Understanding complex process models by abstracting infrequent behavior

David Chapela-Campa, Manuel Mucientes and Manuel Lama

Version: accepted article

This is a POST-PRINT version, the final publication is available via
https://doi.org/10.1016/j.future.2020.07.030

Understanding Complex Process Models by Abstracting Infrequent Behavior

David Chapela-Campa*, Manuel Mucientes, Manuel Lama

Centro Singular de Investigación en Tecnoloxías Intelixentes (CiTIUS)
Universidade de Santiago de Compostela. Santiago de Compostela, Spain

Abstract

Process mining has become very popular in the last years as a way to analyze the behavior of an organization by offering techniques to discover, monitor and enhance real processes. A key point in process mining is to discover understandable process models. To achieve this goal in complex processes, several simplification techniques have been proposed, from the structural simplification of the model to the simplification of the log to discover simpler process models. However, obtaining a comprehensible model explaining the behavior of unstructured large processes —for instance containing hundreds of activities— is still an open challenge. In this paper, we introduce UBeA, a novel technique to abstract non-core behavior from a process model. We also present IBeA, a specific implementation of this proposal to simplify process models by abstracting infrequent behavior, using a frequent behavior extraction algorithm to detect the core behavior. IBeA has been validated with more than 10 complex real processes, most of them from the Business Process Intelligence Challenge (BPIC), showing that it simplifies the process obtaining a better process model than other simplification techniques.

Keywords: event abstraction, model simplification, log simplification, process mining, business process management

1. Introduction

During the past years process mining has emerged as a discipline focusing on techniques to discover, monitor and enhance real processes [1]. One of the key areas of process mining is process discovery, whose objective is to generate a process model describing the behavior of the event log of a process. Once a model is discovered, the analysis and enhancement of the process can be performed to detect possible improvements. However, in unstructured large processes —composed by many activities and where most trace variants[1] have a low frequency—, this analysis and enhancement become more difficult [3].

With the entrance of process mining in the *Big Data* era, these unstructured large processes have become more and more common. In these scenarios, process models use to have a poor *precision* in exchange of a high *fitness*, presenting a spaghetti-like structure [15, 26]. Many techniques have been developed focusing on subparts of the process [9, 10, 13, 17, 30], with the aim to extract as many information as possible. Nevertheless, in order to understand the main behavior in the model, and to be able to analyze and enhance the real process behind it, it is necessary to obtain a simplified process model with a trade-off between fitness and precision.

Although not focusing on unstructured large processes, different techniques have been developed to tackle the problem of simplification in complex processes. Some of these techniques simplify already discovered process models, applying transformations that reduce the model while maintaining the frequent behavior [15, 26]. The problem is that they usually produce simpler, but unstructured, models that deteriorate the understandability of the process. Other approaches opt to first simplify the log, and then obtain a finer process model using a discovery algorithm. Some of these techniques search for outliers in the log traces, removing them with the aim of retaining the frequent behavior of the process [11, 27]. However, the detection of outlier behavior in unstructured large scenarios is hindered by the high variability in the log. Related to this, another naive technique is the removal of the less frequent trace variants in the log. Nevertheless, the removal of full traces leads to a loss of frequent behavior —e.g., frequent subtraces— appearing in part of those traces. There are also approaches that detect the activities with higher contribution to the unstructured nature of the process, to remove them from the log [29]. But, the detection of these elements in a process where most activities follow a disordered distribution is not feasible.

To obtain an overall view of the process, another option could be to remove the infrequent behavior that is hindering the frequent one. But, the deletion of behavior creates inexistent paths in the process —each removal

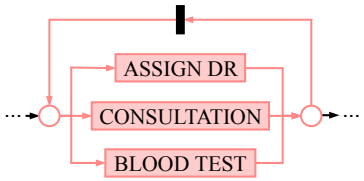
*Corresponding author
Email addresses: david.chapela@usc.es (David Chapela-Campa), manuel.mucientes@usc.es (Manuel Mucientes), manuel.lama@usc.es (Manuel Lama)

[1] The trace variants of a log are the different activity sequences, being their frequency the number of traces following each variant.

This is a POST-PRINT version, the final publication is available via
https://doi.org/10.1016/j.future.2020.07.030

Trace	
... ASSIGN DR - CONSULTATION - BLOOD TEST ...	(×21)
... ASSIGN DR - BLOOD TEST - CONSULTATION ...	(×27)
... BLOOD TEST - ASSIGN DR - ASSIGN DR ...	(×4)
... CONSULTATION - BLOOD TEST ...	(×7)
... ASSIGN DR - CONSULTATION - ASSIGN DR ...	(×5)
...	

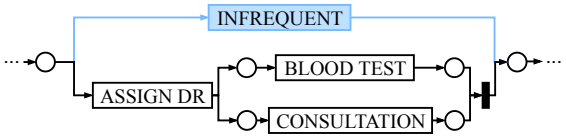
(a) Partial event log.



(b) Model for log in Figure 1a.

Trace	
... ASSIGN DR - CONSULTATION - BLOOD TEST ...	(×21)
... ASSIGN DR - BLOOD TEST - CONSULTATION ...	(×27)
... INFREQUENT ...	

(c) Abstracted event log.



(d) Model for log in Figure 1c.

Figure 1: Motivational example for the abstraction process presented in this paper.

creates a path from the previous events to the succeeding ones. Another alternative, instead of removing behavior, could be to abstract subprocesses in the log by replacing the execution of multiple activities with one [21, 23]. The key point of these techniques is to choose which subprocesses to abstract in order not to lose too much behavior w.r.t. the original process. In this paper we describe an approach to abstract the infrequent behavior into artificial activities, to obtain an overall view of the process, while being aware of the existence and exact location of the infrequent behavior.

Figure 1 shows a motivational example of this behavioral abstraction. It depicts a sample of a log concerning 3 activities occurring in many orders in Figure 1a, with its corresponding model in Figure 1b. In this type of scenarios, where all the activities occur in almost any order, it is usual to obtain a flower-like structure such as the one depicted in Figure 1b. But, in some cases, there is a latent structure hindered by the infrequent behavior. As can be seen, the common behavior is to first assign a doctor, and then perform a blood test either before or after going to the consulting room. This behavior cannot be observed without the abstraction due to the atypical cases hindering the visualization. Figure 1c shows that the infrequent behavior can be encapsulated into the `INFREQUENT` activity, obtaining the structure shown in Figure 1d. `INFREQUENT` can even store the abstracted subtraces to show, if the user zooms in, the encapsulated behavior.

This example shows an important feature of the abstraction process. The abstraction is performed by replacing the infrequent behavior —not the infrequent activities— into an artificial activity. As shown in Figure 1a, the three activities appear in both frequent —first two traces— and infrequent —highlighted traces— behavior. While other techniques would remove either the infrequent traces —also removing the other behavior present in them—, or the infrequent activities —removing activities in both frequent and infrequent contexts—, the behavioral abstraction only replaces the events of an activity when they appear in infrequent behavior.

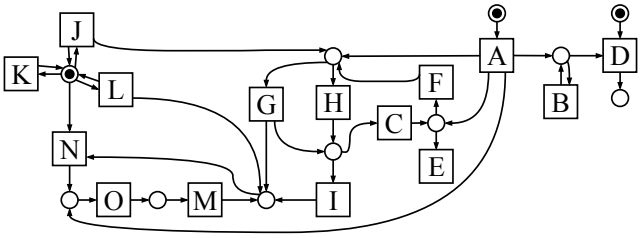
In this paper, we introduce UBeA, a novel technique to abstract the non-core behavior of a process into artificial activities using the relations between the activities, hence, taking into account structures such as parallels, selections or loops. The main novelty of UBeA is that it generates an abstracted version of the process describing the core behavior while being aware of the existence and exact location of the non-core behavior. Furthermore, UBeA allows the user to specify the behavior to maintain, i.e., the core behavior, making it very versatile. We also present IBeA, a specific implementation of UBeA to simplify process models by abstracting infrequent behavior, using WoMine [10] —an algorithm for extracting frequent subprocesses— to detect the core behavior —thus, considering the infrequent behavior as non-core—, allowing to produce a simpler process model while maintaining a trade-off between fitness and precision. IBeA has been validated with a set of 11 complex and real process logs, 10 of them from the Business Process Intelligence Challenge (BPIC), and one from the health domain. Experiments show that the simplification of IBeA generates better process models than other simplification techniques.

The remainder of this paper is structured as follows. Sec. 2 reviews the state of the art for the simplification of process models. Sec. 3 introduces some definitions and background knowledge. Sec. 4 presents the detailed structure of UBeA, followed by the implementation of IBeA in Sec. 5. Finally, Sec. 6 describes the evaluation of the approach, and Sec. 7 summarizes the conclusions of the paper.

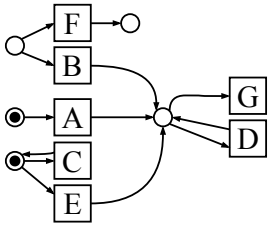
2. Related Work

Diverse techniques have been developed to tackle the simplification of process models. The first proposals focused on a structural simplification using only the information of the model itself [25]. But they quickly evolved to a simplification also using the information from the event log [15, 26]. In [15] an approach to simplify discovered process models while controlling the precision and generaliza-

This is a POST-PRINT version, the final publication is available via
https://doi.org/10.1016/j.future.2020.07.030



(a) A result example of the technique in [15].



