Indulpet Miner: Combining Discovery Algorithms

Sander J. J. Leemans[1]✉, Niek Tax[2], and Arthur H. M. ter Hofstede[1]

[1] Queensland University of Technology, Brisbane, Australia
[2] Eindhoven University of Technology, Eindhoven, the Netherlands
s.leemans@qut.edu.au, n.tax@tue.nl, a.terhofstede@qut.edu.au

Abstract. In this work, we explore an approach to process discovery that is based on combining several existing process discovery algorithms. We focus on algorithms that generate process models in the process tree notation, which are sound by design. The main components of our proposed process discovery approach are the Inductive Miner, the Evolutionary Tree Miner, the Local Process Model Miner and a new bottom-up recursive technique. We conjecture that the combination of these process discovery algorithms can mitigate some of the weaknesses of the individual algorithms. In cases where the Inductive Miner results in overgeneralizing process models, the Evolutionary Tree Miner can often mine much more precise models. At the other hand, while the Evolutionary Tree Miner is computationally expensive, running it only on parts of the log that the Inductive Miner is not able to represent with a precise model fragment can considerably limit the search space size of the Evolutionary Tree Miner. Local Process Models and bottom-up recursion aid the Evolutionary Tree Miner further by instantiating it with frequent process model fragments. We evaluate our approaches on a collection of real-life event logs and find that it does combine the advantages of the miners and in some cases surpasses other discovery techniques.

Keywords: Process mining · process discovery · boosting · process trees · bottom-up recursion.

1 Introduction

Process Mining [1] is a scientific discipline that bridges the gap between process analytics and data analysis, and focuses on the analysis of event data logged during the execution of a business process. Events contain information on what was done, by whom, for whom, where, when, etc. Such event data is often readily available from information systems such as Enterprise Resource Planning (ERP), Customer Relationship Management (CRM) or Business Process Management (BPM) systems. *Process discovery*, which plays a prominent role in process mining, is the task of automatically generating a process model that accurately describes a business process based on such event data. Many process discovery techniques have been developed over the last decade (e.g. [3,5,6,7,12,13,16,17,32]), producing process models in various forms, such as Petri nets [26], process trees [6] and Business Process Model and Notation (BPMN) models [27].

In the research field of Machine Learning, it has long been studied how to combine multiple predictive models into a single combined model. This so-called *ensemble learning* gained traction when Schapire [28] showed that a strong classifier could be generated by combining a collection of weak classifiers through a procedure he called *boosting*. In later years, many different approaches have been developed to combine several predictors into a single more accurate predictor, including *bagging* [4], *stacking* [31] and *Bayesian model averaging* [14].

Dahari et al. [7] recently explored combining multiple process discovery approaches to obtain a single process model. The model that they obtain is based on multiple process models that originate from the Inductive Miner infrequent (IMi) [17] with different parameter settings of the algorithm. In this work, we take this idea one step further by exploring the combination of multiple process discovery algorithms to jointly discover a single process model, thereby bringing some of the ideas of ensemble learning to the field of process mining.

We focus on process discovery algorithms that generate process models in one consistent process representation, namely the *process tree* [6], in order to enable combining the results of discovery approaches. Three existing process discovery algorithms that generate process models in process tree notation are the Inductive Miner (IM) [16,17], the Evolutionary Tree Miner (ETM) [6] and the Local Process Model (LPM) Miner [29]. The Inductive Miner algorithm is a computationally very fast algorithm, however, the process models that it generates often allow for too much behaviour (i.e., they are imprecise) when it is applied to event logs that originate from highly variable or unstructured processes. The ETM uses a genetic algorithm to find a process tree that optimises multiple quality criteria for process models and it can, therefore, find more precise models from logs of unstructured processes. However, finding a high-quality process model with the ETM can be time-consuming when the process or the event log is large or complex. Our combination of process discovery algorithms, Indulpet Miner (IN), aims to combine the strengths of four process discovery techniques: Inductive Miner, Local Process Models, the Evolutionary Tree Miner and a new bottom-up recursive technique (BUR). First, we apply IM on the parts that it can describe precisely. Second, we use the LPM Miner and BUR to mine local patterns of process behaviour that we use as a starting point for the ETM, which prevents that the genetic search of the ETM has to start from scratch. Third, we only apply the ETM locally for the remaining parts.

The remainder of this paper is structured as follows: in Section 2 we discuss related research. In Section 3 we introduce basic concepts and notation that we use throughout the later sections of the paper. In Section 4 we introduce the Indulpet Miner, our novel mining approach. Section 5 evaluates the Indulpet Miner and compares it to existing techniques. Finally, Section 6 concludes the work.

2 Related Work

Several process discovery techniques have been proposed before. We briefly discuss the ones most relevant for this paper; for a more elaborate overview, please refer to [1]. Evoluationary Tree Miner, Inductive Miner and the Local Process Models technique will be described in Section 4.

The Split Miner [3] is a process discovery algorithm that extracts a set of directly follows relations from the event log and, from these relations, mines a process model using several heuristics. The Split Miner is often able to discover more precise process models than the Inductive Miner. Split Miner guarantees that the discovered process models are free of deadlocks, however, unlike soundness-guaranteeing algorithms, Split Miner does not guarantee that the final state of the model can be reached (no *weak soundness*).

Dahari et al. [7] recently developed the Fusion Miner, which is related to the Indulpet Miner in the sense that it mines several process trees and combines them into a single process model. However, the process trees that are combined by the Fusion Miner are restricted to those that are generated by the Inductive Miner infrequent (IMf) [17], while the Indulpet Miner combines process trees that originate from multiple algorithms, thereby making use of the strengths of different algorithms.

Mannhardt et al. [24] proposed a combined approach based on Local Process Models (LPMs) and the Inductive Miner (IM), thereby closely linking to the Indulpet Miner. This approach first uses LPMs to abstract the event log to a different event log where the events are on a higher level of granularity, then applies the IM to this higher level log, and replaces the high-level activities in the discovered model with the LPM patterns to obtain a model on the granularity level of the original log. In this work, we propose to start with IM, thereby reducing the application of the more computationally expensive LPM miner to fragments of the log where the IM fails to find a satisfactory result, while in the solution proposed by Mannhardt et al. [24] the LPM miner always needs to process the full log. Furthermore, this work incorporates the Evolutionary Tree Miner (ETM) and a novel bottom-up recursion (BUR) strategy.

3 Preliminaries

Given an alphabet of activities Σ containing all the basic process steps, Σ^* denotes the set of all sequences over Σ and $\sigma = \langle a_1, a_2, \ldots, a_n \rangle$ denotes a sequence of length n, with $|\sigma|{=}n$. $\langle\rangle$ denotes the empty sequence and $\sigma_1{\cdot}\sigma_2$ is the concatenation of sequences σ_1 and σ_2. A multiset (or bag) over X is a function $B : X \to \mathbb{N}$ which we write as $[a_1^{w_1}, a_2^{w_2}, \ldots, a_n^{w_n}]$, where for $1 \leq i \leq n$ we have $a_i \in \Sigma$ and $w_i \in \mathbb{N}^+$. The set of all multisets over X is denoted with $\mathcal{B}(X)$.

An *event log* is a multiset of traces that denote process executions. For instance, the log $[\langle a, b, c\rangle, \langle b, d\rangle^2]$ consists of one trace that consists of activity a followed by b and c, plus two traces of b followed by d.

