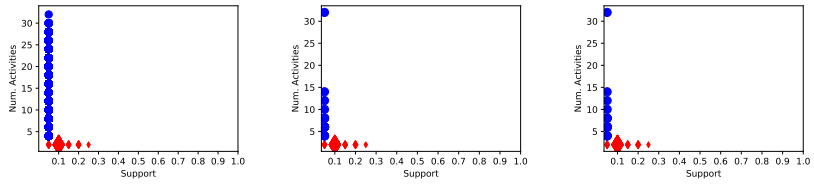
(i) All (ii) Closed (iii) Maximal

(a) BPI'12



(i) All (ii) Closed (iii) Maximal

(b) SEPSIS



(i) All (ii) Closed (iii) Maximal

(c) Laplace

Fig. 7: The support and the number of activities of each LPM in the result set using for original LPMs (red diamonds) compared with PO-LPMs (blue circles) obtained using all, closed and maximal frequent itemsets.

not lead to larger LPMs compared to the closed and maximal PO-LPMs. Fig. 7a shows more detailed results by plotting support against the number of activities for single LPMs, and after merging them using all, closed, and maximal frequent itemsets. We discretized support values in bins of 5%. Each dot in the plot represents the set of LPMs that involve n activities and has a support within $[s - 0.05, s]$, where s denotes the support represented by the bin. The larger the size of the dot is, the larger the size of the corresponding set of LPMs is.

All configurations led to LPMs with a dimension at least double than and up to six times the dimension of single LPMs. The set of all PO-LPMs involves a high number of LPMs with a support smaller or equal to 5%, which motivates the low support values and, in turn, the worsening of the IR values with respect to the single set. Note that most of these low-support PO-LPMs were discarded in the closed and maximal sets, which involve only 10 LPMs each, against the 210 of all PO-LPMs. However, most of the LPMs in these sets

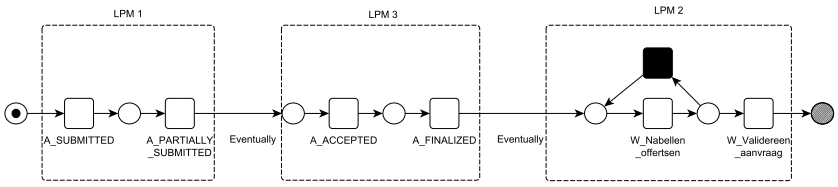


Fig. 8: One of the PO-LPMs from the closed set for BPI'12.

have a support lower than or equal to 20%, thus leading to an average support value equal to around the half of single LPMs.

Fig. 8 presents an example of a merged LPM that is built from the PO-LPM obtained from the closed (maximal) set. The PO-LPM is represented as a Petri net (introduced in Sec. 2.1), where circles represent places, rectangles represent transitions, and black rectangles depict τ-transitions. Places that belong to the initial marking contain a token and places belonging to a final marking are marked as  . The dotted lines surround the original LPMs and edges between original LPMs are labeled with their ordering relation. This merged LPM consists of eventually relations between LPM 1 and LPM 3 and between LPM 3 and LPM 2. This PO-LPM shows that after submitting a loan application it was accepted and finalized, followed by one or more calls to the customer for additional information and finally a validation of the application documents. This merged LPM occurs in 25% of the traces, which is significant given its size. Note that LPMs of this size cannot be mined with existing techniques. Given its size and support this PO-LPM provides the analyst with a higher-level and more meaningful representation of the process compared to the three LPMs separately.

SEPSIS We obtained a small number of single LPMs, mostly comprising two activities, with one LPM involving 4 activities. PO-LPMs are on average three times larger than the single LPMs. However, for this log the increase in size was not enough to properly balance the loss in terms of support; indeed, all configurations achieved IR values worse than the set of original LPMs. The set of all PO-LPMs is again the one with the lowest IR value, while the closed and maximal sets have similar performance. Fig. 7b provides the scatter plot of size/support for LPMs obtained from the SEPSIS log for all tested configurations. The PO-LPMs have a size up to six times the one of most single LPMs; however, many of them have a support between 1% and 5%. Some of these low-support LPMs were not filtered neither in the closed nor in the maximal set.