(b) A result example of one of the techniques in [26].

Figure 2: Result examples of two model simplification techniques.

tion is presented. The process model, expressed in terms of a Petri net, is unfolded into a branching process using the event log, then filtered retaining the frequent parts, and finally folded again into a simpler process model capturing the desired behavior. Other approaches focusing on fitness and precision are, for instance, the collection of log-based techniques presented in [26]. They first rank the importance of the model places and arcs using the log, and then simplify with different alternatives maintaining the more important arcs and places.

These simplification techniques are designed to reduce the structural complexity of unstructured process models while maintaining the fitness. In unstructured large scenarios where most of the behavior is infrequent, understanding the process without a reduction in the fitness —derived from the exclusion of this infrequent behavior— is impossible. Furthermore, a drawback of these techniques is that they usually produce unstructured models that deteriorate the understandability of the process. These techniques are designed to simplify a Petri net by modifying its structure and producing nets such as the ones depicted in Figure 2. Although their fitness is good, these Petri nets can contain multiple source places and complex structures which simplify the model in terms of elements, but make more difficult the comprehension of the behavior occurring in it. They can also have source transitions —producing tokens without control—, and sink transitions —draining tokens from the net.

Other techniques simplify the event log before discovering the process to obtain a simpler process model [27, 29]. This decision allows to use discovery algorithms that obtain sound models. These simplifications of the log are performed in different ways. In [27], authors identify and remove outlier traces using the probability of occurrence of each event conditioned by both its k predecessors and its k successors. This allows to identify those events with a low probability of occurrence, based on its surrounding behavior, i.e., how probable is that an activity follows, or is followed by, a sequence of activities. The main drawback of this technique is that it removes full traces if an outlier is detected in them, also removing the frequent behavior they might contain. Furthermore, its detection relies on the sequential conditional probability, not being able to consider parallel relations.

In [29], Tax et al. present a set of techniques that remove, instead of full traces, activities from the entire log depending on their entropy. These techniques assign an entropy to each activity depending on their distribution of occurrence in the log, i.e., based on the directly-precedes and directly-follows relations among the activities, and remove the most chaotic activities from the log to simplify it. The main drawback of these techniques is that the removal of an activity from the entire log can produce a loss of important information if the activity appears in an unexpected context in some scenarios, but not in others —as shown in Figure 1. In addition, the calculation of the entropy of each activity depends on its relations with all the other activities, making it unscalable when the number of activities grows.

Another approach that overcomes some of the drawbacks of previous techniques is the abstraction of subprocesses of the process. This procedure consists in the replacement of a subprocess with a new activity, either in the log or structurally in the model. In [21] authors propose a supervised method to abstract, in the log, behavioral activity patterns that capture domain knowledge. First, they encode the behavior of the original log in activity patterns. Then, these patterns are composed in an abstraction model and, finally, they align the behavior of this abstraction model and the original log, creating an abstracted event log. The main drawback of this technique is the requirement of expert domain knowledge to define the patterns to be abstracted.

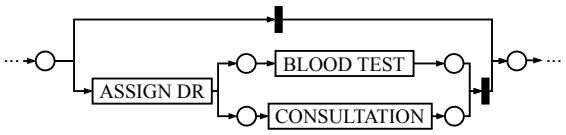
In [23], an unsupervised version of the method in [21] is proposed, using local process models [30] as patterns to abstract. In a first step they discover a fixed number of local process models —a behavioral activity pattern occurring frequently in the log— and rank them in terms of their diversity. Then, these local process models are used as activity patterns in the original abstraction method, and the abstracted log is created. The main drawback of this technique is that, in unstructured large scenarios where most of the behavior is infrequent, the abstraction of frequent subprocesses does not help to simplify the unstructured characteristic of the model. Furthermore, it penalizes significantly its quality due to the removal of behavior frequently executed in the log —this causes a higher impact in the fitness than removing infrequently executed behavior— and the addition of activities not recorded in the log.

Also in the abstraction research field, many techniques alter the granularity level of the data in order to abstract

This is a POST-PRINT version, the final publication is available via
https://doi.org/10.1016/j.future.2020.07.030

Trace	
... ASSIGN DR - CONSULTATION - BLOOD TEST ...	(×21)
... ASSIGN DR - BLOOD TEST - CONSULTATION ...	(×27)
... ...	

(a) Abstracted event log.



(b) Model for log in Figure 3a.

Figure 3: Result of removing the infrequent behavior from the log example in Figure 1a.

the low-level activities of a log into high-level activities that are more understandable to the user [5, 6, 16, 20, 22, 31]. These techniques, analyzed in [37], can be used to produce an abstracted event log, allowing to discover a model that describes a version of the process at a higher level. In complex processes with low-level activities, where the visualization is penalized by the high number and specificity of this type of activities —not by the variability of their behavior—, this kind of abstraction allows to understand better the process. However, in this paper we focus on understanding complex unstructured processes, where the complexity relies on the variability of their behavior, and where activities are considered as high-level activities. Our objective is to produce a simpler version of the same process to observe the frequent behavior, instead of abstracting all the activities to get a higher-level version of the process.

Following with the motivational example in Figure 1, an alternative to these related techniques, and to the proposal of this paper, is to detect the infrequent behavior and, instead of abstracting it, to remove the infrequent events (as shown in Figure 3). Nevertheless, a drawback of the removal of events is that it generates inexistent paths in the process —each removal creates a path from the previous events to the succeeding ones. For instance, in the example of Figure 3, the removal produces an artificial path allowing to skip the execution of the depicted activities. This generates an unreliable model where the user cannot be sure if a connection between two activities is real, or if the connection is replacing any removed infrequent behavior.

By contrast, the abstraction performed by IBeA allows to discover the hidden behavior in the process, while producing a reliable model. The generated process models show the existence and exact location of the infrequent behavior, allowing to observe the encapsulated behavior by treating the abstracted activities as subprocesses — allowing to zoom in and to inspect the replaced behavior.

In summary, current structural simplification techniques are able to reduce the complexity of the process models, but they make more difficult the comprehension of the behavior occurring in it. Some log simplification techniques remove infrequent full traces, which can cause a loss of the frequent behavior they might contain. Other log simplification techniques remove activities from the entire log, reducing the complexity when they appear in infrequent contexts, but also losing important information in the frequent contexts they might appear. Finally, current abstraction approaches focus on altering the granularity level of the activities, resulting in a higher-level version of the process. This produces a different process instead of a simplified version of the same process and, thus, cannot be used for the same purpose. All these techniques perform well in the scenarios they are designed for, but not in complex unstructured processes as the ones in which our paper focuses on.

3. Preliminaries

In this paper, we use place/transition Petri nets [12] (P/T Petri net) to represent process models due to its higher comprehensibility, and the easiness to explain the execution behavior. Furthermore, a Petri net can be automatically transformed into other process model notations such as BPMN, which are commonly used in business environments, and vice versa.

Definition 1 (Petri net). Let A be the set of activities of a process. A Petri net is a tuple $M = (P, T, F)$, where

- P is a finite set of places,
- T is a finite set of transitions,
- $P \cap T = \emptyset$,
- $F \subseteq (P \times T) \cup (T \times P)$ is a set of directed arcs.

A P/T Petri net (Def. 1) is a directed bipartite graph composed by two kinds of nodes: places and transitions — circles and boxes, respectively—, and where arcs connect two nodes of different type, as can be seen in Figure 4a. Each transition is identified by a label corresponding to the activity it represents. We assume that the transition labels are unique, i.e., there are no repeated activities in the net. Unlabeled transitions represent silent transitions, which are only executed for routing purposes and do not correspond to any activity of the process.

We denote as $\bullet t$ the input places and as $t\bullet$ the output places of $t \in T$ (according to F). In this paper, we consider only 1-safe Petri nets, defining its state with the marking function $m : P \nrightarrow \mathcal{P}(A)$[2]. m is a partial function returning, for each place $p \in P$, a set $\{\alpha \mid \alpha \in A\}$ of transition labels representing a token, or $\perp$ —bottom element— if there are no tokens in that place. The labels of a token correspond with the transitions which have produced it. Therefore, a transition t is said to be *enabled* if

[2] $\mathcal{P}(A) = \{A' \mid A' \subseteq A\}$ is the powerset of A.

This is a POST-PRINT version, the final publication is available via
https://doi.org/10.1016/j.future.2020.07.030

$\forall p \in \bullet t$, $m(p) \neq \perp$. The execution of an enabled transition t consumes a token in each $p \in \bullet t$, and produces one token with its label in each $p \in t\bullet$. Silent transitions propagate the labels of the consumed tokens and put them in the tokens it produces. The difference with a usual marking in P/T Petri nets is that the tokens carry the labels of its producing transition. This allows to know, when a transition is executed, which visible transitions have produced the tokens it consumed. Figure 4b shows an example of this marking performing the replay of a trace.

Definition 2 (Event). An event ε corresponds to the execution of the activity $\alpha \in A$ in a particular instant. In this simple definition, an event only specifies the name of the activity, but usually, events store more information as timestamps, resources, etc.

In this paper, events are represented only with the label of the executed activity to ease the comprehension. Nevertheless, it is important to distinguish between an activity —an action from a process that can be modeled with a single transition in the Petri net— and an event —a single execution of an activity. The replacement of an activity implies the replacement of all its events and the transition in the Petri net, but the replacement of an event only implies the replacement of that single execution.