A *directly follows graph* is an abstraction of an event log or a language. The nodes are the activities of the log or language, while the directed edges denote whether in the log or language, an activity may be directly followed by another activity. For instance, the directly follows graph of our example log is

$$\begin{tikzcd} a \arrow[r] & b \arrow[r] \arrow[rr, bend left] & c & d \end{tikzcd} .$$

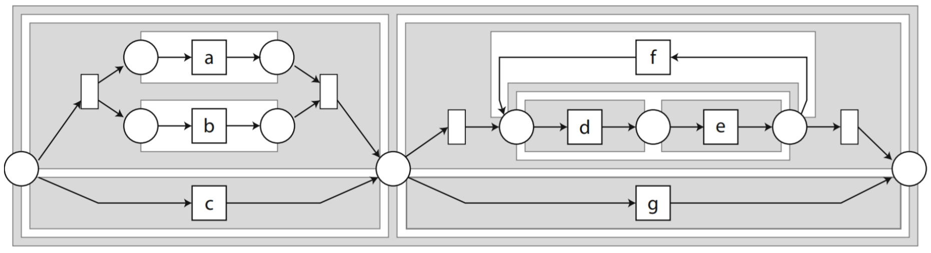


Fig. 1: An example of a labeled Petri net. This labeled Petri net has a block structure which is denoted by the filled regions. (obtained from [18]).

A frequently used process-model notation is the Petri-nets notation [26]. A Petri net is a directed bipartite graph consisting of places (depicted as circles) and transitions (depicted as rectangles), connected by arcs. Transitions represent activities, while places represent the enabling conditions of transitions. A special label τ is used to represent invisible transitions (depicted as narrow rectangles), which are only used for routing purposes and not recorded in the execution log.

In a *labelled Petri net*, labels are assigned to transitions to indicate the type of activity that they model.

A state of a Petri net is defined by its *marking*, and it is often useful to consider a Petri net in combination with an initial marking and a set of possible final markings. This allows us to define the language accepted by the Petri net as a set of sequences of activities ($\mathfrak{L}$). We refer to a Petri net with an initial and a set of final states as an *accepting* Petri net.

A *process tree* [6] is an abstract representation of a block-structured hierarchical process model, in which the leaves represent the activities and the operators describe how their children are to be combined. τ denotes the activity (leaf) whose execution is not visible in the event log. We consider four operators: $\times$, $\rightarrow$, $\wedge$ and $\circlearrowleft$ ($\oplus$ denotes any process tree operator). $\times$ describes the exclusive choice between its children, $\rightarrow$ the sequential composition and $\wedge$ the parallel composition. The first child of a loop $\circlearrowleft$ is the body of the loop; all other children are redo children. First, the body must be executed, followed by zero or more iterations of a redo child and the body; after each iteration, execution can stop. Fig. 1 shows the Petri net corresponding to the process tree $\rightarrow(\times(\wedge(a,b),c),\times(\circlearrowleft(\rightarrow(d,e),f),g))$. Process trees can be straightforwardly translated to Petri nets, and these translated nets are inherently sound.

Notation-wise, let t be a trace and let A be a set of activities, then $t|_A$ refers to a *projected* trace containing only the events of t of which the activities are in A. Notice that this projected trace may be empty. Similarly, for a log L, $L|_A$ denotes a log consisting of the traces of L projected on A.

In an *alignment* procedure, an event log and a process model, which can be a normative or a descriptive, are compared. That is, for each trace of the log, a matching execution path through the model is searched for. This matching path might deviate from both the log, by skipping events, and the model, by skipping activities. A matching with the lowest number of skips is referred to as an *optimal* alignment. For instance, an optimal alignment of the trace $\langle a, b, c \rangle$ and the process tree $\rightarrow(\times(\wedge(a, b), c), \times(\circlearrowleft(\rightarrow(d, e), f), g))$ is: $\frac{|\text{ trace}|\text{a b c -}|}{|\text{model}|\text{a b - g}|}$

4 Indulpet Miner

Indulpet Miner (IN) aims to combine the strengths of four process discovery techniques: Inductive Miner (IM), Local Process Models (LPM), the Evolutionary Tree Miner (ETM) and a new bottom-up recursive technique (BUR). We illustrate IN using Figure 2: starting with a full event log (2a), as a first step, Inductive Miner is applied, which tries to discover some structure in the event log and splits it accordingly into sub-logs, until it is unable to find structure in the sub-logs (2b). We start with the Inductive Miner since it is the only process tree algorithm that guarantees fitness, and hence, in situations where this miner manages to find a precise model, no further work is needed. This step increases the number of event logs but decreases their complexity.

Second, the new bottom-up recursive technique (BUR) is applied (2c). Where IM aims to find the highest-level structure in the log (starting with the root of the process tree), BUR aims to find lowest-level structure in the log (starting with individual activities and combining these into subtrees). The combination of IM and BUR is applied exhaustively until the event logs cannot be reduced in complexity anymore.

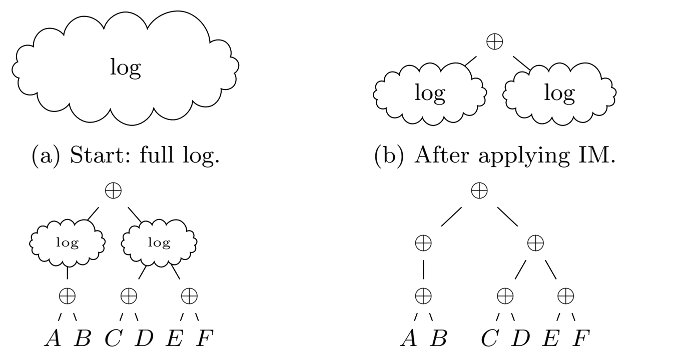
Third, LPM is applied to gather candidate local process models, which serve as starting seeds for ETM. That is, rather than starting from an arbitrary population of models, ETM begins its iterations using the local process models as its population. Initializing the ETM with an initial population of LPMs reduces the search space of the ETM, thereby improving computational efficiency. Finally, the ETM is applied to obtain a complete process tree (2d).

Using these steps, the low-hanging fruit of well-structured behaviour is captured by Inductive Miner and the bottom-up recursive method, thereby minimizing the heavy lifting that has to be performed by the Evolutionary Tree Miner.

In this section, we describe each of the steps of Indulpet Miner. We first describe the three existing techniques Inductive Miner, Evolutionary Tree Miner and Local Process Models in more detail. Second, we introduce the new bottom-up recursive technique. We finish with the pseudo code of IN and a description of its implementation.

4.1 Inductive Miner

Inductive Miner (IM) applies a recursive divide-and-conquer approach to process discovery, using four distinct steps [16]. That is, first the "most important" behaviour in the event log is identified, consisting of a process tree operator ($\times$, $\rightarrow$, $\wedge$, $\circlearrowleft$) and a proper division of the activities in the event log (a *cut*, denoted as $(\oplus, S_1, \ldots S_n)$ for operator $\oplus$ and sets of activities $S_1 \ldots S_n$). The operator is



(a) Start: full log. (b) After applying IM.

(c) After applying BUR. (d) After applying LPM and ETM.