Fig. 9 reports one of the PO-LPMs with the highest support. It starts with the registration of the patient in the emergency room (ER), followed by filling the general triage document (*ER Triage*), which is done concurrently to either filling in a triage form for sepsis cases (*ER SEPSIS Triage*) or the infusion of some liquids (*IV Liquid*). Later in the process, LPM 4 and LPM 5 are executed in parallel. LPM 4 shows that the patient was admitted into the normal care ward and CRP was performed (i.e., a test to detect inflammation); LPM 5 shows that the patient's leukocytes were tested and she was then sent back to the emergency room. Also here, we obtained a meaningful description of interconnected phases of the process and obtained a reasonable support value (i.e, 12%).

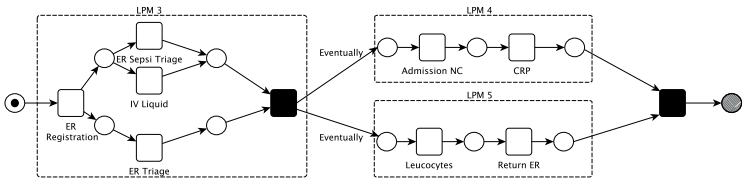


Fig. 9: One of the PO-LPMs from the maximal set for SEPSIS.

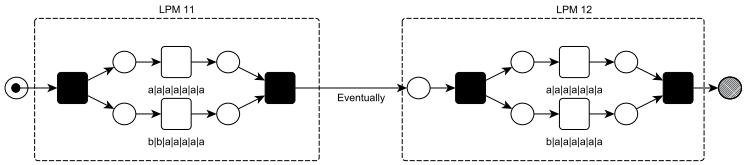


Fig. 10: One of the PO-LPMs from the closed set for Laplace.

Laplace Both single LPMs and PO-LPMs lead to very low IR values for the Laplace log. The reason is that this log is much less structured than the other logs we analyzed, resulting in LPMs with a low support (1%-2%). Moreover, because this process contains many activities, these LPMs involve a small fraction of those activities. On this log, differently from the other logs, the all itemsets configuration obtains high IR values than the others. All configurations led to an average support of 1%; however, PO-LPMs obtained from all frequent itemsets have twice the average size compared to closed and maximal itemsets. The main reason is that the maximal and closed sets filter out many large (but non-maximal or non-closed) PO-LPMs. Fig. 7c provides the scatter plots for size/support for LPMs in the Laplace log for all tested configurations. As expected, we got many large LPMs in the lowest support interval for all configurations. This is particularly evident for the first configuration. It is worth noting that PO-LPMs were able to achieve growth of up to 15 times the average size compared to single LPMs, although this growth comes with a decrease of support from around 40% to a maximum of 1%.

Figure 10 shows one of the closed PO-LPMs with the highest support mined for the Laplace log. The figure shows that when we have a couple of signals, occurring in every order, representing a switch either from low to high values or from high to low values for both the color tone and the light intensity for the first office area, this pair is eventually followed by another pair of signals, representing a switch of the light intensity values in the same area. The support of this PO-LPM is 2%.

6 Related Work

We discuss two areas of related work: *subprocess mining* and *partial order discovery*.

Subprocess Mining. 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 [3,16]. For instance, Bose et al. [3] mine subprocesses by identifying sequences of events that fit a priori defined templates.

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.

Several other techniques, like LPM mining, focus on the mining of more complex patterns that allow for control-flow constructs. Chapela-Campa et al. [8] developed a technique called WoMine-i to mine subprocess patterns with multiple control-flow constructs that are *infrequent*. Lu et al. [22] recently proposed an interactive subprocess exploration tool, which allows the discovery of subprocess patterns that a process analyst can modify based on domain knowledge. Greco et al. [14] propose a Frequent Subgraph Mining (FSM) algorithm that exploits knowledge about relationships among activities (e.g., AND/OR splits) 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.