Definition 3 (Trace). A trace is a list (sequence) $\tau = \langle\, \varepsilon_1, ..., \varepsilon_n \,\rangle$ of events ε_i occurring at a time index i relative to the other events in τ. Each trace corresponds to an execution of the process, i.e., a process instance.

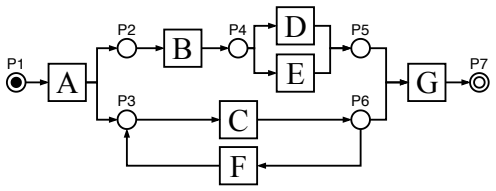
Definition 4 (Log). We define an event log $L = [\tau_1, ..., \tau_m]$ as a multiset of traces τ_i.

As an event represents the execution of an activity, there is an activity sequence $\langle\, \alpha_1, ..., \alpha_n \,\rangle$ corresponding to each trace. We use the term *trace variant* to refer to each unique activitiy sequence $\langle\, \alpha_1, ..., \alpha_n \,\rangle$ of a log L, being the frequency of a trace variant the number of traces in L with the same activity sequence.

As shown in Figure 4, the replay of a trace in a Petri net, using the marking previously commented, allows to extend the information of each event ε_i by identifying the activities which have produced the tokens consumed by ε_i. We use the term *behavioral event* (Def. 5) to refer to an event with this extended information —this information is similar to the input bindings in [2], but related to a Petri net. In this way, the replay of each trace of an event log in the Petri net allows to transform each event into a behavioral event, obtaining a list of behavioral traces (Def. 6), i.e., a behavioral event log (Def. 7).

Definition 5 (Behavioral Event). Let ε_i be the i-th event in a trace τ. Its corresponding behavioral event β_i is a tuple (S, α) where:

- $\alpha \in A$ is the activity executed in ε_i;
- $S = \{s \in \mathbb{N} \mid s < i\}$ is the set of indexes corresponding to the executions of each $\alpha' \in m(p)$ for all $p \in \bullet\alpha$, i.e., the indexes of the events producing the tokens consumed by the execution of α.



(a) Petri net example.

index	executed activity	current active marking	behavioral event
		$m_0(P1) = \{\}$	
0	A	$m_1(P2) = \{A\}$ $m_1(P3) = \{A\}$	$(\{\}, A)$
1	B	$m_2(P3) = \{A\}$ $m_2(P4) = \{B\}$	$(\{0\}, B)$
2	C	$m_3(P4) = \{B\}$ $m_3(P6) = \{C\}$	$(\{0\}, C)$
3	F	$m_4(P3) = \{F\}$ $m_4(P4) = \{B\}$	$(\{2\}, F)$
4	E	$m_5(P3) = \{F\}$ $m_5(P5) = \{E\}$	$(\{1\}, E)$
5	C	$m_6(P5) = \{E\}$ $m_6(P6) = \{C\}$	$(\{3\}, C)$
6	G	$m_7(P7) = \{G\}$	$(\{4, 5\}, G)$

(b) Replay of the trace $\langle\, A, B, C, F, E, C, G \,\rangle$.

Figure 4: Example of a Petri net and the replay of a trace performed to obtain the behavioral events forming the corresponding behavioral trace.

For instance, $(\{4, 5\}, G)$ in Figure 4b records the execution of the activity G caused by the behavioral events at indexes 4 and 5. Similar to an event, a behavioral event can store more information like timestamps, resources, etc. In what follows, we will let α^β and S^β denote, respectively, the elements α and S of the behavioral event β.

Definition 6 (Behavioral Trace). Let M be the Petri net of a process, and $\tau = \langle\, \varepsilon_1, ..., \varepsilon_n \,\rangle$ a trace of the same process. The corresponding behavioral trace of τ w.r.t. M is the sequence $\pi = \langle\, \beta_1, ..., \beta_n \,\rangle$ of behavioral events. π is the result of a replay of all $\varepsilon_i \in \tau$ in M, extending each

Behavioral Log
$\langle\, (\{\}, A), (\{0\}, B), (\{0\}, C), (\{2\}, F), (\{1\}, E), (\{3\}, C), (\{4, 5\}, G) \,\rangle$
$\langle\, (\{\}, A), (\{0\}, C), (\{0\}, B), (\{2\}, E), (\{1\}, F), (\{4\}, C), (\{3, 5\}, G) \,\rangle$
$\langle\, (\{\}, A), (\{0\}, B), (\{0\}, C), (\{1\}, D), (\{2, 3\}, G) \,\rangle$
$\langle\, (\{\}, A), (\{0\}, C), (\{1\}, F), (\{2\}, C), (\{0\}, B), (\{4\}, D), (\{3, 5\}, G) \,\rangle$

Table 1: Example of behavioral log for the model depicted in Figure 4a.

This is a POST-PRINT version, the final publication is available via
https://doi.org/10.1016/j.future.2020.07.030

ε_i by adding the indexes of the events corresponding to the execution of each $\alpha' \in m_i(p)$ for all $p \in \bullet\alpha$, being α the activity executed in ε_i —i.e., the indexes of the events producing the tokens consumed by ε_i.

Definition 7 (Behavioral Log). We define a behavioral event log, or behavioral log, as a multiset $\Pi = [\pi_1, ..., \pi_m]$ of behavioral traces π_i.

Table 1 shows an example of a behavioral log with four behavioral traces. Each behavioral trace is obtained by replaying the original trace in the process model depicted in Figure 4a, following the procedure explained in Figure 4b.

Definition 8 (Abstraction). Given a behavioral trace π, and being A_π the set of activities executed in π. We define an abstraction in π as $\lambda = (\beta, \mathcal{B}, A_I, A_O)$ where:

- β is a behavioral event representing the execution of an abstracted activity;
- $\mathcal{B}$ is a set of behavioral events from π to be replaced with β;
- $A_I \subset A_\pi$ is a set of activities of the events causing the execution of any event in $\mathcal{B}$;
- $A_O \subset A_\pi$ is a set of activities of the events in π whose execution is caused by events in $\mathcal{B}$,

such that:

- $A_I = \{\alpha^{\pi[s]} \in A_\pi \mid (\beta \in \mathcal{B}) \wedge (s \in S^\beta) \wedge (\pi[s] \notin \mathcal{B})\}$, for each behavioral event from $\mathcal{B}$, the activities of those behavioral events corresponding its source indexes and not contained in $\mathcal{B}$;
- $A_O = \{\alpha^\beta \in A_\pi \mid (\beta \in \pi) \wedge (\beta \notin \mathcal{B}) \wedge (\exists s \in S^\beta)[\pi[s] \in \mathcal{B}]\}$, the behavioral events in π having as source a behavioral event of $\mathcal{B}$.

For instance, in the example in Figure 4b, $((\{0\}, \alpha_{abs}), \{(\{0\}, B), (\{1\}, E)\}, \{A\}, \{G\})$ is the abstraction of the behavioral events $(\{0\}, B)$ and $(\{1\}, E)$, with input activity A, and output activity G, by the behavioral event $(\{0\}, \alpha_{abs})$. In the following, we will let β^λ, $\mathcal{B}^\lambda$, A_I^λ and A_O^λ denote, respectively, the elements β, $\mathcal{B}$, A_I and A_O of the abstraction λ.

Related to Def. 8, we use the term *empty abstraction*, represented by $\gamma = (\mathcal{B}, A_I, A_O)$, to define an abstraction that has no behavioral event assigned to it yet; and the term *anti-abstraction*, represented by $\widehat{\mathcal{B}}$, to define a set of behavioral events that remain in the abstracted log, i.e., events not to be abstracted. In the following, we will let $\mathcal{B}^\gamma$, A_I^γ and A_O^γ denote, respectively, the elements $\mathcal{B}$, A_I and A_O of the empty abstraction γ. We use the symbol $\beta \xrightarrow{\mathcal{B}} \beta'$ to indicate that β' is reachable from β through the behavioral events in $\mathcal{B}$ —following the relations present in the sources of each behavioral event.

4. UBeA Algorithm

In this section, we present UBeA (Alg. 1), an algorithm to abstract non-core behavior from a process model. Our proposal takes as input an event log, a Petri net and, for each trace, the indexes of the events not to be abstracted —i.e., the core behavior—, and produces as result the abstracted Petri net and the abstracted log, both with the non-core behavior encapsulated in new activities.

The first step of UBeA is to obtain the behavioral log with the causal relations of each event using the given log and model (Alg. 1: 2). Later, the algorithm builds the abstractions of the behavior not covered by the given event indexes (Alg. 1: 3). Then, the log is abstracted in function `abstractLog` (Alg. 1: 4) by replacing the behavioral events defined in each abstraction λ with the corresponding β^λ — the behavioral event of the artificial activity. Later, using the causal relations present in the abstracted behavioral log (c.f. Sec. 4.3), the abstracted Petri net is rediscovered (Alg. 1: 5). Finally, the abstracted behavioral log is transformed to an event log by keeping only the events —removing the behavioral information (Alg. 1: 6)—, and both the abstracted event log and the abstracted Petri net

Algorithm 1: UBeA algorithm.