Fig. 2: An illustration of Indulpet Miner applying several steps to obtain a process tree from an event log.

recorded as the root of the resulting process tree. Second, the cut found is used to split the event log into smaller sub-logs. Third, IM recurses on each sub-log until a *base case* is encountered, for instance, a log containing only a single activity, which is returned as a process tree leaf. Finally, if IM cannot find a cut, a *fall-through* is returned that overestimates the behaviour of the event log to be able to continue the recursion. For instance, if a particular activity occurs precisely once in each trace, then this activity is filtered out of the event log, the recursion continues, and the activity is put in parallel with the resulting process tree. As a last resort, IM will return a flower model that represents all behaviour over the activities in the event log, thereby guaranteeing fitness. In practice, on event logs of loosely structured processes, this last-resort fall-through of IM may cause a decrease in precision. With the Indulpet Miner, we aim to improve the fall-through of the Inductive Miner, by adding a more involved approach that combines several other process discovery techniques to the original fall-through strategies.

Due to its flexibility, several variants of IM have been proposed, in particular to handle noise and infrequent behaviour (IMf) [17] and to handle lifecycle information (IMlc) [19], and even their combination (IMflc) [19], which we use in Indulpet Miner[3].

Indulpet Miner uses all four steps of IMflc, and the cut selection, recursion and base case steps are used without change. If no cut can be found, before trying the default fall-throughs of IM and possibly overgeneralising, Indulpet first attempts to apply bottom-up recursion and, if that is successful, recurse further using IM. Second, if bottom-up recursion is not successful, Indulpet will apply LPM mining and finally the ETM. Should these not return a result, for

[3] Lifecycle information handling capabilities are necessary for Indulpet as the bottom-up recursion might insert such information in the event log, even if it was not present in the input event log.

instance when running out of a user-chosen time limit, the original fall-throughs of IM are applied to ensure that a process tree is returned at all times.

4.2 Evolutionary Tree Miner

The Evolutionary Tree Miner (ETM) [6] is a technique that applies a genetic search approach to process discovery: it starts with a population of process models and repeatedly evaluates, selects and mutates the models in the population to optimise it towards a set of chosen quality criteria. In each iteration, the models in the population are evaluated, and only the best-performing models are selected and considered further. The mutation steps of ETM are mutation and crossover. Crossover combines the well-performing parts of multiple models, while mutation replaces the ill-performing parts with random variations. To guarantee that all models can, eventually, be considered, a certain degree of randomness is inserted into the selection and mutation steps.

As ETM limits itself to process tree models, soundness is guaranteed. Furthermore, the Evolutionary Tree Miner can optimise for any log- or model-based quality criterion imaginable, and any process-tree construct, including duplicate activities, can be discovered. However, ETM is computationally expensive, which can be addressed by providing an initial population of models to give ETM a head start. In Indulpet Miner, we use the results of LPM mining for this purpose. Furthermore, to decrease the amount of work to be performed by ETM, Indulpet Miner reduces the size and complexity of the event log by a bottom-up recursion technique before calling ETM.

4.3 Local Process Models

Local Process Models (LPMs) [29] are process models that describe frequent but partial behaviour. That is, they model only a subset of the activities that were seen in the event log. An iterative expansion procedure is used to generate a ranked collection of LPMs. The iterative expansion procedure of LPM is often bounded to a maximum number of expansion steps (in practice often to 4 steps), as the expansion procedure is a combinatorial problem of which the size depends on the number of activities in the event log as well as the maximum number of activities in the LPMs that are mined.

LPM keeps a set of process trees, LPM, starting by considering a single activity from Σ, for instance $LPM_1 = a$. In each expansion step, the process trees in the set that occur often enough in the log, that is, their *support* exceeds some threshold, are expanded into larger trees. That is, an arbitrary activity is replaced with a sub-process tree containing that activity. For instance, one of the possible expansions of a is $\rightarrow(a,b)$, thereby creating $LPM_2 = \rightarrow(a,b)$ as an expansion of LPM_1. This procedure is repeated for every possible expansion using the operators $\times$, $\rightarrow$, $\wedge$ and $\circlearrowleft$. For instance, $\rightarrow(a,b)$ could be expanded into $LPM_3 = \rightarrow(a, \wedge(b,c))$, which is an expansion of LPM_2. The expansion procedure is guided by several heuristics and monotonicity properties [29], and ends at a certain user-chosen number of activities (for instance, 4).

All trees in the set that meet the given support threshold are returned. In Indulpet Miner, these returned trees serve as inputs to the ETM step.

4.4 Bottom-up Recursion

Inductive Miner recurses in a top-down fashion, that is, it looks for the 'largest' behaviour in an event log and uses this behaviour to split the log into multiple smaller sublogs. In this section, we propose a novel approach, bottom-up recursion (BUR), that looks for the 'smallest' behaviour in an event log and uses this behaviour to reduce the complexity of the event log. BUR applies four steps: first a *partial cut* $(\oplus, A, B)$ is identified, consisting of a process tree operator $\oplus$ and two sets of activities A, B[4]. Intuitively, a partial cut $(\oplus, A, B)$ denotes that $\oplus(M_A, M_B)$ has been identified as being a subtree of the resulting process tree, where M_A and M_B are process trees representing the choices between the activities of A and B respectively: $M_A = \times(a_1, \dots a_n), M_B = \times(b_1, \dots b_m)$. Second, this partial cut is used to *collapse* the event log: the activities of the partial cut are replaced by a dummy activity in each trace of the log. Third, BUR recurses and a process tree is returned. Fourth, in the resulting process tree, the dummy activity is replaced with a subtree corresponding to the partial cut.

For instance, let $L = [\langle a, b, d\rangle, \langle a, d, b\rangle]$ be an event log. BUR could identify the partial cut $(\rightarrow, \{a\}, \{b\})$. Next, L is collapsed by replacing a and b with a dummy activity y: $L' = [\langle y_{\text{S}}, y_{\text{C}}, d\rangle, \langle y_{\text{S}}, d, y_{\text{C}}\rangle]$. Then, L' is recursed on and, as y and d overlap in time, a process tree $\wedge(y, d)$ results. As a final step, BUR replaces y with a subtree representing the partial cut, and the final model becomes $\wedge(\rightarrow(a, b), d)$.

In the remainder of this section, we first define partial cuts formally and show how they can be identified. Second, we explain the log collapsing.

Partial Cuts and Detecting them. A *partial cut* $(\oplus, A, B)$ consists of an operator $\oplus$ and two sets of activities A and B:

Definition 1 (Partial cut). *Let Σ be an alphabet, $\oplus \neq \times$ be a process tree operator and A, B be sets of activities such that $A \cup B \subseteq \Sigma$. Then, $(\oplus, A, B)$ is a partial cut of Σ.*

Notice that every binary cut of Inductive Miner is also a partial cut. Intuitively, a partial cut $(\oplus, \{A_1, \dots A_m\}, \{B_1, \dots B_n\})$ means that $\oplus(\times(A_1, \dots A_m), \times(B_1, \dots B_n))$is a subtree of the to-be discovered model. That is, $\oplus$ defines the relation between the set of activities A and B, while the relation between the activities within sets A and B is defined by $\times$. Therefore, partial cuts do not need to consider $\times$-operators: we limit ourselves to $\rightarrow$, $\wedge$ and $\circlearrowleft$.

We formalise this intuition in *fitting* partial cuts: a partial cut is fitting if and only if, in each trace, the tree corresponding to the partial cut is executed zero or more times, that is, it is never violated:

[4] For simplicity, we use only binary partial cuts in this paper. The definitions extend to n-ary partial cuts and this does not change the expressivity of the method [18].

