From Place Nets to Local Process Models

Viki Peeva, Lisa L. Mannel, and Wil M. P. van der Aalst

RWTH Aachen University, 52062 Aachen Germany
{peeva, mannel, wvdaalst}@pads.rwth-aachen.de

Abstract. Standard process discovery algorithms find a single process model that describes all traces in the event log from start to end as best as possible. However, when the event log contains highly diverse behavior, they fail to find a suitable model, i.e., a so-called "flower" or "spaghetti" model is returned. In these cases, discovering local process models can provide valuable information about the event log by returning multiple small process models that explain local behavior. In addition to explainability, local process models have also been used for event abstraction, trace clustering, outcome prediction, etc. Existing approaches that discover local process models do not scale well on event logs with many events or activities. Hence, in this paper, we propose a novel approach for discovering local process models composed of so-called place nets, i.e., Petri net places with the corresponding transitions. The place nets may correspond to state- or language-based regions, but do not need to. The goal however is to build multiple models, each explaining parts of the overall behavior. We also introduce different heuristics that measure the model's frequency, simplicity, and precision. The algorithm is scalable on large event logs since it needs only one pass through the event log. We implemented our approach as a ProM plugin and evaluated it on several data sets.

Keywords: Local process models · Process mining · Process discovery.

1 Introduction

The main goal of process mining is to help people analyze and improve processes. One subarea of process mining is process discovery which automatically creates process models from available event logs [1]. Process discovery techniques [13,15,26,27] try to explain and visualize the process from start to end, while other algorithms like sequence and episode mining [19,21] try to mine small patterns that frequently happen in the event log. This paper will focus on local process model discovery which was first introduced in [24] as an individual branch and was positioned between process discovery and pattern mining. Local process models are able to describe complex constructs in contrast to sequences and episodes, but keep the local perspective introduced in pattern mining, which separates them from process discovery. This way, instead of describing a process with one overall model, a set of models is used.

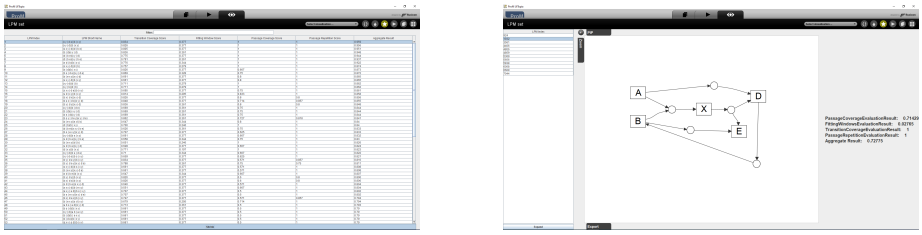
Some processes we want to analyze are too diverse to have a clear structure. Thus, making it almost impossible to discover an end-to-end model, resulting in a discovery of a so-called "flower" or "spaghetti" models. Hence, one straightforward use-case of local process model discovery is when traditional process discovery approaches fail to produce an understandable model. However, the importance of local process models is not constrained to processes where process discovery fails to produce a good model. Despite the limited number of approaches that offer local process model discovery [2,24], local process models have been used in event abstraction [18], classification of traces [20], clustering of resources [8], as sub-part of end-to-end discovery algorithms [12,17], and in different use-case studies [7,11].

In this paper, we introduce a novel approach for discovering local process models. We are inspired by region-theory discovery algorithms. We assume that the possible regions are already available to us, and instead of building one end-to-end model, we combine the regions in smaller local process models. We accept the regions in the form of place nets, that we can get from any of the existing process discovery approaches. Our proposed algorithm is available as a plugin in ProM[1] [25] (Figure 1) that allows the input to be defined as a set of place nets or a Petri net. The first notable difference between our approach and the existing ones is that we build the local process models as Petri nets instead of process trees as in [2,24]. This allows us to find constructs like long-term dependencies that are not possible in process trees. The next significant difference is speed and feasibility. We show that in contrast to the existing approaches, we are able to handle event logs with many activities or events and we return results much faster. Previous approaches rely on pruning infrequent local process models early on to gain on speed. Thus, forcing them to return only frequently appearing models. And although we are able to return frequent models, we are not constrained to find only those, since our speed arises from passing the event log only once and not pruning out infrequent patterns. In the future, this would allow for even broader usage and application of local process models in other areas of process analysis. To summarize, our contribution is threefold:

- We introduce an entirely new technique to build local process models that is completely based on Petri nets.
- We offer a technique that is feasible on event logs with many activities or events because it is linear in the size of the event log.
- We do not limit the results to frequent local process models.

We continue the paper by presenting some related work in Section 2, and preliminaries in Section 3. In Section 4, we present the approach for local process model discovery. Section 5 explains our evaluation strategy and the results we get. Section 6 concludes the paper by summarizing and giving an outlook for future work.

[1] The plugin "LocalProcessModelDiscoveryByCombiningPlaces" is available in ProM 6.11 and the Nightly Builds



(a) Table with local process models (b) Visualizing a local process model

Fig. 1. Implementation of our approach as a ProM plugin.

2 Related Work

As previously mentioned, local process model discovery is positioned in-between traditional process discovery and episode and sequence mining. Although process discovery approaches [4,6,13,15,26,27] are highly valuable for process analysis, the purpose of local process models is different. Local process models try to explain subsequences of the traces like in episode and sequence mining but can discover much more complex constructs as compared to traditional process discovery. To take advantage of all the different discovery techniques, we can use the output they produce as input for our algorithm. To be flexible regarding the existing and future discovery methods, we only require that the input is a Petri net or a set of place nets no matter how or which algorithm produced them.

To the best of our knowledge, there are two existing techniques for mining local process models from event logs. Both [24] and [2] mine local process models by recursively extending process trees.

The approach in [24] was the first to discover local process models and consists of four main steps. In the first step, for each activity in the event log a process tree containing exactly one leaf node that represents the activity is created. This set is the first set of candidate local process models. In the second step, each of the local process models from the candidate set is evaluated on the event log with different quality metrics, and only a subset of them that satisfy certain thresholds are selected in the third step. In the fourth step, the selected local process models (process trees) are expanded by replacing one of the leaves with each process tree operator (sequence, loop, parallel, and exclusive choice) and adding the replaced leaf as one child and an activity not already present in the process tree as a second child. The expanded local process models become candidates in the next step, and the procedure is repeated until the maximal size of the local process model is achieved or none of the candidates pass the selection phase. Each process tree is evaluated on the entire event log and extended with all activities, making the approach infeasible for event logs with many events or activities.

The approach in [2] was inspired by [24], and also recursively extends process trees. However, they create new process trees by combining two existing process

trees, called seeds, that differ only in one leaf node. In the combined process tree this differentiating node is replaced with a process tree operator and the two nodes are added as children to the new operator. In addition to [24] they do not reevaluate the process trees on the entire event log but on projections of the seeds. Additionally, they define compact and maximal process trees and strive to return only such local process models.

Both approaches use monotonicity of model frequency for pruning, but still struggle to return results in reasonable time on mid-sized event logs. To handle this problem, [23] extends the work in [24] by mining local process models for specific subsets of activities decided via heuristics. [22] allows for mining local process models of a specific interest using utility functions, and with [3] the work in [2] is extended to discover patterns that are frequent for a given context. Although not in the focus of this paper, our algorithm can adopt both utility functions and in-context search without impacting our running time significantly.

3 Preliminaries

In this section, we introduce important background information needed for understanding the rest of the paper. We start with some general notations, and we continue with topic-specific definitions.