Input: An event log $L = [\tau_1, ..., \tau_m]$ of traces, a Petri net M, and a sequence Υ with the indexes of the events not to be abstracted, where $\Upsilon[i]$ is the set of indexes of trace τ_i.
Output: An abstracted event log $L' = [\tau'_1, ..., \tau'_m]$ and an abstracted Petri net M' with the non-core behavior abstracted.

1 **Algorithm** `UBeA`(L, M, Υ)
2 $\quad\Pi \leftarrow$ replay of all $\tau \in L$ of M to obtain the behavioral log. `// Def. 7`
3 $\quad\Lambda \leftarrow$ `buildAbstractions`(Π, Υ)
4 $\quad\Pi' \leftarrow$ `abstractLog`(Π, Λ)
5 $\quad M' \leftarrow$ `rediscoverPetriNet`(Π') `// Sec. 4.3`
6 $\quad L' \leftarrow$ remove sources (S) from all $\pi \in \Pi$
7 $\quad$**return** (L', M')
8 **Function** `buildAbstractions`(Π, Υ)
9 $\quad\Gamma \leftarrow \emptyset$
10 $\quad$**forall** $\pi_i \in \Pi$ **do**
11 $\quad\quad\widehat{\mathcal{B}}_{\pi_i} \leftarrow \{\beta \in \pi_i \mid (\forall j \in \Upsilon[i])[\beta \neq \pi_i[j]]\}$
12 $\quad\quad\Gamma_{\pi_i} \leftarrow$ `obtainEmptyAbstractions`(π_i, $\widehat{\mathcal{B}}_{\pi_i}$) `// Alg. 2 (Sec. 4.1)`
13 $\quad\quad\Gamma \leftarrow \Gamma \cup \{\Gamma_{\pi_i}\}$
14 $\quad$**end**
15 $\quad\Lambda \leftarrow$`assignAbstractedEvents`(Γ) `// Alg. 3 (Sec. 4.2)`
16 $\quad$**return** Λ
17 **Function** `abstractLog`(Π, Λ)
18 $\quad\Pi' \leftarrow \emptyset$
19 $\quad$**forall** $\pi_i \in \Pi$ **do**
20 $\quad\quad\pi'_i \leftarrow \pi_i$
21 $\quad\quad$**forall** $\lambda_j \in \Lambda \mid \mathcal{B}^{\lambda_j} \subseteq \pi_i$ **do**
22 $\quad\quad\quad$replace in π'_i all $\beta \in \mathcal{B}^{\lambda_j}$ with β_j
23 $\quad\quad\quad$update source indexes for all $\beta \in \pi'_i$
24 $\quad\quad$**end**
25 $\quad\quad\Pi' \leftarrow \Pi' \cup \{\pi'_i\}$
26 $\quad$**end**
27 $\quad$**return** Π'

This is a POST-PRINT version, the final publication is available via
https://doi.org/10.1016/j.future.2020.07.030

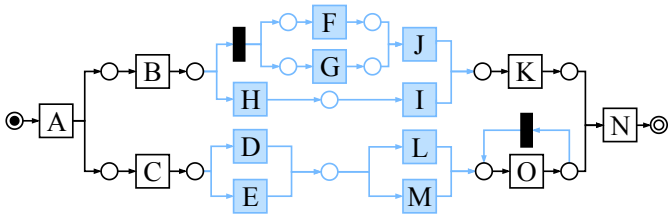
are returned.

The technique designed to build abstractions (Alg. 1: 8-16) is composed by two phases. The first one (Alg. 1: 10-14) is an horizontal analysis, i.e., one trace at a time, that generates the groups of behavioral events to be abstracted. For each trace, the behavioral events identified by the given indexes are collected in their anti-abstraction (Alg. 1: 11). Later, function `obtainEmptyAbstractions` (Alg. 1: 12) groups the behavioral events to be abstracted creating the empty abstractions —abstractions that have not yet been assigned an abstracted behavioral event— corresponding to that trace (c.f. Sec. 4.1). In the second phase (Alg. 1: 15), a vertical analysis of the log is performed to create the abstractions by assigning an abstracted event with the same activity to the empty abstractions with identical contextual behavior (c.f. Sec. 4.2) —i.e., having the same input or output connections. Then, with the information present in the abstractions, function `abstractLog` (Alg. 1: 17-27) abstracts the original behavioral log replacing the behavioral events of each abstraction with the corresponding abstracted behavioral event.

4.1. Create Abstractions from a Trace

The objective of the first phase is to create the empty abstractions that encapsulate the non-core behavior in each trace by grouping the corresponding behavioral events. Alg. 2 describes the abstraction process over a trace. First, the behavioral events to be abstracted are collected, i.e., those not present in the anti-abstraction (Alg. 2: 2). Then, these behavioral events that are connected between them are grouped (Alg. 2: 3). Afterward, an empty abstraction is created for each group (Alg. 2: 5-8), where function `obtainEmptyAbstraction` (Alg. 2: 19-23) returns *i)* $\mathcal{B}$, the set of behavioral events to be abstracted; *ii)* A_I, the input activities of this group; and *iii)* A_O, the output activities of this group.

Figure 5 shows an example where we take as core behavior only those events corresponding to behavior present



(a) Petri net of a process to abstract.

$\langle A, B, \boxed{F}, C, \boxed{D}, \boxed{G}, \boxed{J}, K, \boxed{L}, O, N \rangle$

(b) Trace example of the model in Figure 5a.

$\langle A, C, \boxed{E}, B, \boxed{L}, \boxed{O}, \boxed{O}, \boxed{H}, O, \boxed{I}, K, N \rangle$

(c) Trace example of the model in Figure 5a.

Figure 5: Petri net and two traces to exemplify the abstraction process, with the behavior to abstract highlighted in blue.

Algorithm 2: Get empty abstractions of a behavioral trace (Alg. 1: 12).

Input: A behavioral trace π and its anti-abstraction $\widehat{\mathcal{B}}_\pi$.
Output: A set Γ with the empty abstractions of the behavioral trace π.

1 **Algorithm** `obtainEmptyAbstractions`(π, $\widehat{\mathcal{B}}_\pi$)
2 $\quad\mathcal{B}_{ncore} \leftarrow \{\beta \in \pi \mid \beta \notin \widehat{\mathcal{B}}_\pi\}$
3 $\quad\mathcal{B}_{connected} \leftarrow$ `groupConnectedEvents`($\mathcal{B}_{ncore}$) `// set of sets of` β
4 $\quad\Gamma \leftarrow \emptyset$
5 $\quad$**forall** $\mathcal{B}_i \in \mathcal{B}_{connected}$ **do**
6 $\quad\quad\gamma_i \leftarrow$ `obtainEmptyAbstraction`(π, $\mathcal{B}_i$)
7 $\quad\quad\Gamma \leftarrow \Gamma \cup \{\gamma_i\}$
8 $\quad$**end**
9 $\quad$**return** Γ
10 **Function** `groupConnectedEvents`($\mathcal{B}_{ncore}$)
11 $\quad\mathcal{B}_{connected} \leftarrow \emptyset$ `// set of sets of` β
12 $\quad$**forall** $\beta_i \in \mathcal{B}_{ncore}$ **do**
13 $\quad\quad$**if** $\beta_i \notin \cup\mathcal{B}_{connected}$ **then**
14 $\quad\quad\quad\mathcal{B}' \leftarrow \{\beta_i\} \cup \{\beta' \in \mathcal{B}_{ncore} \mid \beta' \xrightarrow{\mathcal{B}_{ncore}} \beta_i \vee \beta_i \xrightarrow{\mathcal{B}_{ncore}} \beta'\}$
15 $\quad\quad\quad\mathcal{B}_{connected} \leftarrow \mathcal{B}_{connected} \cup \{\mathcal{B}'\}$
16 $\quad\quad$**end**
17 $\quad$**end**
18 $\quad$**return** $\mathcal{B}_{connected}$
19 **Function** `obtainEmptyAbstraction`(π, $\mathcal{B}$)
20 $\quad A_I \leftarrow \{\alpha^{\pi[s]} \in A_\pi \mid (\beta \in \mathcal{B}) \wedge (s \in S^\beta) \wedge (\pi[s] \notin \mathcal{B})\}$
21 $\quad A_O \leftarrow \{\alpha^\beta \in A_\pi \mid (\beta \in \pi) \wedge (\beta \notin \mathcal{B}) \wedge (\exists s \in S^\beta)[\pi[s] \in \mathcal{B}]\}$
22 $\quad\gamma \leftarrow (\mathcal{B}, A_I, A_O)$
23 $\quad$**return** γ

in all traces: the initial AND-split (A, B and C) and the final AND-join without the loop (K, O and N). Table 2 shows the results of the main steps of the first phase over the two traces of Figure 5. To create the groups with the connected behavioral events not present in the anti-abstractions —those without a circumflex in the trace description— the algorithm performs a forward iteration, adding each behavioral event to the same set of its inputs. $\mathcal{B}_{connected}$ contains the groups of behavioral events to abstract. Then, an empty abstraction is created for each group (e.g., γ_1) with the behavioral events of the group (e.g., $\{(\{1\}, F), (\{1\}, G), (\{2, 5\}, J)\}$), the input activities of these behavioral events (e.g., $\{B\}$), and the activities of the behavioral events from π whose inputs belong to the group (e.g., $\{K\}$). For instance, the input activity for γ_1 is only B because it is the behavioral event corresponding to the source indexes of F and G, and the behavioral events of the source indexes of J are inside the group. For the output activities, the algorithm searches for the behavioral events of π_1 for which the source indexes correspond to a behavioral event in the group (i.e., K).