Definition 2 (Fitting partial cut). *Let Σ be an alphabet, let L be an event log over Σ and let $C = (\oplus, A, B) = (\oplus, \{A_1, \dots A_m\}, \{B_1, \dots B_n\})$ be a partial cut. Then, C is a* fitting partial cut *if each trace in the log follows the semantics of the partial cut:*

$$L|_{A\cup B} \subseteq \mathfrak{L}\left(\begin{tikzcd} & & \circlearrowleft \arrow[dll, dash] \arrow[drr, dash] & & & & \\ \tau & & & & \oplus \arrow[dll, dash] \arrow[dr, dash] & & \\ & & \times \arrow[dl, dash] \arrow[d, dash] \arrow[dr, dash] & & & \times \arrow[dl, dash] \arrow[d, dash] \arrow[dr, dash] & \\ & A_1 & \dots & A_m & B_1 & \dots & B_n \end{tikzcd}\right)$$

Notice that this definition gives another reason not to consider $\times$-operators in partial cuts: a partial cut $(\times, \{a\}, \{b\})$ would always fit and thus not express any new information.

In Indulpet Miner, BUR exhaustively considers all partial cuts to find one that is fitting. As soon as a fitting partial cut is found, BUR continues as described before.

It is rather expensive to test whether a given partial cut is fitting, as this requires a pass over the entire event log. To limit the number of times that this time-consuming step has to be performed, we identified some necessary, though not sufficient, conditions that a partial cut has to satisfy in order to be fitting. Thus, BUR uses these conditions to prune the search space of partial cuts.

For these conditions, and for the remainder of this paper, we assume that the partial cut is disjoint and non-empty, i.e. $A \cap B = \emptyset$, $A \neq \emptyset$ and $B \neq \emptyset$.

Lemma 3 (Necessary conditions for fitting partial cuts). *Let Σ be an alphabet, L be an event log over Σ and $C = (\oplus, A, B)$ be a partial cut such that C fits L, $A \cap B = \emptyset$ and $A, B \neq \emptyset$. Then:*

1. *Within the sets of activities, there are no connections in the directly follows graph:*
   $\forall_{X \in \{A,B\}} \forall_{a,b \in X \wedge a \neq b} a \not\twoheadrightarrow b$
2. *Between the sets of activities, the directly follows graph exhibits an "approved" pattern, depending on the operator $\oplus$. Let $a \in A$ and $b \in B$:*
   - $\oplus = \rightarrow$ *There is a directly follows relation between a and b: $a \twoheadrightarrow b$.*
   - $\oplus = \wedge$ *A directly follows relation is present in both directions: $a \twoheadrightarrow b \wedge b \twoheadrightarrow a$.*
   - $\oplus = \circlearrowleft$ *A directly follows relation is present in both directions: $a \twoheadrightarrow b \wedge b \twoheadrightarrow a$.*
3. *For each pair of activities in A, B in relation to each other activity, the directly follows graph exhibits an "approved" pattern, depending on the operator $\oplus$. Let $a \in A$, $b \in B$ and $c \in \Sigma \setminus (A \cup B)$:*

$\oplus = \rightarrow$

$$\begin{tikzcd} a \arrow[rr] & & b \\ & c & \end{tikzcd} \quad \begin{tikzcd} a \arrow[rr] & & b \\ & c \arrow[ul] & \end{tikzcd} \quad \begin{tikzcd} a \arrow[rr] & & b \arrow[dl] \\ & c & \end{tikzcd} \quad \begin{tikzcd} a \arrow[rr] & & b \arrow[dl] \\ & c \arrow[ul] & \end{tikzcd} \quad \begin{tikzcd} a \arrow[rr] \arrow[dr, shift left] & & b \arrow[dl, shift left] \\ & c \arrow[ul, shift left] \arrow[ur, shift left] & \end{tikzcd}$$

$\oplus = \wedge$

$$\begin{tikzcd} a \arrow[rr, shift left] & & b \arrow[ll, shift left] \\ & c & \end{tikzcd} \quad \begin{tikzcd} a \arrow[rr, shift left] \arrow[dr] & & b \arrow[ll, shift left] \arrow[dl] \\ & c & \end{tikzcd} \quad \begin{tikzcd} a \arrow[rr, shift left] & & b \arrow[ll, shift left] \\ & c \arrow[ul] \arrow[ur] & \end{tikzcd} \quad \begin{tikzcd} a \arrow[rr, shift left] \arrow[dr, shift left] & & b \arrow[ll, shift left] \arrow[dl, shift left] \\ & c \arrow[ul, shift left] \arrow[ur, shift left] & \end{tikzcd}$$

$\oplus = \circlearrowleft$
$$\begin{tikzcd}[column sep=small] a \arrow[r, bend left] & b \arrow[l, bend left] \\ & c \end{tikzcd} \quad \begin{tikzcd}[column sep=small] a \arrow[r, bend left] \arrow[rd] & b \arrow[l, bend left] \\ & c \end{tikzcd} \quad \begin{tikzcd}[column sep=small] a \arrow[r, bend left] & b \arrow[l, bend left] \\ & c \arrow[lu] \end{tikzcd} \quad \begin{tikzcd}[column sep=small] a \arrow[r, bend left] \arrow[rd, bend left] & b \arrow[l, bend left] \\ & c \arrow[lu, bend left] \end{tikzcd} \quad \begin{tikzcd}[column sep=small] a \arrow[r, bend left] \arrow[rd, bend left] & b \arrow[l, bend left] \arrow[ld, bend left] \\ & c \arrow[lu, bend left] \arrow[ru, bend left] \end{tikzcd}$$

4. *Let x be an activity, then x is a* projected start activity *if* $\exists_{t \in L|_{A \cup B}} t = \langle x, \ldots \rangle$. *Then, for $\oplus = \wedge$, both a and b must be projected start activities. Then, for $\oplus = \circlearrowleft$, a must be both a projected start activity and b must not. (a symmetric requirement holds for projected end activities)*

The proof of this lemma follows from inspection of the semantics of partial cuts. We show that the conditions of the lemma are not sufficient to conclude that a partial cut is fitting by means of a counterexample: consider the event log $L_1 = [\langle a, b, c, b \rangle, \langle c, a, c \rangle]$ and the partial cut $C_1 = (\rightarrow, a, b)$. The directly follows graph of L_1 is $\begin{tikzcd} a \arrow[r] \arrow[rr, bend left] & c \arrow[l, bend left] \arrow[r] & b \arrow[l, bend left] \end{tikzcd}$. The partial cut C_1 satisfies all conditions of Lemma 3, but does not fit L_1, as, for instance, the second trace of L_1 projected on a and b yields $\langle a \rangle$, which violates the partial cut, as this cut indicates that after each a there should be a b.

To handle noise, the fitness requirement of Definition 2 can be relaxed. That is, BUR searches for the partial cut with the highest fitness, as long as the fitness (measured as the fraction of traces that adhere to the partial cut) reaches a certain user-chosen threshold.

Log Collapsing As the second step, given a partial cut, BUR collapses the event log, depending on the operator $\oplus$ of the cut.

In LPM [24], a similar step is performed ("log abstraction"). However, the LPM collapsing procedure is computationally much more expensive due to the need to use alignments in order to obtain fitness and precision measures, which BUR does not need in this step.

In BUR, any activity that is not part of the partial cut is ignored. Let y be a fresh activity that does not appear in the log. Then, every execution instance of the partial cut is replaced with an execution instance of y. That is, the first event of each instance is replaced with y_{start}, the last event with $y_{complete}$, and all other events of the activities in the partial cut in between are removed. In case the partial log is not fitting, non-fitting events are removed.