General. We use sets ($\{a, b, ...\}$), multisets ($[a^2, b, ...]$), sequences ($\langle a, b, ...\rangle$), and tuples ($(a, b, ...)$) as usually defined. Given a set X, X^* represents the set of all sequences over X, and $\mathbb{M}(X)$ is the set of all multisets over X. Given a sequence $\sigma = \langle s_1, s_2, ...s_n\rangle$, we access the i-th element of the sequence with $\sigma[i]$, i.e., $\sigma[i] = s_i$, for $1 \leq i \leq n$. We extend σ with an additional element s_{n+1} by writing $\sigma \cdot s_{n+1}$. We call the sequence σ' a *subsequence* of σ, if and only if $\sigma' = \langle s_l, s_{l+1}, ...s_m\rangle$ and $1 \leq l < m \leq n$ (we write $\sigma' \sqsubseteq \sigma$ or $\sigma' = \sigma[l, m]$ if the indices are known). We call σ' a *relaxed subsequence* (we write $\sigma' \sqsubsetneq \sigma$) if and only if for some $k \geq 1$ there is $\sigma' = \langle s_{i_1}, s_{i_2}, ...s_{i_k}\rangle$ such that $1 \leq i_1 < i_2 < ... < i_k \leq n$, i.e., we drop any number of elements from σ (at most $n-1$) and keep the order for the rest. We write $\{\sigma' \text{ op } \sigma\}$ or $[\sigma' \text{ op } \sigma]$ where op $\in \{\sqsubseteq, \sqsubsetneq, \sqsubseteq_k, \sqsubsetneq_k\}$, to denote the set or multiset of all sequences σ' that satisfy the given operator in regard to σ. We use $\sqsubseteq_k$ and $\sqsubsetneq_k$ when we are interested in subsequences respectively relaxed subsequences of a particular length. To recalculate sets or multisets from other sets, multisets or sequences, we use the $\{\cdot\}$ and $[\cdot]$ operators. We use $f(X) = \{f(x)|x \in X\}$ (respectively $f(\sigma) = \langle f(s_1), f(s_2), ...f(s_n)\rangle$) to apply the function f to every element in the set X (the sequence σ) and $f_{\restriction X}$ (respectively $\sigma_{\restriction X}$) to denote the projection of the function f (respectively the sequence σ) on the set X.

Process Mining. The collected data used for process analysis is given in the form of *event logs*. Hence, in Definition 1, we formally define *traces* and *event logs*. Note that although traces are usually defined as sequences of events, in this work, we are interested only in the activity the events represent.

Definition 1. *Given the universe of activities $\mathcal{A}$, we define $\rho \in \mathcal{A}^*$ as a* trace*, and $L \in \mathbb{M}(\mathcal{A}^*)$ as an* event log*.*

In Definition 2, we define *labeled Petri nets*. Note that a transition $t \in T$ with $l(t) = \tau$ is called silent, and that there may be duplicate transitions $t_1, t_2 \in T$ such that $l(t_1) = l(t_2)$.

Definition 2 (Labeled Petri net). *A* labeled Petri net $N = (P, T, F, A, l)$ *is a tuple, where P is a set of places and T is a set of transitions such that $P \cap T = \emptyset$. $F \subseteq (P \times T) \cup (T \times P)$ is the flow relation, $A \subseteq \mathcal{A}$ is a set of activities, and $l : T \to A \cup \{\tau\}$ the labeling function.*

Now given a labeled Petri net $N = (P, T, F, A, l)$, for each element $x \in P \cup T$ we define the *preset* of x to be $\bullet x = \{y | (y, x) \in F\}$, and the *postset* of x to be $x\bullet = \{y | (x, y) \in F\}$. We additionally define the set $\overline{N} = \{(t_i, t_o) | \exists_{p \in P}((t_i, t_o) \in \bullet p \times p\bullet)\}$ to denote all pairs of transitions in the net N, that are directly connected via a place. We call each such pair a *passage*.

A labeled Petri net can be in a given state with the help of *markings*. Given a labeled Petri net $N = (P, T, F, A, l)$, we define a *marking* M as $M \in \mathbb{M}(P)$, and with $[\,]$ we denote the *empty marking*. Every element in the marking M represents a *token* in one of the places in P. The state can change by following the *firing rule*. We say that a transition $t \in T$ is *enabled* in the marking M if and only if there is a token in each place in the preset of t, i.e., $\bullet t \subseteq M$ (we write $M[t\rangle$). A transition t can *fire* in marking M if and only if it is *enabled* in M. By firing, the transition changes the marking to $M' = (M \setminus \bullet t) \cup t\bullet$. In this case, we can write $M \xrightarrow{t} M'$. To denote getting from M to M' by firing a sequence of transitions $\sigma = \langle t_1, ..., t_n \rangle \in T^*$ such that $M \xrightarrow{t_1} M_1 \xrightarrow{t_2} M_2 \cdots \xrightarrow{t_n} M'$, we write $M \xrightarrow{\sigma} M'$.

In Definition 3, we define a *union* of two labeled Petri nets and we extend for multiple labeled Petri nets. Then in Definition 4 we define what it means for a labeled Petri net to be *connected*.

Definition 3 (Union of labeled Petri nets). *Given two labeled Petri nets $N_1 = (P_1, T_1, F_1, A_1, l_1)$ and $N_2 = (P_2, T_2, F_2, A_2, l_2)$ we define their* union *as $N_1 \cup N_2 = N = (P, T, F, A, l)$ where $P = P_1 \cup P_2$, $T = T_1 \cup T_2$, $F = F_1 \cup F_2$, $A = A_1 \cup A_2$, and $l : T \to A \cup \{\tau, \bot\}$ is the mapping* $l(t) = \begin{cases} l_1(t), \text{ if } t \in T_1 \setminus T_2 \\ l_2(t), \text{ if } t \in T_2 \setminus T_1 \\ l_1(t), \text{ if } t \in T_1 \cap T_2 \wedge l_1(t) = l_2(t) \\ \bot, \quad \text{otherwise} \end{cases}$.
The union is a valid union *if there is no $t \in T$ such that $l(t) = \bot$. We write $\bigcup_{i=1}^{n} N_i = (\cdots((N_1 \cup N_2) \cup N_3) \cdots \cup N_n)$ to denote the* union *of the set of labeled Petri nets $\{N_1, \ldots, N_n\}$.*

Definition 4 (Connected labeled Petri net). *A labeled Petri net $N = (P, T, F, A, l)$ is* connected*, if and only if for each two different elements $x, x' \in P \cup T$ there exists a sequence $\langle y_1, \ldots, y_n \rangle$ such that $n \geq 2$, $(y_i, y_{i+1}) \in F$ or $(y_{i+1}, y_i) \in F$ for $1 \leq i < n$ and $y_1 = x$ and $y_n = x'$.*

Local Process Models. Our algorithm discovers a set of *local process models*, that we represent with labeled Petri nets. We discover these *local process models* from an *event log* and a *set of place nets*. To represent our input, in Definition 5 we define a *place net* as a labeled Petri net with only one place.

Definition 5 (Place net). *A* place net *is a labeled Petri net* $N_p = (\{p\}, T, F, A, l)$*, where* $\{p\}$ *is a set of places containing one place only, and* T *is a set of transitions such that* $P \cap T = \emptyset$*.* $F \subseteq (\{p\} \times T) \cup (T \times \{p\})$ *is the flow relation,* $A \subseteq \mathcal{A}$ *is a set of activities, and* $l : T \to A \cup \{\tau\}$ *the labeling function.*

Next, with the help of Definitions 3 and 4, in Definition 6 we define a *local process model* as a *union* of place nets.

Definition 6 (Local Process Model). *Given a set of* place nets $N_{p_i} = (\{p_i\}, T_i, F_i, A_i, l_i)$ *for* $1 \leq i \leq k$*, their union is a* local process model*,* $LPM = \bigcup_{i=1}^{k} N_{p_i}$*, if and only if* LPM *is a* valid union *and a* connected *labeled Petri net.*

What makes our *local process models* local is the behavior. Therefore, we define the term *locality* or *local distance* to be the maximal length of the trace's subsequences we want our local process models to explain. Since *local process models* are a subset of labeled Petri nets, *markings*, *enabled transitions* and *firing rule*, also hold for them. The opportunity to change states and fire transitions makes it possible *local process models* to describe behavior. Since we want to discover models that explain selected parts of the behavior in an event log, we need to somehow align the two. Thus, in Definition 7 we define how a local process model can replay a sequence of activities. In addition, we want to be able to skip some of the activities during the replay, so we also define *relax replay* (Definition 8).

Definition 7 (Replay). *Given a local process model* $LPM = (P, T, F, A, l)$ *and a sequence of activities* $\rho = \langle a_1, a_2, \ldots a_n \rangle$*, we say* LPM replays ρ *if and only if there exists a sequence of transitions* $\sigma = \langle t_1, t_2, \ldots, t_m \rangle \in T^*$ *such that* $l(\sigma)_{\restriction A} = \rho$ *and* $[\,] \xrightarrow{\sigma} [\,]$*.*

Definition 8 (Relaxed Replay). *Given a local process model* $LPM = (P, T, F, A, l)$ *and a sequence of activities* $\rho = \langle a_1, a_2, \ldots a_n \rangle$*, we say* LPM relax replays ρ *if and only if there exists at least one relaxed subsequence* $\rho' \in [\rho' \sqsubseteq \rho]$ *that* LPM *can* replay*.*

By defining *replay* and *relax replay* to require starting and ending in an empty marking, makes the subset of *place nets* $N_p = (\{p\}, T, F, A, l)$ for which $\bullet p \subseteq p\bullet$ or $p\bullet \subseteq \bullet p$ unsuitable for our *local process models*. Hence, in the continuation we will discard *place nets* of this type.