4.2. Activity Assignment to Each Abstraction

Once each trace has its non-core behavior grouped in different empty abstractions, the second phase starts (Alg. 3). In this phase, all the empty abstractions of the log are compared to assign an event with the same activity to those with identical contextual behavior —coming from the same

This is a POST-PRINT version, the final publication is available via
https://doi.org/10.1016/j.future.2020.07.030

Algorithm 3: Assign an event with an abstracted activity to each empty abstraction (Alg. 1: 15).

Input: A set Γ of empty abstractions.
Output: The set Λ of abstractions with the events of the abstracted activities.

1 **Algorithm** `assignAbstractedEvents`(Γ)
2 $\quad\Gamma_I \leftarrow \emptyset$ `// set of sets of` γ `with identical inputs`
3 $\quad$**forall** $\gamma_i \in \Gamma$ **do**
4 $\quad\quad$**if** $(\gamma_i \notin \cup\Gamma_I)$ **then**
5 $\quad\quad\quad\Gamma' \leftarrow \{\gamma' \in \Gamma \mid A_I^{\gamma'} = A_I^{\gamma_i}\}$
6 $\quad\quad\quad\Gamma_I \leftarrow \Gamma_I \cup \{\Gamma'\}$
7 $\quad\quad$**end**
8 $\quad$**end**
9 $\quad\Gamma_O \leftarrow \emptyset$ `// set of those sets in` Γ_I `with identical outputs`
10 $\quad$**forall** $\Gamma_i \in \Gamma_I$ **do**
11 $\quad\quad$**if** $(\Gamma_i \notin \cup\Gamma_O)$ **then**
12 $\quad\quad\quad\Gamma' \leftarrow$ sets in Γ_I with identical output activities than Γ_i
13 $\quad\quad\quad\Gamma_O \leftarrow \Gamma_O \cup \{\Gamma'\}$
14 $\quad\quad$**end**
15 $\quad$**end**
16 $\quad\Lambda \leftarrow \emptyset$
17 $\quad$**forall** $\Gamma_i \in \Gamma_O$ **do**
18 $\quad\quad\alpha \leftarrow$ create new activity
19 $\quad\quad$**forall** $\gamma_j \in \Gamma_i$ **do**
20 $\quad\quad\quad S \leftarrow$ source indexes of all $\mathcal{B}^{\gamma_j}$ not pointing to an event in $\mathcal{B}^{\gamma_j}$
21 $\quad\quad\quad\beta \leftarrow (S, \alpha)$
22 $\quad\quad\quad\lambda \leftarrow (\beta, \mathcal{B}^{\gamma_j}, A_I^{\gamma_j}, A_O^{\gamma_j})$
23 $\quad\quad\quad\Lambda \leftarrow \Lambda \cup \{\lambda\}$
24 $\quad\quad$**end**
25 $\quad$**end**
26 $\quad$**return** Λ

activities or going to the same activities in the model. For this, the empty abstractions are first grouped by their input activities (Alg. 3: 3-8). Then, the groups that also have the same output activities are merged (Alg. 3: 10-15). Finally, an activity is created and assigned to each group of empty abstractions (Alg. 3: 17-25) —each groups contains the empty abstractions sharing the input and/or output activities.

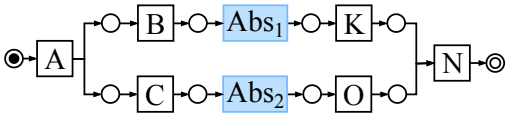
Continuing with the example in Table 2, the second phase groups all the empty abstractions, first by their input activities, obtaining two groups ($\{\lambda_1, \lambda_4\}$ and $\{\lambda_2, \lambda_3\}$), and, second, by their outputs, not merging any group because the output activities of the empty abstractions in the first group are $\{K\}$, and the output activities of the second group are $\{O\}$. Once the empty abstractions are grouped, the assignment of artificial activities is performed. A behavioral event with the activity Abs_1, and its corresponding source indexes, is assigned to the empty abstractions γ_1 and γ_4, and another behavioral event with activity Abs_2, and its corresponding indexes, to γ_2 and γ_3. Once the second phase of the algorithm is finished, the abstraction process in the log is performed by: *i)* removing the behavioral events of each abstraction; *ii)* inserting each abstracted behavioral event in the position of the last removed event;

$$\langle\ (\{\}, A), (\{0\}, B), (\{0\}, C), (\{1\}, Abs_1), (\{3\}, K), (\{2\}, Abs_2), (\{5\}, O), (\{4, 6\}, N)\ \rangle$$

(a) Abstracted behavioral trace of Figure 5b.

$$\langle\ (\{\}, A), (\{0\}, C), (\{0\}, B), (\{1\}, Abs_2), (\{3\}, O), (\{2\}, Abs_1), (\{5\}, K), (\{4, 6\}, N)\ \rangle$$

(b) Abstracted behavioral trace of Figure 5c.



(c) Abstracted Petri net for the process in Figure 5.

Figure 6: Petri net and two traces to exemplify the abstraction process.

and *iii)* updating the source indexes of the trace that have changed —if the first element of a list is removed, the indexes of the following elements change. Following this procedure, the behavioral traces of Figure 6 are obtained.

4.3. Petri net reconstruction

Once the abstraction process is performed, the abstracted Petri net can be rediscovered from the abstracted behavioral log. Each behavioral trace contains the causal relations among its activities and, thus, a Causal net (Def. 9) can be directly extracted from these relations.

Definition 9 (Causal net [2]). A Causal net (C-net) is a tuple $C = (A, a_i, a_o, D, I, O)$ where:

- A is a finite set of activities;
- $a_i \in A$ is the start activity;
- $a_o \in A$ is the end activity;
- $D \subseteq A \times A$ is the dependency relation,
- $AS = \{X \subseteq \mathcal{P}(A) \mid X = \{\emptyset\} \vee \emptyset \notin X\}$;
- $I \in A \to AS$ defines the set of possible input bindings per activity; and
- $O \in A \to AS$ defines the set of possible output bindings per activity,

such that:

- $D = \{(a_1, a_2) \in A \times A \mid a_1 \in \bigcup I(a_2)\}$;
- $D = \{(a_1, a_2) \in A \times A \mid a_2 \in \bigcup O(a_1)\}$;
- $\{a_i\} = \{a \in A \mid I(a) = \{\emptyset\}\}$;
- $\{a_o\} = \{a \in A \mid O(a) = \{\emptyset\}\}$;
- all activities in the graph (A, D) are on a path from a_i to a_o.

This is a POST-PRINT version, the final publication is available via
https://doi.org/10.1016/j.future.2020.07.030

$\tau_1 = \langle\ \widehat{A}, \widehat{B}, F, \widehat{C}, D, G, J, \widehat{K}, L, \widehat{O}, \widehat{N}\ \rangle$

π_1	$\langle\ (\{\}, A), (\{0\}, B), (\{1\}, F), (\{0\}, C), (\{3\}, D), (\{1\}, G), (\{2, 5\}, J), (\{6\}, K), (\{4\}, L), (\{8\}, O), (\{7, 9\}, N)\ \rangle$
$\mathcal{B}_{connected}$	$\{(\{1\}, F), (\{1\}, G), (\{2, 5\}, J)\}$ and $\{(\{3\}, D), (\{4\}, L)\}$
Γ	$\gamma_1 = (\{(\{1\}, F), (\{1\}, G), (\{2, 5\}, J)\}, \{B\}, \{K\})$ $\gamma_2 = (\{(\{3\}, D), (\{4\}, L)\}, \{C\}, \{O\})$

$\tau_2 = \langle\ \widehat{A}, \widehat{C}, E, \widehat{B}, L, O, O, H, \widehat{O}, I, \widehat{K}, \widehat{N}\ \rangle$

π_2	$\langle\ (\{\}, A), (\{0\}, C), (\{1\}, E), (\{0\}, B), (\{2\}, L), (\{4\}, O), (\{5\}, O), (\{3\}, H), (\{6\}, O), (\{7\}, I), (\{9\}, K), (\{8, 10\}, N)\ \rangle$
$\mathcal{B}_{connected}$	$\{(\{1\}, E), (\{2\}, L), (\{4\}, O), (\{5\}, O)\}$ and $\{(\{3\}, H), (\{7\}, I)\}$
Γ	$\gamma_3 = (\{(\{1\}, E), (\{2\}, L), (\{4\}, O), (\{5\}, O)\}, \{C\}, \{O\})$ $\gamma_4 = (\{(\{3\}, H), (\{7\}, I)\}, \{B\}, \{K\})$

Table 2: Key elements obtained in the first phase of the algorithm for the traces in Figure 5 —the circumflex indicates the events belonging to the anti-abstractions, π is the corresponding behavioral trace, $\mathcal{B}_{connected}$ is the set of groups of behavioral events to abstract, and Γ is the set of empty abstractions created from these groups.