For instance for $(\rightarrow, \{a\}, \{b\})$: $\langle a, b, c, a, b, a, b \rangle$ is collapsed into $\langle y_{start}, y_{complete}, c, y_{start}, y_{complete}, y_{start}, y_{complete} \rangle$. As another example, for $(\circlearrowleft, \{a\}, \{b\})$: $\langle a, b, c, a, c, a, b, a, b, a \rangle$ is collapsed into $\langle y_{start}, c, y_{complete}, c, y_{start}, y_{complete} \rangle$.

Example of BUR. Let L_1 be $\{\langle a, b \rangle, \langle b, a \rangle, \langle a, b, c, d, a, b \rangle, \langle b, a, c, d, b, a \rangle\}$. Its directly follows graph is
$$\begin{tikzcd} a \arrow[r, bend left] \arrow[rd] & b \arrow[l, bend left] \arrow[d] \\ d \arrow[u] \arrow[ru] & c \arrow[l] \end{tikzcd}.$$

– The partial cut $(\wedge, \{a\}, \{c\})$ does not preserve fitness, with as a counterexample the trace $\langle a, b \rangle$, as this trace contains a but not c;

– The partial cut $(\circlearrowleft, \{a\}, \{c\})$ preserves fitness, but the directly follows graph does not contain the required edge $c \twoheadrightarrow a$;
– The partial cut $(\wedge, \{a\}, \{b\})$ preserves fitness and has an approved directly follows pattern;
– The partial cut $(\rightarrow, \{c\}, \{d\})$ preserves fitness and has an approved directly follows pattern;

Arbitrarily choose the partial cut $(\wedge, \{a\}, \{b\})$ and merge L_1 using this cut and the fresh activity e: $L_2 = \{\langle e_{\mathrm{S}}, e_{\mathrm{C}}\rangle, \langle e_{\mathrm{S}}, e_{\mathrm{C}}\rangle, \langle e_{\mathrm{S}}, e_{\mathrm{C}}, c, d, e_{\mathrm{S}}, e_{\mathrm{C}}\rangle, \langle e_{\mathrm{S}}, e_{\mathrm{C}}, c, d, e_{\mathrm{S}}, e_{\mathrm{C}}\rangle\}$.

On L_2, the algorithm recurses. Its directly follows graph is $\begin{tikzcd} e \arrow[rr, bend left] & d \arrow[l] & c \arrow[l] \end{tikzcd}$.

– The partial cut $(\wedge, \{e\}, \{c\})$ does not preserve fitness, with as a counterexample the trace $\langle e_{\mathrm{S}}, e_{\mathrm{C}}\rangle$, as e appears but c does not.
– The partial cut $(\circlearrowleft, \{e\}, \{c\})$ preserves fitness, but the directly follows graph does not contain the edge $c \twoheadrightarrow e$, so an approved pattern is not present;
– The partial cut $(\rightarrow, \{c\}, \{d\})$ preserves fitness and has an approved directly follows pattern;

Choose the partial cut $(\rightarrow, \{c\}, \{d\})$ and merge L_2 using this cut and the fresh activity f: $L_3 = \{\langle e_{\mathrm{S}}, e_{\mathrm{C}}\rangle, \langle e_{\mathrm{S}}, e_{\mathrm{C}}\rangle, \langle e_{\mathrm{S}}, e_{\mathrm{C}}, f_{\mathrm{S}}, f_{\mathrm{C}}, e_{\mathrm{S}}, e_{\mathrm{C}}\rangle, \langle e_{\mathrm{S}}, e_{\mathrm{C}}, f_{\mathrm{S}}, f_{\mathrm{C}}, e_{\mathrm{S}}, e_{\mathrm{C}}\rangle\}$. On L_3, BUR recurses and obtains the directly follows graph $\begin{tikzcd} e \arrow[r, bend left] & f \arrow[l, bend left] \end{tikzcd}$.

– The partial cut $(\wedge, \{e\}, \{f\})$ does not preserve fitness, with as a counterexample the trace $\langle e_{\mathrm{S}}, e_{\mathrm{C}}\rangle$;
– The partial cut $(\circlearrowleft, \{f\}, \{e\})$ does not preserve fitness, with as a counterexample the trace $\langle e_{\mathrm{S}}, e_{\mathrm{C}}\rangle$;
– The partial cut $(\circlearrowleft, \{e\}, \{f\})$ preserves fitness and has an approved directly follows pattern;

Thus, choose the partial cut $(\circlearrowleft, \{e\}, \{f\})$ and construct a process tree top-down by replacing the introduced fresh activities with process trees corresponding to their partial cuts: $\circlearrowleft(e, f)$ to $\circlearrowleft(\wedge(a, b), f)$ and as the final result $\circlearrowleft(\wedge(a, b), \rightarrow(c, d))$.

Complexity and Rediscoverability. All possible partial cuts are explored, and the first one that is encountered that satisfies Definition 2 is returned. The number of possible partial cuts is $O(2^{2^{|\Sigma|}})$, all of which need to be considered. The properties of Lemma 3 are applied for each partial cut to minimise the time spent per partial cut. If a property fails, the partial cut is discarded. If all properties of Lemma 3 hold, the fitness of the partial cut with respect to the event log is measured to find the best-fitting partial cut. In practice, this last step is rarely called.

Acknowledging these run-time considerations, BUR could be used as a standalone process discovery algorithm for smaller logs. Such an algorithm would be able to distinguish all process trees consisting of the four operators $\times$, $\rightarrow$,

$\wedge$ and $\circlearrowleft$ (but excluding τ leaves). In particular, so-called short loops can be handled, which have been shown to pose difficulties for discovery algorithms such as the IM and the α-algorithm [1]. For instance, $\wedge(\circlearrowleft(a,b),\circlearrowleft(c,d))$ can be distinguished from $\circlearrowleft(\wedge(a,c),\wedge(b,d))$, even though these two trees have the same directly follows graph.

4.5 Indulpet: Algorithm & Implementation

To summarise, Indulpet Miner applies the following steps:

```
function INDULPET(log $L$)
    if IMflc finds a base case $b$ in $L$ then return $b$ end if
    if IMflc finds a cut $c$ of operator $\oplus$ in $L$ then
        $L_1 \dots L_n \leftarrow$ split $L$ using $c$
        return $\oplus$(INDULPET($L_1$), ... INDULPET($L_n$))
    end if
    if BUR finds a partial cut $c = (\oplus, \{Q_1, \dots Q_q\}, \{R_1, \dots R_r\})$ in $L$ then
        $L' \leftarrow$ collapse $L$ using $c$ and a fresh activity $a'$
        $T' \leftarrow$ INDULPET($L'$)
        return $T$ with all $a'$'s replaced by $\oplus(\times(Q_1, \dots Q_q), \times(R_1, \dots R_r))$
    end if
    $X \leftarrow$ LPM($L$)
    if $M \leftarrow$ ETM($L$, $X$) then return $M$ end if
    return the first fallthrough of IMflc that applies
end function
```

Please note that Indulpet adheres to the IM framework, thus all guarantees and proofs provided for the IM framework apply [15]. The proof obligations for fitness do not hold due to the ETM step, thus fitness is not guaranteed. However, rediscoverability holds for Indulpet, that is, the ability to rediscover the language of a system-model underlying the event log, while making some assumptions on the class of the system-model and the event log [15, Theorem 6.43].