In addition, we use *replay* and *relax replay* to define the *language* (Definition 9) and *relaxed language* (Definition 10) for a given local process model LPM.

Definition 9 (Language). *Given a local process model* $LPM = (P, T, F, A, l)$*, we define* $\mathcal{L}(LPM) = \{\rho \in A^* | \exists_{\sigma \in T^*}(l(\sigma)_{\restriction A} = \rho \wedge [\,] \xrightarrow{\sigma} [\,])\}$ *to be the* language *of* LPM*.*

Definition 10 (Relaxed Language). *Given a local process model* $LPM = (P, T, F, A, l)$*, we define the* relaxed language *of* LPM *as* $\mathcal{L}_{rlx}(LPM) = \{\rho \in A^* | \exists_{\rho' \in \mathcal{L}(LPM)}(\rho' \sqsubseteq \rho)\}$*.*

We conclude this section, by defining how a local process model can be *compact* in regard to a sequence of activities (Definition 11).

Definition 11 (Compact Local Process Model). *Given a local process model* $LPM = (P, T, F, A, l)$ *and a sequence of activities* $\rho = \langle a_1, a_2, ..., a_n \rangle \in \mathcal{L}_{rlx}(LPM)$*, we say* LPM *is compact with respect to* ρ *if and only if it holds that* $\exists_{\sigma \in T^*}([\,] \xrightarrow{\sigma} [\,] \wedge l(\sigma)_{\restriction A} \sqsubseteq \rho \wedge \forall_{p \in P}(\exists_{t \in \{\sigma\}}(p \in \bullet t \cup t\bullet)))$*.*

4 Approach

Our algorithm combines place nets into local process models. Hence, as input we require place nets and an event log for which we want to build the local process models. However, for n place nets, there are $2^n - 1$ non-empty candidate local process models. Even if we remove the ones that do not satisfy Definition 6, our search space would still be enormous. Additionally, some of the local process models we build, can be too complicated or not satisfy basic quality expectations. Therefore, we propose a framework with three modules (Figure 2). Since our search space directly depends on the number of place nets we use, we use the first module for filtering and adapting the place nets to limit their number. However, at the same time the quality of the built local process models directly depends on the quality of the chosen place nets, so we want to choose these wisely. After the place nets are chosen, the second module introduces the main algorithm for building local process models. The goal of the algorithm is to consider different subsets of place nets, construct their union and check whether it can relax replay subsequences of the traces in the given event log. Although we restrict the set of place nets we use, we can still end up with a lot of local process models. Therefore, we also provide a module for evaluating and ranking the found local process models with different metrics. In the following, we introduce each of the modules, with the main focus on the combination algorithm (the second module in Figure 2).

4.1 Place net Adaptation and Filtering (PAF)

We use an "oracle" to get the place nets from which we build our local process models. Any algorithm that returns a labeled Petri net or a set of place nets based on an event log can be considered an oracle. The oracle can return many place nets, so for efficiency reasons, we want to limit the number of those we use for building local process models. On the other side, the set of place nets we

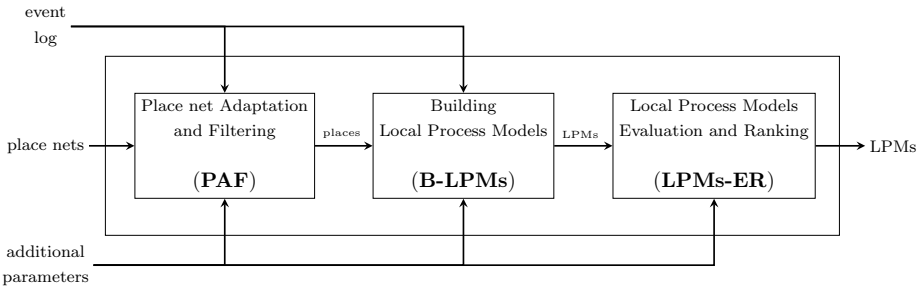


Fig. 2. Top view of our framework for local process model discovery

use restricts our models to a representational bias defined by them. Hence, to promote simplicity and higher relax replay frequency (see Definition 8), we rank the place nets by giving preference to nets with fewer transitions that can relax replay more subsequences. For place nets that rank the same on the previous criteria, we use the lexicographic order of the included transitions. Afterward, we keep the k highest ranking place nets to build local process models, where k is a user-defined parameter. For simplicity in the continuation of the paper, we assume that given the set of place nets P that we return now $\bigcup P$ is a valid union. Otherwise, we keep track of all the subsets where label disagreements exist and do not use multiple contradicting place nets in one local process model.

4.2 Building Local Process Models (B-LPMs)

This module covers the part of the framework that combines place nets into local process models, thus, making it the main contribution of this paper. To explain the approach, we give a high-level pseudo-code in Algorithm 1. There are three main steps that make up the gist of the algorithm, and get us from a set of place nets, to a set of local process models that describe the event log:

1. Focus on locality by iterating all subsequences in the event log of certain length (Line 2).
2. Build local process models for each subsequence separately (Line 3).
3. Store the built local process models in a single structure (Line 4).

The high-level algorithm looks pretty straightforward. However, optimizing the *traversal* of the event log on line 2, the particulars of the *global storage* and how we *create* local process models that relax replay the window, is what makes the algorithm not only feasible but also efficient.

Focus on locality. We want our local process models to describe what happens within some local distance in the event log. Hence, with a *sliding window* we get subsequences of certain length, that we call *windows*. The sliding window size represents the *locality* we are interested in, and we accept it as an input parameter. We formally define the sliding window in Definition 12.

Algorithm 1: Combining Places in Local Process Models

input : L - event log; d - local distance; P - set of place nets;
output: LPM - set of local process models

1 $LPM \leftarrow [\,]$; // initialize the global storage
2 **forall** $w \in [\rho' \sqsubseteq_d \rho | \rho \in L]$ **do**
  // for each subsequence of L of length d find subsets of P that relax replay w and satisfy some additional constraints AC (e.g., compactness)
3 $lpms \leftarrow \{\bigcup P' | P' \subseteq P \wedge w \in \mathcal{L}_{rlx}(\bigcup P') \wedge AC(\bigcup P', w)\}$;
4 $LPM \leftarrow LPM \cup lpms$; // add lpms to the global storage

Definition 12 (Sliding Window). *Given a trace* $\rho = \langle a_1, a_2, ..., a_n \rangle$ *and locality* $d > 0$*, we define the function* $W_d(i, \rho) = \begin{cases} \rho[i, i+d-1], & \text{if } 1 \leq i \leq n-d+1 \\ \langle\rangle, & \text{otherwise} \end{cases}$ *to be a* sliding window*. Each generated subsequence for a concrete* i *and* ρ *we call a* window*.*

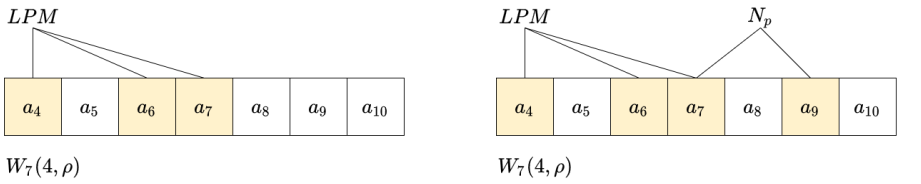
The sliding window helps us to iterate the event log, and focus on a local level. However, for each window, we need to efficiently and exhaustively (considering our representational bias and limitations) combine places into local process models that can relax replay that window.

Building local process models for one window. At this point, we have our set of place nets $P = \{N_{p_1}, N_{p_2}, \ldots, N_{p_k}\}$ and a sequence of activities, i.e., our window w. Our goal is to find subsets of P, $P' \subseteq P$, to form local process models, $LPM = \bigcup P'$, that satisfy Definition 6 considering the following constraints:

- LPM can *relax replay* w ($w \in \mathcal{L}_{rlx}(LPM)$)
- LPM is *compact* in regard to w (see Definition 11)

Additionally, we want to be time efficient. Therefore, given a trace $\rho = \langle a_1, a_2, \ldots, a_n \rangle$ we consider that two consecutive windows $W_d(m, \rho)$ and $W_d(m+1, \rho)$, share $d-1$ of their elements. The models found for this overlapping sequence shared by both windows, are the same. Hence, it is important that we do not recalculate these models, which in turn defines the goal to *reuse* local process models shared between consecutive windows.