The relations between the activities in a C-net are modeled by their bindings. Each activity has its input and output bindings —a set of sets of activities—, defining a possible binding with each set of activities. For instance, an activity D having $\{\{A, B\}, \{C\}\}$ as input bindings establishes that the input activities in an execution of D can be A and B, or only C; and an activity D having $\{\{E\}, \{F\}\}$ as output bindings establishes that an execution of D can cause the execution of E or the execution of F. This notion is strictly related to the information contained in the behavioral traces. Each behavioral event β contains an input binding in its sources (S^β). Thus, the input bindings of an activity α are the sets composed by the activities of the behavioral events of S^β for all β recording an execution of α. For instance, the input bindings of N w.r.t. the behavioral traces in Figure 6 are $\{K, O\}$ and $\{O, K\}$, hence, {{K, O}}.

To obtain the output bindings of each activity, each set in the output bindings of an activity α is composed by the activities of all the behavioral events having the index of the same execution of α in its sources (S). For instance, the index of the activity A is 0 in both behavioral traces in Figure 6. The output bindings of A are $\{B, C\}$ in Figure 6a and $\{C, B\}$ in Figure 6b. Thus, the output bindings of A are $\{\{B, C\}\}$.

With this procedure, the C-net corresponding to the abstracted process model can be extracted. Then, the transformation of the C-net to a Petri net can be performed as described in [2]. Finally, this Petri net is reduced removing unnecessary silent activities, obtaining the abstracted Petri net. Figure 6c shows the abstracted Petri net obtained by applying this process on the behavioral traces abstracted in previous subsections.

4.4. UBeA Complexity

The time complexity of UBeA is $\mathcal{O}(m \cdot n^2)$, where m is the number of traces of the log, and n the average number of events per trace.

5. IBeA: Abstract using WoMine

UBeA is independent of the algorithm that identifies the non-core behavior and can be used for many purposes. For instance, it can abstract subprocesses to observe relations among them; or, given a set of activities, abstract the remaining behavior to observe the interaction between these activities. Nevertheless, in this paper we focus in the abstraction of the infrequent behavior, as this behavior increases the complexity of the process. In this section, we present IBeA, an algorithm to simplify process models by abstracting the infrequent behavior. IBeA is based on UBeA combined with WoMine [10] to detect the frequent behavior not to be abstracted.

WoMine is an algorithm that extracts, from a process model, subprocesses which are frequently executed in the log. The algorithm performs an a priori search starting with the minimal structures of the process model, and expanding them until they become infrequent. The input elements of WoMine are an event log, a process model and a threshold. The result is the set of maximal subprocesses extracted from the model which are executed in a percentage of traces of the log higher than the defined threshold. The events corresponding to the execution of these subprocesses can be identified, serving as the input of UBeA as core behavior not to be abstracted.

5.1. IBeA Algorithm

Algorithm 4 shows the main structure of IBeA. The input of IBeA is composed by an event log, a Petri net and a frequency threshold. The first step of the algorithm is to obtain the frequent subprocesses using WoMine (Alg. 4: 2). Then, these subprocesses are filtered retaining only those with a size higher than one —being the size the number of arcs of the longest sequence in the subprocess—, or containing the start or end activity —to maintain this property in the simplified model (Alg. 4: 3). Later, for each trace, the event indexes concerning the execution of any filtered subprocess are stored as core behavior (Alg. 4: 6).

This is a POST-PRINT version, the final publication is available via
https://doi.org/10.1016/j.future.2020.07.030

Algorithm 4: IBeA algorithm.

Input: An event log $L = [\tau_1, ..., \tau_m]$ of traces, a Petri net M, and a threshold t.
Output: A simplified event log $L' = [\tau'_1, ..., \tau'_m]$ and a simplified Petri net M' with the infrequent behavior abstracted.

1 **Algorithm** `IBeA(`L, M, t`)`
2 $P \leftarrow$ `WoMine(`L, M, t`) // using alg. in [10]`
3 $P' \leftarrow \{p \in P \mid (size(p) \geq 1) \vee$ (p contains the initial or end activity)$\}$
4 $\Upsilon \leftarrow ()$ `// empty sequence`
5 **forall** $\tau_i \in L$ **do**
6 $v_i \leftarrow \{$index of $\beta \in \pi \mid \beta \in p'.executedEvents[\pi] \wedge p' \in P'\}$
7 $\Upsilon \leftarrow \Upsilon \cup \{v_i\}$
8 **end**
9 $(L', M') \leftarrow$ `UBeA(`L, M, Υ`) // Alg. 1`
10 **return** (L', M')

Finally, UBeA is run with the input log, Petri net, and indexes of the core behavior (Alg. 4: 9). The result of IBeA is composed by both the simplified log and the simplified Petri net.

An example of this process can be seen in Figure 5 where, assuming a balanced distribution in choices, and a frequency threshold of 70%, WoMine recovers as frequent subprocesses the initial AND-split (A, B and C) and the final AND-join without the loop (K, O and N). The events belonging to the execution of these subprocesses are taken as core behavior in the example in Sec 4.

In summary, IBeA allows to simplify both the process model and event log of a process by abstracting infrequent behavior. This simplification not only makes simpler and more understandable the process model, but gives the potential to apply techniques from all phases of process mining —discovery, analysis and enhancement— in order to improve the process.

5.2. IBeA Complexity

The time complexity of IBeA is the combination of UBeA and WoMine complexities: $\mathcal{O}(m \cdot n^2 + m \cdot n \cdot 2^p)$, where m is the number of traces in the log, n the average number of events per trace, and p the number of frequent activities in the process model.

6. Experimentation

In this section we evaluate the performance of IBeA with a set of experiments that have been executed in a computer with an Intel Core i7-2600 and 16GB of RAM[3].

6.1. Datasets

For this experimentation, 11 real logs have been used: one from the health domain —sepsis cases from a hospital [24]— and 10 from the Business Process Intelligence Challenge [28, 32, 33, 34]. To ensure all processes have a single start and a single end activity, all logs have been preprocessed by adding one artificial event of these activities at the start and end of each trace, respectively[4]. Also, all event names have been combined with their lifecycle in order to discern between different stages of the same activity (*START*, *COMPLETE*, etc.). The characteristics of these 11 logs, after this preprocessing, are shown in Table 3.

The abstraction performed by IBeA is designed to simplify the process model improving its simplicity by establishing a trade-off between fitness and precision. Nevertheless, this abstraction also penalizes these quality metrics. The fitness is penalized by the removal of supported behavior, i.e., the infrequent behavior obfuscating the process. The precision is also penalized by the addition of unsupported behavior, i.e., the abstracted activities. Although the fitness cannot be increased in any way, the precision penalization is commonly compensated by the removal of the infrequent behavior, which obfuscates the process visualization. However, there are scenarios where the penalization caused by the abstraction makes impossible to obtain a better process model in terms of these quality metrics.

Two log features are the most relevant to describe in which datasets the abstraction of infrequent behavior might be disruptive. One of these features is the number of activities. In processes where the number of activities is low —e.g., $\text{BPIC13}_{\text{clo}}$ and $\text{BPIC13}_{\text{op}}$—, the penalization caused by the inclusion of abstracted activities —not present in the original log— can be too high to compensate the simplification. The other feature is the percentage of the log covered by the most repeated activity sequences —the most frequent trace variants. In logs where few variants cover a high percentage of the log traces —logs with low trace variability—, the discovery of a model with those variants may already lead to a simpler process model difficult to overcome. Regarding this feature, note that the three most repeated variants in logs of $\text{BPIC12}_{\text{fin}}$, $\text{BPIC13}_{\text{clo}}$ and $\text{BPIC13}_{\text{op}}$ support more than the 40% of the traces in the log —as can be seen in the last three columns in Table 3.

6.2. Procedure

We have used the Inductive Miner [18] to discover, for each dataset, the process model that IBeA takes as input to perform the simplification. Regarding the configuration of IBeA, we have established a maximum size of 3 for the subprocesses that WoMine searches, retaining only the maximal subprocesses under this size. This limit allows to discover frequent subprocesses with up to three sequential arcs —with no limit for the number of parallel or selection branches—, and saves execution time.

[3] The algorithm, datasets and results can be downloaded from `http://tec.citius.usc.es/processmining/IBeA/`

[4] The addition of single start and single end activities does not alter the behavior of the process, and improves the model understandability by centralizing the start and end in one point.

This is a POST-PRINT version, the final publication is available via
https://doi.org/10.1016/j.future.2020.07.030

	#Traces	#Events	#Activities	Variants			
				#	% 1st	% 2nd	% 3rd
BPIC11	1143	152577	626	981	3.59%	1.49%	1.40%
$BPIC12_{fin}$	13087	288374	38	4366	26.20%	14.30%	2.07%
$BPIC13_{clo}$	1487	9634	9	327	32.62%	8.68%	7.40%
$BPIC13_{inc}$	7554	80641	15	2278	23.15%	6.94%	4.66%
$BPIC13_{op}$	819	3989	7	182	21.49%	15.02%	6.72%
$BPIC15_1$	1199	54615	400	1170	0.67%	0.50%	0.33%
$BPIC15_2$	832	46018	412	828	0.24%	0.24%	0.24%
$BPIC15_3$	1409	62499	385	1349	1.06%	0.85%	0.71%
$BPIC15_4$	1053	49399	358	1049	0.28%	0.19%	0.19%
$BPIC15_5$	1156	61395	391	1153	0.17%	0.17%	0.17%
Sepsis-cases	1050	17314	18	846	3.33%	2.29%	2.10%

Table 3: Characteristics of the logs used in the experimentation: number of traces (*#Traces*); number of events (*#Events*); number of activities (*#Activities*); number of variants —traces with the same activity sequence— (*Variants*), and the percentage of the log covered by the three variants with more traces.