Partial Order Discovery. The discovering of partial ordering relations among log events has been traditionally addressed by *Episode Discovery* [25]. An episode is defined as a collection of partially ordered events. The goal of Episode Discovery consists in determining all the episodes in an event log whose support is above a user-defined threshold. Episodes are usually detected by grouping together events falling in the same window (e.g., a time or a proximity window), generating all possible candidates (i.e., all possible partial orders configuration) and then checking the frequency of the candidates. Since the seminal work of Mannila et al. [25], several approaches have been proposed to enhance the efficiency of episode discovery, addressing different application domains (e.g., [15,37]). Recently, Leemans et al. [19] introduced an approach tailored to discover episodes from event logs generated by business processes, where it is possible to exploit the notion of process instance to determine the episodes. The output of their approach consists of directed graphs where nodes correspond to activities and edges to eventually follow precedence relations. Our work presents some similarities with [19], in the sense that also the discovery of our PO-LPMs is based on the notion of process instances. However, our work defines ordering relations among patterns of events, rather than between single events. Moreover, our approach provides a more fine grained analysis by distinguishing among sequential, eventually and concurrency relations.

7 Conclusions and Future Work

In this work, we have explored the synergy effects between subtrace and LPM mining, showing how their combination enables the gathering of relevant process insights that would remain hidden when both are applied separately. Specifically, we extended the LPM algorithm in [32] to account for ordering constraints mined using SUBDUE subtraces. Moreover, we proposed an approach (adapting the approach of [13]) to derive ordering relations between LPMs to infer partial orders between them. We evaluated our approach on four 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. Our experiments also show that the approach is able to infer partially ordered models, thereby providing a more complete and meaningful overview on the process compared to single LPMs, although this comes at the price of a loss in support.

In future work, we plan to explore the use of other subtrace mining techniques to derive LPMs. Moreover, we plan to investigate semi-automatic techniques to move from PO-LPMs to process models expressed in a standard notation by converting the partial relations in actual process constructs. This will allow the reuse of the discovered process patterns for further analysis.

Acknowledgement This work is partially supported by ITEA3 through the APPSTACLE project (15017) and by the RSA-B project SeCludE.

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. Bose, R.P.J.C., van der Aalst, W.M.P.: Abstractions in process mining: A taxonomy of patterns. In: BPM. pp. 159–175. Springer (2009)
4. 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
5. Buijs, J.C.A.M., van Dongen, B.F., van der Aalst, W.M.P.: A genetic algorithm for discovering process trees. In: CEC. pp. 1–8. IEEE (2012)
6. Burges, C., Shaked, T., Renshaw, E., Lazier, A., Deeds, M., Hamilton, N., Hullender, G.: Learning to rank using gradient descent. In: ICML. pp. 89–96 (2005)
7. Carmona, J., Cortadella, J., Kishinevsky, M.: A region-based algorithm for discovering petri nets from event logs. In: Dumas, M., Reichert, M., Shan, M.C. (eds.) Business Process Management. pp. 358–373. Springer Berlin Heidelberg, Berlin, Heidelberg (2008)
8. Chapela-Campa, D., Mucientes, M., Lama, M.: Discovering infrequent behavioral patterns in process models. In: BPM. pp. 324–340. Springer (2017)
9. Diamantini, C., Genga, L., Potena, D.: Behavioral process mining for unstructured processes. Journal of Intelligent Information Systems 47(1), 5–32 (2016)
10. van Dongen, B.F.: BPI challenge 2012 (2012), doi:10.4121/uuid:3926db30-f712-4394-aebc-75976070e91f
11. Fournier-Viger, P., Gomariz, A., Gueniche, T., Soltani, A., Wu, C.W., Tseng, V.S.: SPMF: a Java open-source pattern mining library. Journal of Machine Learning Research 15(1), 3389–3393 (2014)
12. Fournier-Viger, P., Lin, J.C.W., Vo, B., Chi, T.T., Zhang, J., Le, H.B.: A survey of itemset mining. Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery 7(4) (2017)
13. Genga, L., Potena, D., Martino, O., Alizadeh, M., Diamantini, C., Zannone, N.: Subgraph mining for anomalous pattern discovery in event logs. In: NFMCP. pp. 181–197. Springer (2017)
14. 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)
15. Huang, K.Y., Chang, C.H.: Efficient mining of frequent episodes from complex sequences. Information Systems 33(1), 96 – 114 (2008)
16. Huang, Z., Lu, X., Duan, H.: On mining clinical pathway patterns from medical behaviors. Artificial intelligence in medicine 56(1), 35–50 (2012)
17. Järvelin, K., Kekäläinen, J.: Cumulated gain-based evaluation of ir techniques. ACM Transactions on Information Systems 20(4), 422–446 (2002)
18. Jonyer, I., Cook, D., Holder, L.: Graph-based Hierarchical Conceptual Clustering. Journal of Machine Learning Research 2, 19–43 (2002)