Idea. The core idea is to create new local process models by extending existing ones with an additional place net such that the relaxed subsequence of the window that they can replay increases in length. We start with the empty local process model that can somehow replay the empty trace, and we want to extend it with carefully selected place nets such that two activities from the window can be replayed. In the next step, we would extend those local process models by adding an additional place net such that the newly created local process models can replay three of the activities in the window. We continue as long as there



(a) Existing local process model LPM (b) New local process model $LPM \cup N_p$

Fig. 3. Extension of a local process model with a new place net.

are still unprocessed activities in the window. For example, let us consider the window $W_7(4, \rho)$ in Figure 3a where ρ is a trace. We have built a local process model LPM such that $\langle a_4, a_6, a_7 \rangle \in \mathcal{L}(LPM)$[2]. Since $\langle a_4, a_6, a_7 \rangle \sqsubseteq_{\approx} W_7(4, \rho)$, $W_7(4, \rho) \in \mathcal{L}_{rlx}(LPM)$. We now want to extend LPM with an additional place net $N_p = (\{p\}, T_p, F_p, A_p, l_p)$. What is specific for N_p is that it should be able to replay a_7 such that a token is put in p, and also replay one of the unprocessed activities (a_8, a_9 or a_{10}) such that the token is removed from p. Hence, the newly built local process model $LPM \cup N_p$ is empty after replaying four activities from the window, and the used firing sequence is an extension of the firing sequence used for replaying $\langle a_4, a_6, a_7 \rangle$ on LPM. In our case the new activity is a_9 and we visualize this in Figure 3b. To know whether we can extend LPM with N_p we have to check that we do not break the replay of $\langle a_4, a_6 \rangle$. Hence, we need the firing sequence σ for which we replayed $\langle a_4, a_6 \rangle$, to ensure that σ can still fire when the new place net is added. To know where to connect the place net and the local process model such that a_7 can be replayed we need the marking M after firing σ i.e., $[\,] \xrightarrow{\sigma} M$. At the end, we also store the two indices $indIn$ and $indOut$ in the window for which the last extension happened. Note that $\sigma_{\uparrow_A} \sqsubseteq_{\approx} w[1, indIn]$ and $w[1, indOut] \in \mathcal{L}_{rlx}(LPM)$.

Algorithm. We now present an algorithm that builds local process models given a set of place nets P and a window w. We explained that at every step we extend existing local process models with new place nets. To be aware of the extension path from which we got to a particular local process model and how to continue extending it, we organize the local process models in a tree structure that we call *local tree*. Each node in the *local tree* represents a local process model $LPM = (P, T, F, A, l)$ that can relax replay w. There is an edge between two nodes n and n' when the local process model represented by n' was built by extending the local process model in n with an additional place net. We formally define our *local tree* in Definition 13.

Definition 13 (Local Tree). *A* local tree $LT = (N, E)$ *is a pair, where* N *is a set of nodes and* E *a set of edges such that:*

- *A node* $n = (LPM, \sigma, M, indIn, indOut)$ *is a tuple, where* $LPM = (P, T, F, A, l)$ *is a local process model,* $\sigma \in T^*$ *is a sequence*

[2] Note that this doesn't have to be the only one such local process model.

of transitions, $M \in \mathbb{M}(P)$ *is the marking* $[\,] \xrightarrow{\sigma} M$*, and* $indIn, indOut \in \mathbb{N}$ *are the indices for which the last extension happened.*
- *An edge* $e = (n, n')$ *is a pair of nodes.*

In Algorithm 2 we give the pseudo-code of the entire procedure. As input we are given the *set of place nets* P and the *window* w. We start by initializing the local tree to contain only a root node that represents the empty local process model (line 1). Then we traverse all activity pairs of the window, and for each pair, we extend existing local process models in LT with additional place nets. We get suitable place nets by filtering those that can replay the currently considered two events $w[i]$ and $w[j]$, and suitable nodes by filtering those that contain a local process model in a marking in which $w[i]$ can be replayed (lines 5 and 6). Afterward, we restrict that local process models are extended with a place net only if the place net does not add a new constraint on an already used transition (line 9). Then, we find a common transition of the place net and the local process model that can replay $w[i]$. If there are no such transitions and the node is not the root, the extension can not happen (line 12). If there are multiple such transitions we randomly choose one (line 14). We create a new node n' (line 15) that represents the local process model built by adding the place net N_p to the local process model in the node n, in a marking after replaying $w[i]$. We add the newly created node in the tree and connect it with the node from which it was created (lines 16 and 17). We finish by adding the local process model to the final set if after replay of $w[j]$ we end in the empty marking (lines 18 and 19).

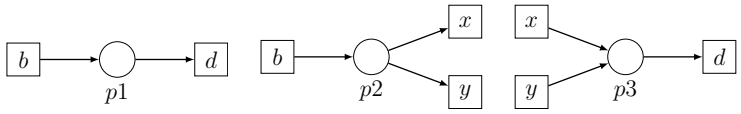
Fulfillment of Constraints and Goals. In the following, we give some intuitions that connect the design of the algorithm to the constraints and the goal. The first constraint is that each returned local process model satisfies Definition 6. In the PAF module we assumed that $\bigcup P$ is a valid union. Hence, the union of any subset $P' \subseteq P$ is also a valid union. That the local process model is connected is satisfied by requiring $T' \neq \emptyset$ when the place net we add is not the first in the local process model (line 12). The constraint that each created local process model can relax replay w is satisfied by combining lines 5, 6, 9 and 18. The filterings of the nodes and place nets, ensure that a local process model is extended with a new place net only when the newly created local process model replays one more activity of the window than its base local process model. In line 9 we make sure we do not break the successful replay of the base local process model, and with line 18 we make sure that there is at least one unprocessed activity in the window, after whose replay the local process model ends in an empty marking. Definition 11 is also satisfied because of the filtering in line 5. A place net is added to a local process model only if a token can be put in the place it represents and removed from it, by replaying two activities. Therefore each place is marked at some point of the replay. Finally, our goal to *reuse* local process models between consecutive windows, is satisfied by the way we organize our nodes in the tree, i.e., how we create edges (line 17). Given two nodes n and n' such that $(n, n') \in LT.E$, we know that $n.LPM.l(n.\sigma) = n'.LPM.l(n'.\sigma)[1, |n'.\sigma| - 1]$. Hence, the most distant ancestor of n' apart from the root, is some node $n*$ that

Algorithm 2: Building Local Process Models for a Window

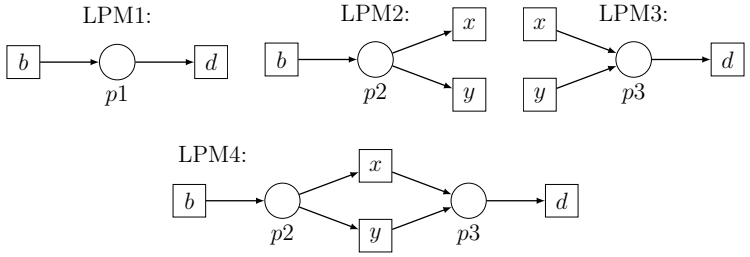
input : w - window; P - set of place nets;
output: $LPMs$ - set of local process models
// In the pseudo-code we use a dot notation for accessing elements of an object, similar as in object-oriented programming.
1 $LT \leftarrow (N = \{root = (\emptyset, \langle\rangle, [\,], 0, 0)\}, E = \emptyset)$; // initialize the storage
2 $d = |w|$; // length of the window
3 **for** $j \leftarrow 1$ **to** d **do**
4 **for** $i \leftarrow 1$ **to** $j - 1$ **do**
// for each pair of events get suitable place nets and nodes
5 $P' \leftarrow \{N_p \in P | \langle w[i], w[j] \rangle \in \mathcal{L}(N_p)\}$;
6 $N' \leftarrow \{n \in N | n.LPM.l(n.\sigma)_{\restriction_{n.LPM.A}} \cdot w[i] \in \mathcal{L}(n.LPM)\} \cup \{root\}$
// try to extend LPM in each node with each place net
7 **for** $n = (LPM, \sigma, M, indIn, indOut) \in N'$ **do**
8 **for** $N_p = (\{p\}, T_p, F_p, A_p, l_p) \in P'$ **do**
9 **if** $p\bullet \cap \{\sigma\} \neq \emptyset$ **then**
10 **continue**; // no new constraint
11 $T' \leftarrow \{t' \in LPM.T \cap \bullet p | LPM.M[t'\rangle \wedge LPM.l(t') = w[i]\}$;
12 **if** $n \neq root \wedge T' = \emptyset$ **then**
13 **continue**; // no common transition
14 $t \leftarrow_R T'$ // choose any transition
15 $n' \leftarrow (LPM \cup N_p, \sigma \cdot t, (M \setminus \bullet t) \cup t\bullet, i, j)$; // create node
16 $LT.N \leftarrow LT.N \cup n'$; // add node
17 $LT.E \leftarrow LT.E \cup (n, n')$; // add edge
// add $n'.LPM$ in final set if $w \in \mathcal{L}_{rlx}(n'.LPM)$
18 **if** $\exists_{t \in n'.LPM.T}(n'.M \xrightarrow{t} [\,] \wedge n'.LPM.l(t) = w[j])$ **then**
19 $LPMs \leftarrow LPMs \cup \{n'.LPM\}$;
20 **return** $LPMs$