Indulpet Miner has been implemented as a plug-in of the ProM framework [11] and is distributed in the package manager of ProM 6.8. Given the high complexity of bottom-up recursion (there are $O(2^{2^n})$ possible partial cuts that all need to be considered), a time limit of 10 minutes is applied for this step. Furthermore, the ETM step is time-limited to 10 minutes as well. Both time limits can be overridden by a user. Please note that both steps can be called many times, thus Indulpet might take longer than the set time limit.

5 Evaluation

Several process discovery techniques have been proposed, and in this section, we compare Indulpet Miner to existing discovery techniques. In particular, we aim to answer two questions: 1) does Indulpet Miner strike a new balance in log-quality criteria compared to existing techniques, and 2) does Indulpet Miner combine the advantages of Inductive Miner, Local Process Models, bottom-up recursion and the Evolutionary Tree Miner?

To perform this evaluation, we applied the discovery algorithms to several real-life event logs. The real-life event logs that were included are described in Table 1. We perform our measures using 3-fold cross-validation, in which each log is split into three buckets. That is, each trace is put in one of the three buckets randomly. Each combination of two buckets is used for discovery, while one bucket is used for evaluation.

To address the randomness of some algorithms (Indulpet, Evolutionary Tree Miner), we repeat the 3-fold validation ten times. That is, for each real-life log, each algorithm is applied 30 times in total, yielding 30 models.

All models are translated to accepting Petri nets (to reduce the impact of this translation on the results, models were structurally reduced after the translation), and fitness is measured using alignments [2]. For most event logs, precision is measured using ETC [25] (as the procedure from process tree via accepting Petri net to alignment and ETC is not deterministic, it was applied 5 times for each model). For the bpic15 events logs, computing the alignments proved too time- and memory consuming and we evaluated fitness and precision using Projected Conformance Checking (PCC) [21]. As we are interested in the trade-offs between quality criteria, we do not report on the f-score. Furthermore, the number of places, transitions, and arcs in the accepting Petri nets is reported as simplicity and we verified the boundedness of models using the LoLa tool [30].

All existing discovery techniques that guarantee soundness and of which an implementation is publicly available were included: Evolutionary Tree Miner (ETM) [6] and Inductive Miner - infrequent (IMf) [17]. Furthermore, we added Split Miner (SM) [3], which guarantees models without deadlocks, though neither weak soundness nor boundedness. Finally, a baseline flower model (F) was included, which is a model that supports all behaviour over all activities seen in the event log. It would be interesting to include the Fusion Miner [7] and the approach of [24] as well, however both techniques lack an implementation that is compatible with the ProM framework, therefore disallowing the application of evaluation measures provided by ProM such as ETC, PCC, simplicity measures, and soundness checking.

Reproducibility. All event logs are publicly available, and the source code we used to run these experiments is available at `https://svn.win.tue.nl/repos/prom/Packages/SanderLeemans/Trunk/`.

5.1 Results & Discussion

The results of the evaluation are shown in Table 2. Some results could not be obtained and have been denoted with exclamation marks: ETM ran out of memory or ran for multiple days, while SM returned unbounded models that could not be measured by either alignment-based or PCC conformance checking measures.

Run time. Table 2 shows indicative run times, given as $\log_{10}$, such that 1 is one second, 2 is hundred seconds, etc. These results suggest that IN is faster than ETM, while IMf is faster than IN, both by several orders of magnitude.

		events	traces	activities
bpic12	BPI Challenge 2012 [8]: a mortgage application-to-approval process in a Dutch financial institute.	262,200	13,087	36
bpic15	BPI Challenge 2015 [9]: a building-permit approval process in five Dutch municipalities.	52,217	1199	395
		44,354	832	410
		59,681	1409	383
		47,293	1053	356
		59,083	1156	389
bpic18	BPI Challenge 2018 [10]: an agricultural grant-application process of the European Union. There are eight logs in this data set, each representing a sub-process for a particular document.	161,296	43,808	7
		46,669	29,297	6
		293,245	15,260	20
		569,209	29,059	16
		197,717	5,485	15
		132,963	14,750	10
		984,613	43,809	24
		128,554	43,802	6
rtf	Road Traffic Fines [22]: a process of collecting road traffic fines by an Italian government.	561,470	150,370	11
sps	Sepsis [23]: a sepsis-treating process in a hospital	15,214	1,050	16

Table 1: Event logs used in the evaluation.

	bpic12				bpic15-1				bpic15-2			
	f	p	s	t	f	p	s	t	f	p	s	t
IMf	0.97±0.01	0.59±0.03	186.57±9.59	0	1.00±0.00	0.66±0.04	1286.07±149.54	1	0.99±0.00	0.76±0.04	1048.10±167.84	1
IN	0.36±0.07	0.88±0.05	116.30±30.52	3	0.78±0.01	0.97±0.01	91.50±54.38	3	0.74±0.01	0.96±0.01	114.10±23.71	3
ETM	!	!	!	!	!	!	!	!	!	!	!	!
SM	0.96±0.00	0.68±0.00	239.00±0.00	1	!	!	4715.20±8.88	2	!	!	5077.27±10.28	
F	1.00±0.00	0.11±0.00	85.00±0.00	-1	1.00±0.00	0.64±0.01	1145.30±20.02	-1	1.00±0.00	0.64±0.01	1172.70±24.44	-1
	bpic15-3				bpic15-4				bpic15-5			
	f	p	s	t	f	p	s	t	f	p	s	t
IMf	1.00±0.00	0.71±0.03	1201.63±133.03	1	0.99±0.00	0.75±0.03	1082.47±98.26	0	0.99±0.00	0.79±0.05	1108.60±141.55	1
IN	0.79±0.01	0.97±0.01	118.37±80.59	3	0.76±0.02	0.93±0.03	272.17±63.98	4	0.75±0.02	0.96±0.02	244.17±120.50	3
ETM	!	!	!	!	!	!	!	!	!	!	!	!
SM	!	!	3995.47±8.96		!	!	3882.73±5.00	2	!	!	4730.20±11.38	2
F	1.00±0.00	0.66±0.01	1114.50±16.26	-1	1.00±0.00	0.65±0.02	1024.30±27.16	-1	1.00±0.00	0.65±0.02	1116.40±26.41	-1
	bpic18-1				bpic18-2				bpic18-3			
	f	p	s	t	f	p	s	t	f	p	s	t
IMf	1.00±0.00	0.95±0.02	54.87±0.73	-0	0.96±0.00	0.96±0.05	43.70±6.73	-1	0.93±0.00	0.53±0.01	74.77±1.28	1
IN	1.00±0.00	0.95±0.02	54.87±0.73	-0	0.96±0.00	0.96±0.05	43.70±6.73	-1	0.79±0.04	0.95±0.04	55.60±17.73	3
ETM	0.97±0.04	0.95±0.12	77.50±86.35	3	0.99±0.00	0.76±0.21	183.93±91.24	4	!	!	!	!
SM	1.00±0.00	0.97±0.03	59.00±0.00	1	1.00±0.00	0.95±0.02	90.00±0.00	1	1.00±0.00	0.67±0.01	291.00±0.00	1
F	1.00±0.00	0.32±0.00	30.00±0.00	-0	1.00±0.00	0.51±0.00	27.00±0.00	-1	1.00±0.00	0.14±0.00	68.90±0.55	-0
	bpic18-4				bpic18-5				bpic18-6			
	f	p	s	t	f	p	s	t	f	p	s	t
IMf	0.86±0.04	0.35±0.03	96.57±14.44	1	0.80±0.01	0.67±0.01	129.10±15.84	-0	0.97±0.00	0.55±0.02	73.83±2.74	-1
IN	0.61±0.19	0.83±0.27	71.97±50.49	4	0.78±0.01	0.69±0.05	164.00±43.89	2	0.97±0.00	0.55±0.02	73.83±2.74	-0
ETM	!	!	!	!	!	!	!	!	!	!	!	!
SM	0.99±0.00	0.55±0.00	247.00±0.00	1	0.88±0.00	0.74±0.00	131.00±0.00	1	1.00±0.00	0.72±0.01	147.00±0.00	1
F	1.00±0.00	0.18±0.00	57.00±0.00	-0	1.00±0.00	0.15±0.00	54.00±0.00	-1	1.00±0.00	0.34±0.01	39.00±0.00	-1
	bpic18-7				bpic18-8				rtf			
	f	p	s	t	f	p	s	t	f	p	s	t
IMf	0.93±0.02	0.83±0.01	164.67±17.54	2	1.00±0.00	0.89±0.06	54.87±3.17	-0	0.98±0.00	0.79±0.04	97.97±4.78	1
IN	0.89±0.03	0.84±0.02	220.43±70.34	3	1.00±0.00	0.89±0.05	54.87±3.17	-0	0.98±0.00	0.79±0.02	96.50±7.00	2
ETM	!	!	!	!	!	!	!	!	0.89±0.08	0.68±0.27	117.93±85.12	3
SM	0.02±0.00	0.96±0.00	333.00±0.00	2	1.00±0.00	0.97±0.03	66.00±0.00	1	1.00±0.00	0.96±0.00	82.00±0.00	1
F	1.00±0.00	0.64±0.01	81.00±0.00	-0	1.00±0.00	0.41±0.01	27.00±0.00	-0	1.00±0.00	0.38±0.01	46.00±0.00	-0
	sps											
	f	p	s	t								
IMf	0.91±0.02	0.41±0.06	140.23±13.85	-1								
IN	0.91±0.02	0.43±0.05	122.43±8.66	2								
ETM	!	!	!	!								
SM	0.76±0.00	0.73±0.01	138.00±0.00	0								
F	1.00±0.00	0.21±0.00	61.00±0.00	-2								