We have compared our approach with two techniques from the related work (c.f. Sec. 2): *Matrix Filter*[5] [27], and *Activity Filter*[6] [29]. We have also considered a naive simplification technique such as retaining the variants with higher percentage of coverage —henceforth referred to as *Repetitions*. These techniques focus on the simplification of the event log, but they can be used to obtain a simplified process model if a discovery algorithm is executed afterwards. With this purpose, we have used the Inductive Miner Infrequent (IMf) [19] with 5 different thresholds (0%, 10%, 20%, 30% and 40%) [7] —being 0% the equivalent to use the original Inductive Miner. In this way, we can analyze which technique obtains the best results combined with a discovery algorithm. As IBeA also produces a simplified event log, we have included it in the set of techniques to be tested in combination with a discovery algorithm, naming this combination as IBeA-IMf.

All simplification techniques have been run in each dataset using 9 thresholds, from 10% to 90% with a step of 10. In IBeA, *Matrix Filter* and *Repetitions* this value establishes the threshold for the simplification. In *Activity Filter*, we have used it to denote the number of activities to retain. For instance, a threshold of 10% in a log with 50 activities corresponds with the simplified log containing the 5 less chaotic activities.

For each simplified model we have measured, using the original log, the fitness —Alignment-based fitness [4]—, precision —Negative Event Precision [8]— and simplicity —Weighted P/T average arc degree [7]. The use of the original log to measure the quality metrics penalizes the addition of abstracted activities performed by IBeA. Nevertheless, we have used the original log to prove that, even with this penalization, the abstraction of the infrequent behavior allows to obtain a better process model. With this comparison, the experimentation proves which algorithm performs the best simplification, either by producing the simplified process model or by simplifying the event log enabling to discover a better process model.

As we look for simple process models with a good trade-off between fitness and precision, facilitating the understanding of the frequent behavior occurring in the process —a model with an extremely low precision allows too many behavior not recorded in the log, obfuscating the real behavior—, we have summarized the fitness and precision in the F_{score} metric (Equation 1), penalizing low values in any of them:

$$F_{score} = 2 \cdot \frac{Fitness \cdot Precision}{Fitness + Precision} \quad (1)$$

For some models with hundreds of activities, the calculation of the precision takes a very long time. For this reason, we have established a lower bound for precision (0.05). When the precision value reaches this threshold, we stop the calculation and return a precision of 0.05.

Regarding the simplicity, we have normalized it to be expressed in values in $[0, 1]$ where, as the F_{score}, a greater value is better. The normalization (S_n) is calculated as

[5] Using the plugin *Matrix Filter* in ProM with `Mean` as the `Threshold adjusting Method`.

[6] Using the plugin *Activity Filter: Indirect Entropy optimized with Greedy Search* in ProM [14].

[7] We have also used other discovery algorithms such as Heuristics Miner [35] and ILP [36] among others, but the runtime and memory needed for the process discovery or to obtain the quality metrics make unfeasible to use these algorithms.

This is a POST-PRINT version, the final publication is available via
https://doi.org/10.1016/j.future.2020.07.030

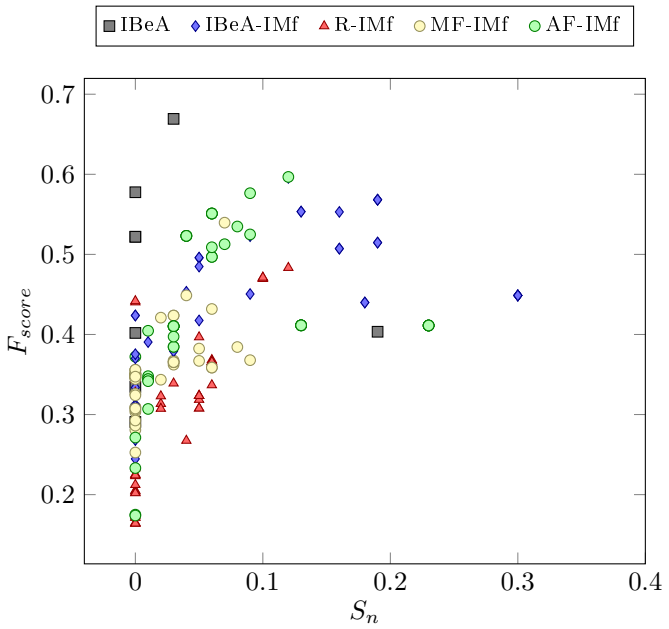


Figure 7: S_n vs. F_{score} of the models simplified by IBeA and the models discovered with the IMf algorithm using as input the simplified logs of the sepsis cases dataset obtained by IBeA, Repetitions (R), Matrix Filter (MF), and Activity Filter (AF).

shown in Equation 2.

$$S_n = 1 - \frac{min(S_{raw}, S_s)}{S_{raw}} \tag{2}$$

Where S_s is the simplicity to be normalized —i.e., the simplicity of the model discovered having as input the simplified log—, and S_{raw} is the simplicity of the best model —regarding F_{score} and simplicity— obtained with the IMf using the different thresholds and having as input the original log. With this normalization, S_n indicates the percentage of simplification of each model w.r.t. the best model obtained with the original log, taking a value of 0 if the simplified model is more complex than the original, i.e., if there is no simplification.

6.3. Results

As commented in Section 6.2, we obtain a set of simplified logs for each technique, and 5 simplified process models for each of these simplified logs. Figure 7 shows, for the sepsis cases dataset, the F_{score} and S_n of each model discovered by all the simplification techniques. For instance, each green circle denotes the F_{score} and S_n of each model discovered with the IMf using as input each of the logs simplified by the Activity Filter technique.

This chart depicts the typical results layout for most of the datasets, where there is no technique overcoming the others in both dimensions —for instance, the results of IBeA-IMf outperform the results of AF-IMf in terms of simplicity, but not in F_{score}. For this reason, to make a fairer analysis among the different techniques, we have compared the area covered by the dominant points of each technique. The dominant points are those overcoming each of all the other points in, at least, one dimension. For instance, a model with an $F_{score} = 0.4$ and an $S_n = 0.5$ is dominant if each of the other models has an $F_{score} < 0.4$ and/or an $S_n < 0.5$. Based on this, for each approach, the area covered by the dominant points encompasses the surface of all the models overcome in both dimensions by, at least, one of the results of the technique.

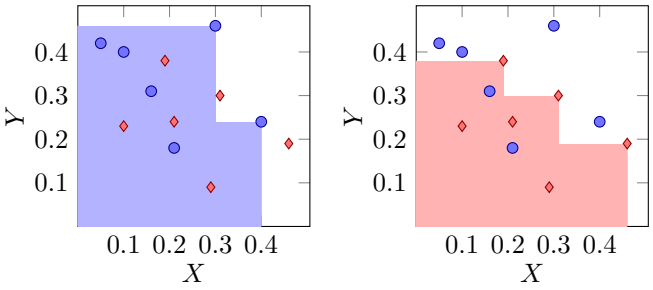


Figure 8: Example of the area covered by the dominant points in an artificial scenario with two techniques.

Figure 8 depicts an example of this area in an artificial scenario for two techniques. For instance, the dominant points of the blue technique —left— are those in $(X = 0.3, Y = 0.46)$ and $(X = 0.4, Y = 0.24)$; and the area covered by these points —i.e., the shadowed area— encompasses all the surface having a $X \leq 0.3$ and a $Y \leq 0.46$, or a $X \leq 0.4$ and a $Y \leq 0.24$.

Figure 9 shows, for each dataset, the area covered by the dominant points of the models obtained with each technique. $\text{BPIC13}_{\text{clo}}$ and $\text{BPIC13}_{\text{op}}$ datasets present a low trace variability in the log —more than the 40% of the traces in the log follow only three different activity sequences— and a low number of activities —less than 10— (c.f. Sec. 6.1). In these datasets, the simplified models obtained with the most frequent variants are difficult to overcome in terms of quality metrics, explaining the best results of a technique like *Repetitions*. Furthermore, due to the low number of activities, the addition of abstracted activities penalizes the quality metrics more than the simplification it performs. The simplicity of the abstracted models barely improves the simplicity of the original model, reducing drastically the area covered by the dominant points.