is a child of the root. Then, $n*.\sigma = \langle n'.\sigma[1]\rangle$. Therefore, if we want to remove all local process models that replay $w[1]$, we just need to remove all children of the root $n*$ for which $n*.LPM.l(n*.\sigma[1]) = w[1]$ (have in mind that $|n*.\sigma| = 1$ for the children of the root).

Example. To clarify how the algorithm works given its input, we additionally provide an example. For simplicity we assume that $t = l(t)$ for each transition. Given the set of place nets P (see Figure 4a) and the window $w = \langle b, a, x, a, d\rangle$ we build local process models by following Algorithm 2. We first initialize the local tree $LT = (\{root\}, \emptyset)$ and the resulting set $LPMs = \emptyset$. Then we iterate through the window with the indices i and j. We start with $i = 1$ and $j = 2$. Since $w[i] = b$ and $w[j] = a$, we get $P' = \emptyset$ so we continue. For $i = 1$ and $j = 3$ ($w[i] = b$ and $w[j] = x$) we filter $P' = \{N_{p2}\}$, $N' = \{root\}$. Since $p2\bullet \cap \{root.\sigma\} = \emptyset$ and $T' = \{b\}$, we create the node $n1 = (\{N_{p2}\}, \langle b\rangle, [p2^1], 1, 3)$ and add it as child to the root node ($LT = (\{root, n1\}, \{(root, n1)\})$). Because $[p2^1] \xrightarrow{x} [\,]$ and $w[j] = x$ we add the local process model to the final set ($LPMs = \{\{N_{p2}\}\}$). We skip $i = 2$, $j = 3$ and all pairs for $j = 4$, since $P' = \emptyset$ for them. For $i = 1$



(a) Example place nets



(b) Found local process models

Fig. 4. Place nets and Local Process Models for the example

and $j = 5$ we calculate $P' = \{N_{p1}\}$, $N' = \{root\}$. Given $p1 \bullet \cap \{root.\sigma\} = \emptyset$ and $T' = \{b\}$, we create the node $n2 = (\{N_{p1}\}, \langle b \rangle, [p1^1], 1, 5)$, add it as child to the root node and to the final set ($LT = (\{root, n1, n2\}, \{(root, n1), (root, n2)\})$ and $LPMs = \{\{N_{p2}\}, \{N_{p1}\}\}$). We again skip $i = 2$, $j = 5$ since $P' = \emptyset$. For $i = 3$ and $j = 5$ we calculate $P' = \{N_{p3}\}$, $N' = \{root, n1\}$. For N_{p3} and $root$, $p3 \bullet \cap \{root.\sigma\} = \emptyset$ and $T' = \{x\}$ so we create $n3 = (\{N_{p3}\}, \langle x \rangle, [p3^1], 3, 5)$. For N_{p3} and $n1$, $p3 \bullet \cap \{n1.\sigma\} = \emptyset$ and $T' = \{x\}$ so we create $n4 = (\bigcup\{N_{p2}, N_{p3}\}, \langle b, x \rangle, [p3^1], 3, 5)$. We add $n3$ as child to the root node and $n4$ as child to $n1$. Our local tree now is $LT = (\{root, n1, n2, n3, n4\}, \{(root, n1), (root, n2), (root, n3), (n1, n4)\})$ and final set $LPMs = \{\{N_{p2}\}, \{N_{p1}\}, \{N_{p3}\}, \bigcup\{N_{p2}, N_{p3}\}\}$. We do nothing for $i = 4$ and $j = 5$ since $P' = \emptyset$. The final set $LPMs$ is given in Figure 4b.

Choice and Concurrency. After processing the window w, the tree contains all local process models LPM for which $w \in \mathcal{L}_{rlx}(LPM)$ and given the used firing sequence σ it holds that $\forall_{t \in \sigma}(M \xrightarrow{t} M' \implies M \cap M' = \emptyset)$, i.e., concurrency is not considered. To build the concurrency constructs, we combine nodes from different branches in the local tree and take the union of the local process models that the nodes contain. The number of transitions that can be concurrent directly depends of the number of nodes we combine. To avoid an explosion of possibilities, the number of concurrent transitions can not be too large since we try all possible node combinations. Because of the place nets, the choice construct is embedded in our input, so no additional processing is needed.

Silent and Duplicate Transitions. The presented algorithm handles the duplicate transitions as all other transitions. However, in the case of silent transitions we convert the set of place nets to a set of paths. Each path is a valid and connected union of one or multiple place nets connected via silent transitions. Then, on

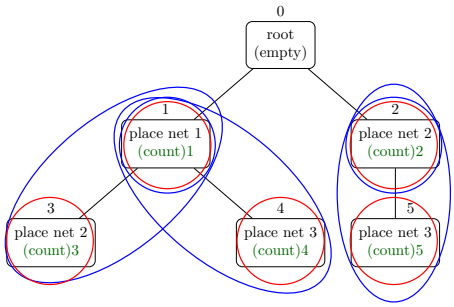


Fig. 5. Global Tree Structure. With red, we denote the place nets in the global tree, blue the local process models, and the count in green is for the number of windows the local process model (starting in the root and ending in that node) can relax replay.

line 5 we check whether the sequence consisted of the two activities, is in the language of the path and the path is compact for the sequence.

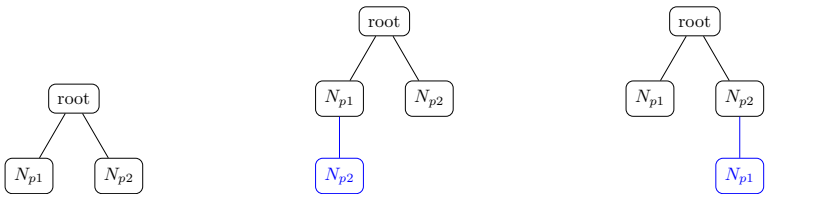
Collecting local process models on a global level. The local process models we want to store in the global storage are just sets of place nets. Hence, to represent them efficiently, we use a tree structure as shown in Figure 5. Every node in the tree stores one place net. At the same time each node also represents exactly one local process model by taking the union of the place nets in the path from that node to the root. Hence, in each node we also keep the number of windows the corresponding local process model can relax replay. Any additional information about the local process model that we might want to store in the future, can be stored in the same way as the relax replay count.

Structuring the tree this way we share place nets between the stored local process models. To also make the structure efficiently extendable, we want each path in the tree to represent a unique local process model. Therefore, we introduce a *rank* function. The rank function $rank : P \mapsto \mathbb{N}$ gives priority to each place net which in turn determines the order in which the place nets appear in the tree path representing the local process model. In Figure 6 we illustrate the problem when a local process model $\bigcup\{N_{p1}, N_{p2}\}$ needs to be added to the tree in Figure 6a.

After processing each window, we add all discovered local process models to the global storage. At the end, after processing all windows, the tree will contain each local process model we find together with the number of windows each local process model can relax replay.