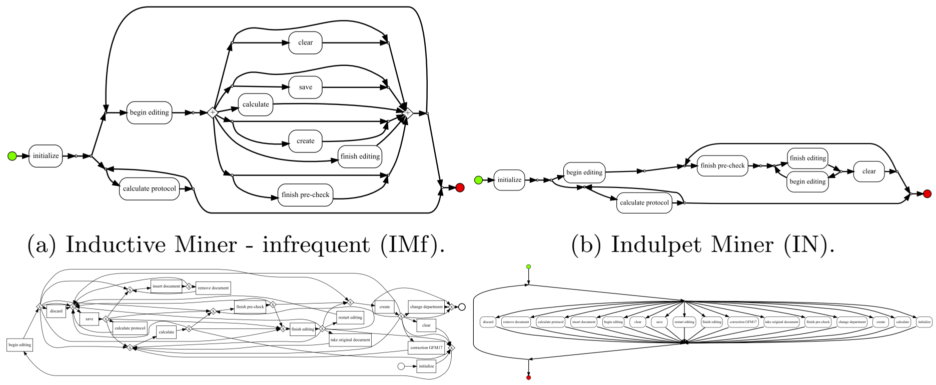
Table 2: Results of the evaluation. Fitness: f, precision: p, simplicity: s, time in $\log_{10}$ seconds: t. The numbers are given as average ± sample standard deviation.

Comparing IN with SM yields mixed results: on some logs (e.g. bpic12, bpic15_1), SM is faster, while on others (e.g. bpic18_1, bpic18_2) IN uses its top-down recursion more and is faster, although the logs on which IN is faster tend to be the smaller logs, where IN behaves as IMf. It is clear that IMf and SM are preferred choices if a process model is to be obtained fast. Nevertheless, to put things in perspective, the maximum run time of IN observed in this experiment was 5 hours, which suggests that IN is still feasible, even for large and complex logs.

Quality. For bpic12, IN achieves the simplest model (except baseline F) and the highest precision, but fitness is significantly lower than the other algorithms. On all bpic15 logs, IN consistently achieves the best simplicity (even surpassing baseline F) and precision, with a lower fitness. On these logs, SM discovers very complex unbounded models, which can be measured using neither alignments nor the PCC framework.

The bpic18 logs differ much in complexity (6 to 24 activities) and show mixed results: on bpic18_1 and bpic18_2, IN discovers the same models as IMf, with their measures differing marginally from SM, though always being Pareto optimal. On bpic18_3 and bpic18_4, all tested algorithms perform Pareto optimal. To illustrate these models, four have been included in Figure 3: in the model by IMf, some concurrent activities can be arbitrarily repeated, leading to a rather low precision. The model by SM does not contain any concurrency and has a high fitness. However, it is much less simple and precise than the other models. The model by IN has a lower fitness, but a much higher precision and is very simple. To provide some intuition: in the vast majority of traces in this log, the activities "begin editing" and "finish editing" are alternating according to intuition: for instance, only in 34 of the 29,059 traces, the trace ends after "begin editing" rather than "finish editing", and only 15,325 of the 295,621 executions of both activities are repeated. The model by IMf only requires that after each "finish editing" there must be a "begin editing", the model by SM requires "begin editing" to be executed first, but afterwards no further constraint is posed on these two activities, and the model by F does not express this constraint at all. Only IN discovers this constraint correctly by duplicating the "begin editing" activity.

On bpic18_5, SM is the clear winner on all dimensions. Remarkably, on this log IMf has a large standard deviation on simplicity, which, as IMf is deterministic, indicates that the 3-fold procedure withheld useful information from discovery. As the standard deviation of SM is much smaller, SM uses less information from the event logs. On bpic18_6, IMf and IN discover the same models, but SM achieves the highest fitness and precision, and F the best simplicity (all models are Pareto optimal). On bpic18_7, IMf achieves the highest fitness, while IN discovers models with slightly higher precision and lower fitness. The models by SM have the highest precision, but a very low fitness, as the log contains many activities that are repeated (so-called short loops), which SM does not discover. On bpic18_8, all algorithms achieve perfect fitness, IN and IMf achieve the best simplicity, and SM achieves the highest precision.



(a) Inductive Miner - infrequent (IMf). (b) Indulpet Miner (IN).

(c) Split Miner (SM). (d) Flower model (F).

Fig. 3: Results for bpic18-4 ("Geo parcel document").

On rtf, SM discovers a model that is better on all measures than IMf, IN and ETM. Finally, on sps, IN is Pareto-optimal over IMf, and SM achieves the highest precision, at the cost of fitness and simplicity.

The results show that IN in many cases combines the advantages of its sub-algorithms: the speed, feasibility and fitness of IMf with the precision of ETM. However, it is difficult to appoint a clear winner: algorithms strike different balances of log-quality criteria, and all routinely achieve Pareto optimality. Nevertheless, we can conclude that Indulpet Miner can achieve a different balance in log-quality criteria than the other tested techniques and might be a useful discovery algorithm in the toolbox of analysts, depending on the use case at hand.