19. Leemans, M., van der Aalst, W.: Discovery of frequent episodes in event logs. In: SIMPDA. pp. 1–31. Springer (2015)
20. Leemans, S.J., Fahland, D., van der Aalst, W.M.: Discovering block-structured process models from event logs containing infrequent behaviour. In: International conference on business process management. pp. 66–78. Springer (2013)
21. Lin, J., Keogh, E., Lonardi, S., Chiu, B.: A symbolic representation of time series, with implications for streaming algorithms. In: SIGMOD Workshop on Research Issues in DM&KD. pp. 2–11. ACM (2003)
22. 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: CoopIS. Springer (2018)
23. Maggi, F.M., Mooij, A.J., van der Aalst, W.M.P.: User-guided discovery of declarative process models. In: CIDM. pp. 192–199. IEEE (2011)
24. Mannhardt, F., Blinde, D.: Analyzing the trajectories of patients with sepsis using process mining. In: RADAR+EMISA. pp. 72–80. CEUR (2017)
25. Mannila, H., Toivonen, H., Verkamo, A.I.: Discovery of frequent episodes in event sequences. Data mining and knowledge discovery 1(3), 259–289 (1997)
26. Măruşter, L., van Beest, N.R.T.P.: Redesigning business processes: a methodology based on simulation and process mining techniques. Knowledge & Information Systems 21(3), 267–297 (2009)
27. 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)
28. Reisig, W.: Petri nets: an introduction, vol. 4. Springer (2012)
29. Schönig, S., Cabanillas, C., Jablonski, S., Mendling, J.: Mining the organisational perspective in agile business processes. In: BPMDS. pp. 37–52. Springer (2015)
30. Tax, N., Dumas, M.: Mining non-redundant sets of generalizing patterns from sequence databases. arXiv preprint arXiv:1712.04159 (2017)
31. Tax, N., Genga, L., Zannone, N.: On the Use of Hierarchical Subtrace Mining for Efficient Local Process Model Mining. In: Proceedings of International Symposium on Data-driven Process Discovery and Analysis. pp. 8–22. CEUR-WS.org (2017)
32. Tax, N., Sidorova, N., van der Aalst, W.M.P., Haakma, R.: Heuristic approaches for generating local process models through log projections. In: CIDM. pp. 1–8. IEEE (2016)
33. 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)
34. 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: BPM Demos. vol. 615, pp. 34–39. CEUR (2010)
35. Van der Werf, J.M.E., van Dongen, B.F., Hurkens, C.A., Serebrenik, A.: Process discovery using integer linear programming. In: International conference on applications and theory of petri nets. pp. 368–387. Springer (2008)
36. van de Werff, T., Niemantsverdriet, K., van Essen, H., Eggen, B.: Evaluating interface characteristics for shared lighting systems in the office environment. In: DIS. pp. 209–220. ACM (2017)
37. Zhou, W., Liu, H., Cheng, H.: Mining closed episodes from event sequences efficiently. In: PAKDD. pp. 310–318. Springer (2010)