4.3 Local Process Models Evaluation and Ranking (LPMs-ER)

Our exhaustive search can end up in a large number of local process models. Hence, we need to limit the number of local process models we return and first show the ones we classify as more relevant. One simple restriction is to limit the



(a) Initial global tree (b) Adding N_{p2} to N_{p1} (c) Adding N_{p1} to N_{p2}

Fig. 6. Difficulty in the global tree for adding the same local process models multiple times when we do not use a rank function.

minimal and maximal number of places and transitions a local process model can have.

To measure the quality of our local process models we propose different heuristics. All metrics are calculated for an event log L, a local distance d, and a local process model $LPM = (P, T, F, A, l)$. With $W^L = [w \in \bigcup_{\rho \in L} [\rho' \sqsubseteq_d \rho]]$ we define the multiset of all windows in L with length d, and with $S^L_{LPM} = [s \in [w' \sqsubseteq w] | w \in W^L \wedge s \in \mathcal{L}(LPM)]$ a multiset of the sequences replayed by LPM during relax replay of the windows.

- *Fitting windows evaluation* calculates the fraction of windows a local process model can relax replay (Equation (1)).

$$fw(LPM, L) = \frac{|\{w \in W^L | w \in \mathcal{L}_{rlx}(LPM)\}|}{|W^L|} \tag{1}$$

  This metric is in a way an adaptation for calculating fitness for the local process models. We never expect one local process model to explain the entire event log, so to make the metric comparable, we compare the values to the best scoring local process model.
- *Passage coverage evaluation* calculates the fraction of the passages used in the relax replay of the fitting windows (Equation (2)).

$$pc(LPM, L) = \frac{|\{(t_1,t_2) \in \overline{LPM} | \exists_{s \in S^L_{LPM}} (\exists_{i \in \{1,\ldots,|s|-1\}} (s_i = l(t_1) \wedge s_{i+1} = l(t_2)))\}|}{|\overline{LPM}|} \tag{2}$$

  The values are in the interval $(0, 1]$, where we get 1 when all the local process model passages are used at least once. This metric is similar to precision since lower values mean that the local process model allows more behavior than seen in the event log.
- *Passage repetition evaluation* calculates whether multiple place nets of the local process model contain the same passages (Equation (3)). We define $\#_{(t_1,t_2)} = |\{p \in P | (t_1, t_2) \in \bullet p \times p\bullet\}$ to be the number of place nets in LPM that have the passage (t_1, t_2).

$$pr(LPM, L) = \frac{|LPM| \cdot |\overline{LPM}| - \sum_{(t_1,t_2) \in \overline{LPM}} \#_{(t_1,t_2)}}{|LPM| \cdot |\overline{LPM}| - |\overline{LPM}|} \tag{3}$$

Table 1. Information about the event logs used in our analysis

Event log alias	Trace variants count	Activities count	Total event count
BPIC2012 [9]	4366	24	182467
BPIC2019 [10]	11973	42	338247
RTFM [14]	231	13	2353
Sepsis [16]	846	16	13775
Artificial Small	2	7	45
Artificial Big	96	13	1624

This metric tries to express the simplicity of the local process model. The value of 1 denotes that each passage is contained by only one place net, and 0 denotes that all passages are contained in all place nets.

– *Transition coverage evaluation* calculates in how many of the relax replayed windows in which a transition t can be used, that transition is actually used during the replay. The average value over all transitions is returned. (Equation (4)).

$$tc(LPM, L) = \frac{1}{|T|} \cdot \sum_{t \in T} \frac{|[s \in S^L_{LPM} | \exists_{i \in \{1,\dots,|s|\}}(s[i]{=}l(t))]|}{|[w \in W^L | w \in \mathcal{L}_{rlx}(LPM) \wedge \exists_{i \in \{1,\dots,|w|\}}(w[i]{=}l(t))]|} \quad (4)$$

The values for the metric are in the interval $(0, 1]$. Low values indicate that we use only a few transitions in our local process model during the relax replay, meaning our model is more complex than necessary.

We finish by ranking the found local process models, by taking the average score of the presented evaluation metrics. The higher the average score, the better the rank of the local process model.

5 Evaluation and Results

In this section, we evaluate our method on real and artificial event logs (see Table 1). We split the evaluation into several parts. We start by discussing how quality is defined and measured for local process models, and the challenges around it. Then, we compare the results of our algorithm with several process discovery approaches and presented related work on a specific event log. Afterward, we present the running time our algorithm has on different event logs, and the effect different parameters have on it. We end the evaluation section by comparing the running time with the running time of existing approaches discussed in related work [2,24]. For all experiments, we use the plugin we implemented in ProM and the eST Miner [15] as a place oracle. To allow for reproducibility of the experiments, we provide the artificial event logs and the sets of place nets we use at https://github.com/VikiPeeva/PlacesAndEventLogs.

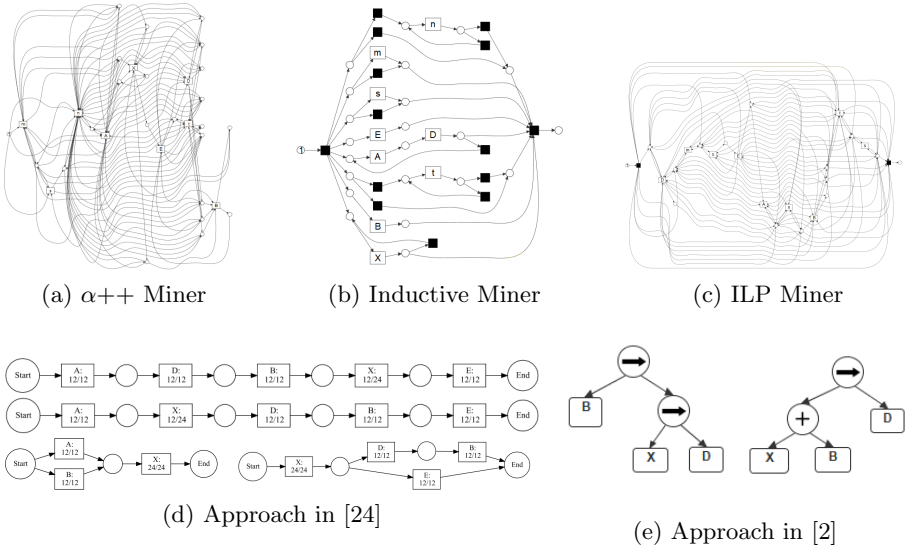
5.1 Quality Definition and Challenges

Calculating the quality of local process models is challenging because of all the different ways it can be looked at. From one side we can look at the quality

of each individually returned local process model or the quality of all of them as a group. If we use local process model discovery when traditional process discovery fails, the desired result would be a minimal set of local process models that cover the entire event log with as little overlaps between them as possible. This is discussed in [5] where one event log is analyzed by hand and compared to the results from [24]. Both [24] and [2], nor their future work offer this as a possibility and neither our algorithm. However, as discussed in the introduction, that is not the only usage of local process models. If we are interested in what happens when patients are cured, when companies lose money, when employees resign, etc., then we might be interested in finding local process models in regard to some utility functions or different contexts. This is to some degree investigated in [22] and [3] accordingly. Our work, currently does not support this type of local process mining, however, it is orthogonal to the current work, and can be integrated in the algorithm. With the previous information in mind we see how challenging is to give quality comparison on hundreds returned local process models between different approaches, especially when the most straight-forward comparison - event log coverage - is not available for any of them. Hence, for us, the goal was the new approach we propose to be more feasible than the existing ones in regard to running time and number of local process models found, and extendable towards event log coverage and utility mining.

5.2 Discoverability of Constructs

To illustrate the need for local process model discovery, and why the approaches proposed in [24] and [2] are not enough, we give an event log whose traces are generated by repeating the pattern $AXDBXE$. We additionally add noise (from the alphabet l, m, n) between the different occurrences of the pattern and in smaller amount in-between the pattern itself. An example trace in such event log would be $\langle m, A, X, n, D, B, X, E, l \rangle$. We ran the generated event log with α++ miner [26], inductive miner [13], ILP miner [27], the local process model discovery approaches proposed in [24] and [2], and the approach proposed in this paper. We present the results we get in Figure 7. ILP and α++ miner returned a spaghetti-like models, while the inductive miner returned a model with mostly flowery behavior. In none of these models the pattern is clearly visible. The approaches in [24] and [2] although returning local process models that represent parts of the pattern, are not able to return a local process model that describes the pattern accurately. In contrast, our algorithm finds a local process model that completely describes the pattern (Figure 1b) in addition to the other local process models that we find. By finding this model we show that we are able to skip in-between noise, and that we can discover constructs like long-term dependencies which the approaches in [2] and [24] cannot because of the representational bias of process trees.