6 Conclusion & Future Work

We have presented a novel process discovery approach which combines several existing process discovery algorithms that are based on process trees: the Inductive Miner (IM) [16], the Evolutionary Tree Miner (ETM) [6] and the Local Process Model (LPM) miner [29]. Additionally, we have proposed a novel process tree mining approach that is based on bottom-up recursion, which is too computationally complex to process a full event log but is useful as an element in the Indulpet miner to find local structures of process behaviour. The Indulpet miner is guaranteed to find sound process models, and we have shown on a collection of real-life event logs that the Indulpet miner often provides a Pareto optimal, yet different, trade-off between fitness, precision and simplicity compared to the Inductive Miner and the Split Miner algorithms.

An interesting direction of future work would be to explore heuristic search or other intelligent approaches to search the space of all possible cuts of the proposed bottom-up recursion technique. This would enable the use of bottom-up recursion as a process discovery technique by itself, rather than an element in a hybrid mining approach, which is currently rendered infeasible by the computational complexity of exploring the full space of bottom-up recursion cuts. Another interesting direction for future work would be to make the ETM and LPM approaches able to deal with lifecycle transitions, which would make the Indulpet Miner fully compatible with logs that have lifecycle transitions. In the case of IM, there already exists a version that is able to handle lifecycles [20]. Making the Indulpet Miner compatible with lifecycles amongst others makes it possible to calculate more accurate process performance statistics on the process model.

Acknowledgement We thank Joos Buijs for his help in integrating the Evolutionary Tree Miner (ETM) with the Indulpet Miner, and Eric Verbeek for coming up with its name.

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 Interdisc. Rew.: Data Mining and Knowledge Discovery **2**(2), 182–192 (2012). https://doi.org/10.1002/widm.1045
3. Augusto, A., Conforti, R., Dumas, M., La Rosa, M., Polyvyanyy, A.: Split miner: automated discovery of accurate and simple business process models from event logs. Knowledge and Information Systems pp. 1–34 (2018)
4. Breiman, L.: Bagging predictors. Machine learning **24**(2), 123–140 (1996)
5. vanden Broucke, S.K.L.M., De Weerdt, J.: Fodina: a robust and flexible heuristic process discovery technique. Decision Support Systems (2017)
6. Buijs, J.C.A.M., van Dongen, B.F., van der Aalst, W.M.P.: A genetic algorithm for discovering process trees. In: IEEE Congress on Evolutionary Computation. pp. 1–8. IEEE (2012)
7. Dahari, Y., Gal, A., Senderovich, A., Weidlich, M.: Fusion-based process discovery. In: International Conference on Advanced Information Systems Engineering. pp. 291–307. Springer (2018)
8. van Dongen, B.: BPI challenge 2012 dataset (2012). https://doi.org/10.4121/uuid:3926db30-f712-4394-aebc-75976070e91f
9. van Dongen, B.: BPI challenge 2015 dataset (2015). https://doi.org/10.4121/uuid:31a308ef-c844-48da-948c-305d167a0ec1
10. van Dongen, B., Borchert, F.: BPI challenge 2018 dataset (2018). https://doi.org/10.4121/uuid:3301445f-95e8-4ff0-98a4-901f1f204972
11. van Dongen, B.F., de Medeiros, A.K.A., Verbeek, H.M.W., Weijters, A.J.M.M., van der Aalst, W.M.P.: The ProM framework: A new era in process mining tool support. In: International Conference on Applications and Theory of Petri Nets. pp. 444–454 (2005). https://doi.org/10.1007/11494744_25
12. Goedertier, S., Martens, D., Vanthienen, J., Baesens, B.: Robust process discovery with artificial negative events. Journal of Machine Learning Research **10**(Jun), 1305–1340 (2009)
13. Günther, C.W., van der Aalst, W.M.P.: Fuzzy mining–adaptive process simplification based on multi-perspective metrics. In: International Conference on Business Process Management. pp. 328–343. Springer (2007)

14. Hoeting, J.A., Madigan, D., Raftery, A.E., Volinsky, C.T.: Bayesian model averaging: a tutorial. Statistical science pp. 382–401 (1999)
15. Leemans, S.J.J.: Robust process mining with guarantees. Ph.D. thesis, Ph. D. thesis, Eindhoven University of Technology (2017)
16. Leemans, S.J.J., Fahland, D., van der Aalst, W.M.P.: Discovering block-structured process models from event logs - a constructive approach. In: International Conference on Applications and Theory of Petri Nets and Concurrency. pp. 311–329. Springer (2013)
17. Leemans, S.J.J., Fahland, D., van der Aalst, W.M.P.: Discovering block-structured process models from event logs containing infrequent behaviour. In: International Conference on Business Process Management. pp. 66–78. Springer (2013)
18. Leemans, S.J.J., Fahland, D., van der Aalst, W.M.P.: Discovering block-structured process models from incomplete event logs. In: International Conference on Applications and Theory of Petri Nets and Concurrency. pp. 91–110. Springer (2014)
19. Leemans, S.J.J., Fahland, D., van der Aalst, W.M.P.: Using life cycle information in process discovery. In: Business Process Management Workshops. pp. 204–217 (2015). https://doi.org/10.1007/978-3-319-42887-1_17
20. Leemans, S.J.J., Fahland, D., van der Aalst, W.M.P.: Using life cycle information in process discovery. In: International Conference on Business Process Management. pp. 204–217. Springer (2015)
21. Leemans, S.J.J., Fahland, D., van der Aalst, W.M.P.: Scalable process discovery and conformance checking. Software and System Modeling **17**(2), 599–631 (2018). https://doi.org/10.1007/s10270-016-0545-x
22. de Leoni, M., Mannhardt, F.: Road traffic fine management process (2015). https://doi.org/dx.doi.org/10.1007/s00607-015-0441-1
23. Mannhardt, F.: Sepsis cases - event log (2018). https://doi.org/10.4121/uuid:915d2bfb-7e84-49ad-a286-dc35f063a460
24. Mannhardt, F., Tax, N.: Unsupervised event abstraction using pattern abstraction and local process models. In: Working Conference on Enabling Business Transformation by Business Process Modeling, Development, and Support. pp. 55–63. CEUR-ws.org (2017)
25. Munoz-Gama, J.: Conformance Checking and Diagnosis in Process Mining - Comparing Observed and Modeled Processes, Lecture Notes in Business Information Processing, vol. 270. Springer (2016). https://doi.org/10.1007/978-3-319-49451-7
26. Murata, T.: Petri nets: Properties, analysis and applications. Proceedings of the IEEE **77**(4), 541–580 (1989)
27. Object Management Group: Notation (BPMN) version 2.0. OMG Specification (2011)
28. Schapire, R.E.: The strength of weak learnability. Machine learning **5**(2), 197–227 (1990)
29. 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)
30. Tredup, R., Rosenke, C., Wolf, K.: Elementary net synthesis remains np-complete even for extremely simple inputs. In: International Conference on Application and Theory of Petri Nets and Concurrency. pp. 40–59 (2018). https://doi.org/10.1007/978-3-319-91268-4_3
31. Wolpert, D.H.: Stacked generalization. Neural networks **5**(2), 241–259 (1992)
32. van Zelst, S.J., van Dongen, B.F., van der Aalst, W.M.P.: Avoiding over-fitting in ILP-based process discovery. In: International Conference on Business Process Management. pp. 163–171. Springer International Publishing (2015)