(a) α++ Miner (b) Inductive Miner (c) ILP Miner

(d) Approach in [24] (e) Approach in [2]

Fig. 7. Process models for an event log focusing on the pattern $AXDBXE$.

5.3 Running Time vs Parameters

The main parameters that we can control are the number of place nets we use and the size of the local distance. Other important parameter is the cardinality of the concurrency, i.e, what is the maximal number of transitions we allow to be in a concurrent construct. Hence, we show diagrams to see how these parameters affect the running time of the algorithm.

In Figure 8, we show the running time for place net counts of $[50, 75]$ and locality of $[5, 7, 10, 12]$. As expected, given a fixed amount of place nets used, the running time increases as the locality increases, and also the other way around, given a fixed locality, the running time increases as the number of place nets used increases. We can notice that for 50 place nets the algorithm finishes in less than five minutes for all event logs and different localities except for *BPIC2019* and locality 12. However, when considering 75 place nets, for all event logs except *Artificial Small* the limit of ten minutes is reached at locality 12. What is interesting to see is that both *Artificial Big* and *RTFM* have a larger running time for place net count of 75 and localities 5, 7 and 10 than *BPIC2012* and *BPIC2019* although the latter are much larger event logs, both in the number of events and number of activities they contain (see Table 1). This shows the impact the linearity of our algorithm has in regard to the size of the event log, and the importance of how we choose which place nets to use.

Regarding our concurrency cardinality parameter, we see in Figure 9 that we are able to handle concurrency constructs with 4 transitions for 50 place nets, and 3 transitions for 75 place nets. However, we notice that by adding the possibility for just one more transition, the running time exceeds 10 minutes.

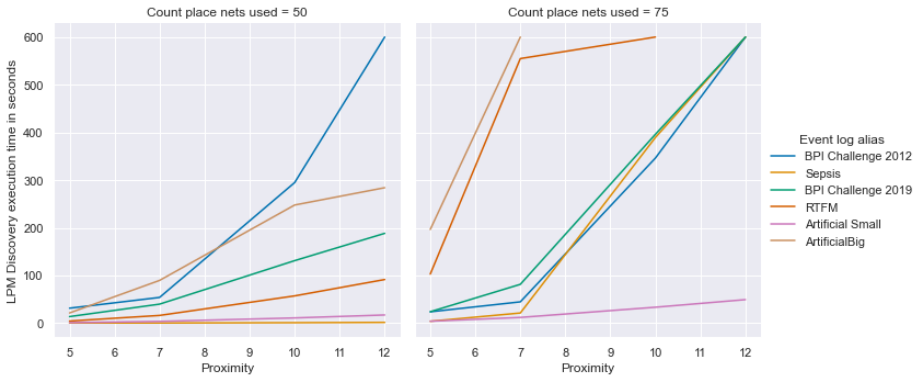


Fig. 8. Diagram that shows the effect different settings for the count of place nets and local distance have on the running time.

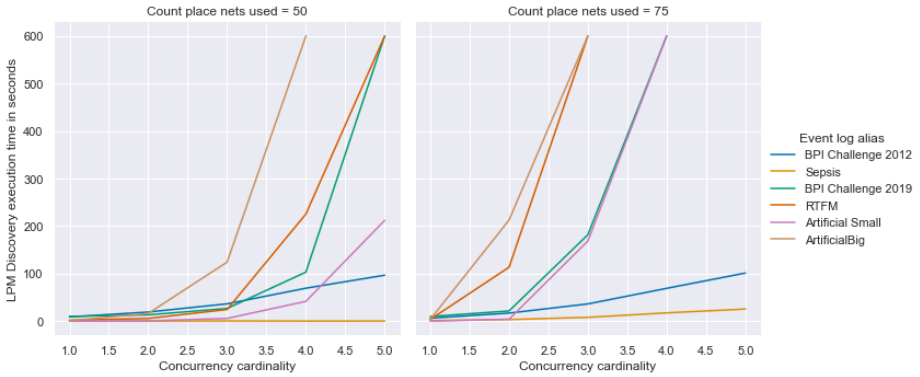


Fig. 9. Diagram that shows how the concurrency cardinality parameter affects the running time.

5.4 Comparison to other approaches

In this section, we focus on the comparison of our approach to the ones presented in [24] and [2] in regard to the running time. We run all algorithms on real and artificial event logs with time limit of 10 minutes on a PC with i7-1.8GHz, 16GB RAM and Windows 10. We use the provided default settings of the plugins where for the approach of Tax et al. the default settings also include the log projections explained in [23]. The only setting we vary for our algorithm is the number of places used (50, 75 and 100). We present the results at Table 2. We see that for the artificial event logs our approach is comparable in the time needed to return results to the one in [24] when we use 50 places. However, when it comes to real event logs, our approach is notably faster than the other two. For example, on the *BPIC2019* event log the other approaches do not return results at all because of memory problems, while we are able to build a large amount of local

Table 2. Results comparison to [24] and [2]

Event log	Our approach			Approach in [24]		Approach in [2]	
	#places	runtime	#LPMs	runtime	#LPMs	runtime	#LPMs
BPIC2012	50	4s	284	90s	454	out of time	
	75	20s	2473				
	100	**23s**	**6484**				
BPIC2019	50	28s	3190	out of memory		out of memory	
	75	**48s**	**7617**				
	100	out of time	/				
RTFM	50	15s	8967	out of time		out of time	
	75	**368s**	**90862**				
	100	out of time	/				
Sepsis	50	2s	18	56s	4627	125s	375
	75	4s	3384				
	100	**22s**	**14951**				
Artificial Big	50	40s	8979	**70s**	**56110**	out of time	
	75	536s	65383				
	100	out of time	/				
Artificial Small	50	2s	5123	**2s**	**2665**	16s	126
	75	9s	15623				
	100	25s	34844				

process models in less than a minute when we use less than 75 places. For the *BPIC2012* event log, [2] needs more than 10 minutes to return results and [24] investigates 454 candidate local process models in 90 seconds. This is less than what we can discover and it needs four times more time than our approach. The *Sepsis* and *RTFM* event logs further confirm these results, which shows that our algorithm is able to handle large event logs much better, while returning a large amount of local process models.

6 Conclusion and Outlook

In this paper, we introduced a novel way of discovering local process models. We proposed a first solution to the problem, which can be further investigated and extended. Our first goal was to have an algorithm that can find local process models for large event logs, and we achieved this by building local process models through one pass of the event log. Different quality dimensions that we discussed are returning minimal number of local process models that cover the entire event log or mining using utility functions. These are compelling directions that we plan to investigate as future work. Another point is that we get the place nets from which we build local process models from an oracle which currently is a regular process discovery algorithm. Hence, how to generate place nets valuable for local process model discovery or build the local process models without using place nets is something that warrants further research. The algorithm we propose is able to process large event logs and is flexible to support improvements for the above mentioned topics without destroying the linear complexity on the size of the event log.

Acknowledgments: We thank the Alexander von Humboldt (AvH) Stiftung for supporting our research.

References

1. van der Aalst, W.M.P.: Process Mining - Data Science in Action, Second Edition. Springer (2016). https://doi.org/10.1007/978-3-662-49851-4
2. Acheli, M., Grigori, D., Weidlich, M.: Efficient discovery of compact maximal behavioral patterns from event logs. In: Giorgini, P., Weber, B. (eds.) Advanced Information Systems Engineering - 31st International Conference, CAiSE 2019, Rome, Italy, June 3-7, 2019, Proceedings. Lecture Notes in Computer Science, vol. 11483, pp. 579–594. Springer (2019). https://doi.org/10.1007/978-3-030-21290-2_36
3. Acheli, M., Grigori, D., Weidlich, M.: Discovering and analyzing contextual behavioral patterns from event logs. IEEE Transactions on Knowledge and Data Engineering (2021). https://doi.org/10.1109/TKDE.2021.3077653
4. Bergenthum, R., Desel, J., Lorenz, R., Mauser, S.: Process mining based on regions of languages. In: Alonso, G., Dadam, P., Rosemann, M. (eds.) Business Process Management, 5th International Conference, BPM 2007, Brisbane, Australia, September 24-28, 2007, Proceedings. Lecture Notes in Computer Science, vol. 4714, pp. 375–383. Springer (2007). https://doi.org/10.1007/978-3-540-75183-0_27
5. Brunings, M., Fahland, D., van Dongen, B.F.: Defining meaningful local process models. In: van der Aalst, W.M.P., Bergenthum, R., Carmona, J. (eds.) Proceedings of the International Workshop on Algorithms & Theories for the Analysis of Event Data 2020 Satellite event of the 41st International Conference on Application and Theory of Petri Nets and Concurrency Petri Nets 2020, virtual workshop, June 24, 2020. CEUR Workshop Proceedings, vol. 2625, pp. 6–19. CEUR-WS.org (2020)
6. Carmona, J., Cortadella, J., Kishinevsky, M.: A region-based algorithm for discovering petri nets from event logs. In: Dumas, M., Reichert, M., Shan, M. (eds.) Business Process Management, 6th International Conference, BPM 2008, Milan, Italy, September 2-4, 2008. Proceedings. Lecture Notes in Computer Science, vol. 5240, pp. 358–373. Springer (2008). https://doi.org/10.1007/978-3-540-85758-7_26
7. Deeva, G., Weerdt, J.D.: Understanding automated feedback in learning processes by mining local patterns. In: Daniel, F., Sheng, Q.Z., Motahari, H. (eds.) Business Process Management Workshops - BPM 2018 International Workshops, Sydney, NSW, Australia, September 9-14, 2018, Revised Papers. Lecture Notes in Business Information Processing, vol. 342, pp. 56–68. Springer (2018). https://doi.org/10.1007/978-3-030-11641-5_5
8. Delcoucq, L., Lecron, F., Fortemps, P., van der Aalst, W.M.P.: Resource-centric process mining: clustering using local process models. In: Hung, C., Cerný, T., Shin, D., Bechini, A. (eds.) SAC '20: The 35th ACM/SIGAPP Symposium on Applied Computing, online event, [Brno, Czech Republic], March 30 - April 3, 2020. pp. 45–52. ACM (2020). https://doi.org/10.1145/3341105.3373864
9. van Dongen, B.F.: BPI Challenge 2012 (2012)
10. van Dongen, B.F.: BPI Challenge 2019 (2019)
11. Kirchner, K., Markovic, P.: Unveiling hidden patterns in flexible medical treatment processes - A process mining case study. In: Dargam, F.C.C., Delias, P., Linden, I., Mareschal, B. (eds.) Decision Support Systems VIII: Sustainable Data-Driven and Evidence-Based Decision Support - 4th International Conference, ICDSST 2018, Heraklion, Greece, May 22-25, 2018, Proceedings. Lecture Notes in Business Information Processing, vol. 313, pp. 169–180. Springer (2018). https://doi.org/10.1007/978-3-319-90315-6_14

12. Leemans, S.J.J., Tax, N., ter Hofstede, A.H.M.: Indulpet miner: Combining discovery algorithms. In: Panetto, H., Debruyne, C., Proper, H.A., Ardagna, C.A., Roman, D., Meersman, R. (eds.) On the Move to Meaningful Internet Systems. OTM 2018 Conferences - Confederated International Conferences: CoopIS, C&TC, and ODBASE 2018, Valletta, Malta, October 22-26, 2018, Proceedings, Part I. Lecture Notes in Computer Science, vol. 11229, pp. 97–115. Springer (2018). https://doi.org/10.1007/978-3-030-02610-3_6
13. Leemans, S., Fahland, D., Aalst, W.: Discovering Block-structured Process Models from Event Logs: A Constructive Approach. In: Colom, J.M., Desel, J. (eds.) Applications and Theory of Petri Nets 2013. lncs, vol. 7927, pp. 311–329. springer (2013). https://doi.org/10.1007/978-3-642-38697-8_17
14. de Leoni, M.M., Mannhardt, F.: Road Traffic Fine Management Process (2015)
15. Mannel, L.L., van der Aalst, W.M.P.: Finding complex process-structures by exploiting the token-game. In: Donatelli, S., Haar, S. (eds.) Application and Theory of Petri Nets and Concurrency - 40th International Conference, PETRI NETS 2019, Aachen, Germany, June 23-28, 2019, Proceedings. Lecture Notes in Computer Science, vol. 11522, pp. 258–278. Springer (2019). https://doi.org/10.1007/978-3-030-21571-2_15
16. Mannhardt, F.: Sepsis cases - event log (Dec 2016). https://doi.org/10.4121/uuid:915d2bfb-7e84-49ad-a286-dc35f063a460
17. Mannhardt, F., de Leoni, M., Reijers, H.A., van der Aalst, W.M.P., Toussaint, P.J.: Guided process discovery - A pattern-based approach. Inf. Syst. **76**, 1–18 (2018). https://doi.org/10.1016/j.is.2018.01.009
18. Mannhardt, F., Tax, N.: Unsupervised event abstraction using pattern abstraction and local process models. In: Gulden, J., Nurcan, S., Reinhartz-Berger, I., Guédria, W., Bera, P., Guerreiro, S., Fellmann, M., Weidlich, M. (eds.) Joint Proceedings of the Radar tracks at the 18th International Working Conference on Business Process Modeling, Development and Support (BPMDS), and the 22nd International Working Conference on Evaluation and Modeling Methods for Systems Analysis and Development (EMMSAD), and the 8th International Workshop on Enterprise Modeling and Information Systems Architectures (EMISA) co-located with the 29th International Conference on Advanced Information Systems Engineering 2017 (CAiSE 2017), Essen, Germany, June 12-13, 2017. CEUR Workshop Proceedings, vol. 1859, pp. 55–63. CEUR-WS.org (2017), http://ceur-ws.org/Vol-1859/bpmds-06-paper.pdf
19. Mannila, H., Toivonen, H., Verkamo, A.: Discovery of Frequent Episodes in Event Sequences. Data Mining and Knowledge Discovery **1**(3), 259–289 (1997)
20. Pijnenborg, P., Verhoeven, R., Firat, M., Laarhoven, H.v., Genga, L.: Towards evidence-based analysis of palliative treatments for stomach and esophageal cancer patients: a process mining approach. In: 2021 3rd International Conference on Process Mining (ICPM). pp. 136–143 (2021). https://doi.org/10.1109/ICPM53251.2021.9576880
21. Srikant, R., Agrawal, R.: Mining sequential patterns: Generalizations and performance improvements. In: Apers, P.M.G., Bouzeghoub, M., Gardarin, G. (eds.) Advances in Database Technology - EDBT'96, 5th International Conference on Extending Database Technology, Avignon, France, March 25-29, 1996, Proceedings. Lecture Notes in Computer Science, vol. 1057, pp. 3–17. Springer (1996). https://doi.org/10.1007/BFb0014140

22. Tax, N., Dalmas, B., Sidorova, N., van der Aalst, W.M.P., Norre, S.: Interest-driven discovery of local process models. Inf. Syst. **77**, 105–117 (2018). https://doi.org/10.1016/j.is.2018.04.006
23. Tax, N., Sidorova, N., van der Aalst, W.M.P., Haakma, R.: Heuristic approaches for generating local process models through log projections. In: 2016 IEEE Symposium Series on Computational Intelligence, SSCI 2016, Athens, Greece, December 6-9, 2016. pp. 1–8. IEEE (2016). https://doi.org/10.1109/SSCI.2016.7849948
24. Tax, N., Sidorova, N., Haakma, R., van der Aalst, W.M.P.: Mining local process models. J. Innov. Digit. Ecosyst. **3**(2), 183–196 (2016). https://doi.org/10.1016/j.jides.2016.11.001
25. Verbeek, E., Buijs, J.C.A.M., van Dongen, B.F., van der Aalst, W.M.P.: Prom 6: The process mining toolkit. In: Rosa, M.L. (ed.) Proceedings of the Business Process Management 2010 Demonstration Track, Hoboken, NJ, USA, September 14-16, 2010. CEUR Workshop Proceedings, vol. 615. CEUR-WS.org (2010), http://ceur-ws.org/Vol-615/paper13.pdf
26. Wen, L., van der Aalst, W.M.P., Wang, J., Sun, J.: Mining process models with non-free-choice constructs. Data Min. Knowl. Discov. **15**(2), 145–180 (2007). https://doi.org/10.1007/s10618-007-0065-y
27. van Zelst, S.J., van Dongen, B.F., van der Aalst, W.M.P., Verbeek, H.M.W.: Discovering workflow nets using integer linear programming. Computing **100**(5), 529–556 (2018). https://doi.org/10.1007/s00607-017-0582-5