$\text{BPIC12}_{\text{fin}}$ contains more activities, but its trace variability is also very high —again more than the 40% of the traces in the log follow only three different activity sequences. Due to the higher number of activities, the penalization of adding abstracted activities is compensated by the simplification performed, and IBeA and IBeA-IMf overcome MF-IMf and AF-IMf. Nevertheless, the high variability allows `Repetitions` to obtain the best results. $\text{BPIC13}_{\text{inc}}$ also contains more activities than the other datasets from BPIC13, but it presents a lower trace variability —the 30% of the traces in the log follow only three different activity sequences. The trace variability is still

This is a POST-PRINT version, the final publication is available via
https://doi.org/10.1016/j.future.2020.07.030

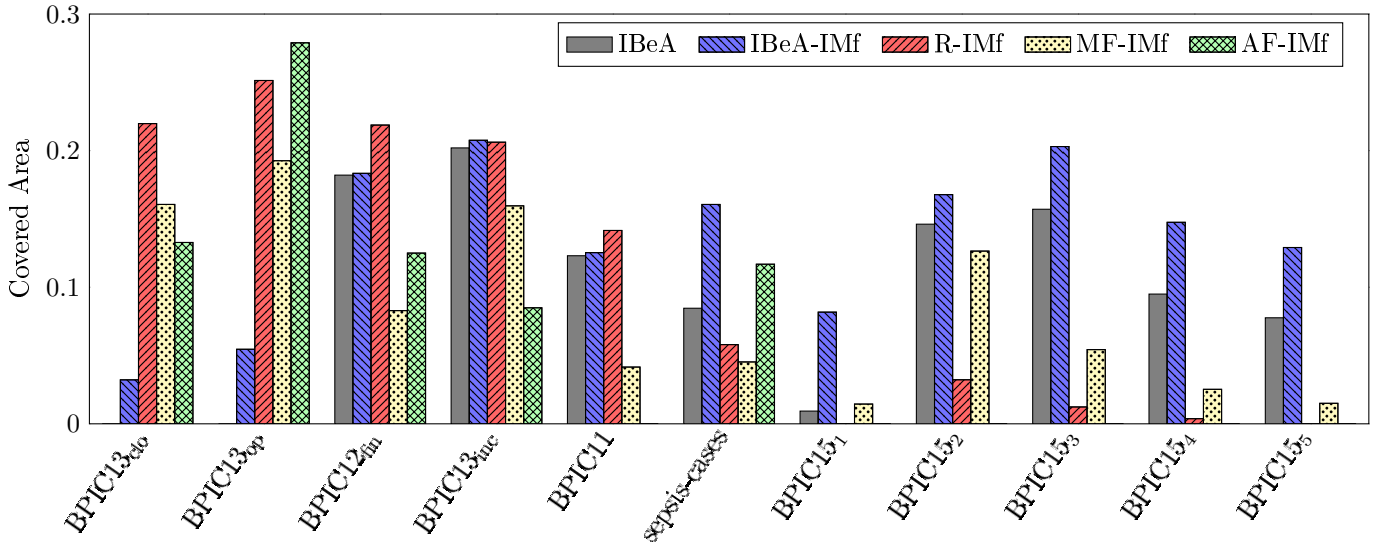


Figure 9: Area covered by the dominant points (S_n vs. F_{score}) of each simplification technique, for all datasets. The missing columns in some datasets are either because the simplification technique does not converge in a feasible time for that dataset —AF-IMf in BPIC11 and BPIC15 datasets— or because the simplified models are more complex than the models discovered with the original log —the other 4 missing columns.

high, explaining the good performance of `Repetitions`, but in this case the simplification of IBeA compensates its penalization enough to obtain the best results when combined with IMf.

In the BPIC11 dataset the best results are obtained by *Repetitions* despite the fact that the trace variability in the log is really high. This is a specific case where the 10% of more repeated traces contain enough common behavior to discover a simple and good process model. In fact, the other results of *Repetitions* —all thresholds except 10%— are worse than the results of all other techniques.

IBeA-IMf overcomes all the other techniques in the datasets with a high trace variability in the log, where a naive technique such as *Repetitions* is not useful —Sepsis-cases and the 5 logs from BPIC15. Although IBeA outperforms the other simplification techniques in most of these datasets, the best results are obtained by IBeA-IMf. The reason is that in these datasets the simplification of a complex process discovered with the original log is hampered by the infrequent behavior present in that log. This situation does not happen when the discovery algorithm uses the simplified log, because the relations of the model are built again.

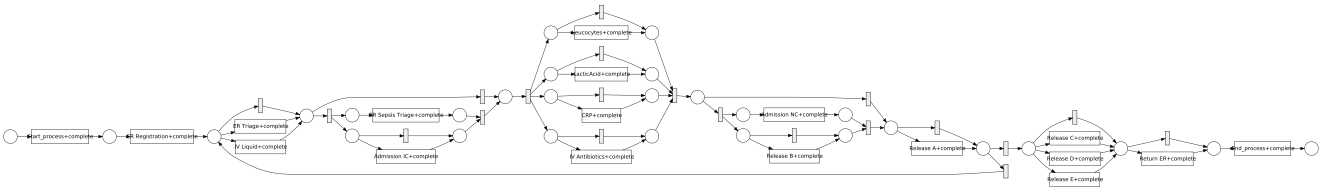
In summary, the simplification of the log performed by IBeA combined with the discovery of IMf outperforms the other simplification techniques in 7 datasets out of 11 —$\text{BPIC13}_{\text{inc}}$, Sepsis-cases, BPIC15_1, BPIC15_2, BPIC15_3, BPIC15_4 and BPIC15_5—, obtaining the second best result in another 2 datasets —$\text{BPIC12}_{\text{fin}}$ and BPIC11. As commented in Sec. 6.1, the datasets where IBeA does not obtain the best results contain a low trace variability in the log, and a naive technique such as *Repetitions* is the best option. Nevertheless, if the variability of the traces is high, the abstraction of IBeA is the best option for logs with both high (BPIC15) and low (Sepsis-cases) number of activities.

6.4. Visual results

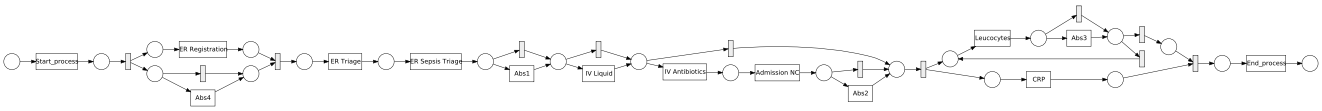
To show the potential of our technique, Figure 10 depicts, for two datasets, the best model discovered with the IMf algorithm having as input the original log, and one process model discovered with IMf having as input the log simplified by IBeA. Figure 10a depicts the best process model discovered using the original log in the sepsis cases dataset, having a fitness of 0.98 and a precision of 0.17. The model seems to be structured but, if we analyze the structure in the middle part, almost all activities are skippable. Furthermore, due to the loop of the silent activity going backwards, this structure allows to execute almost any activity, in any order and any number of times —the same behavior than a flower-like structure. This flower-like structure is necessary to have a fitness close to 1 in processes with high trace variability. On the other hand, if we previously abstract the infrequent behavior, decreasing the fitness to 0.75, we can observe the latent behavior in the process (Figure 10b), with only a couple of skippable activities and no loops producing flower-like structures, doubling the precision. As can be seen, the simplified model is better —w.r.t. F_{score} and simplicity— even though the quality metrics are being penalized by the addition of abstracted activities not present in the original log.

Figure 10c shows a more extreme case, depicting the spaghetti-like structure of the model obtained by the IMf algorithm having as input the BPIC15_3 dataset. To obtain a fitness close to 1 the model contains hundreds of activ-

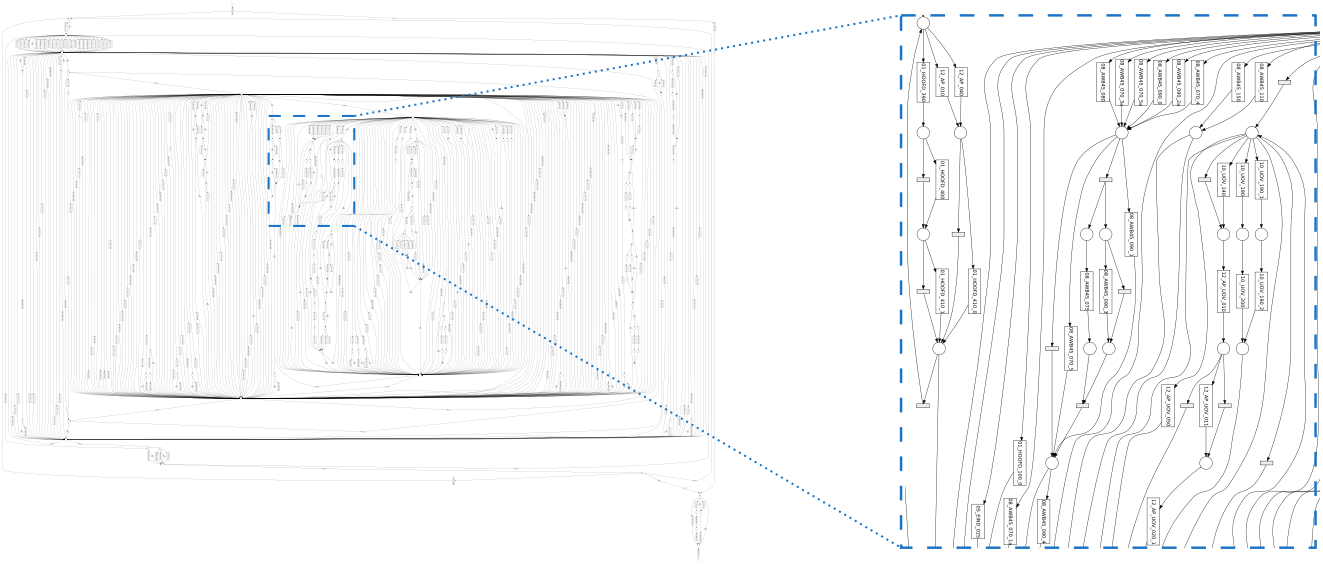
This is a POST-PRINT version, the final publication is available via
https://doi.org/10.1016/j.future.2020.07.030



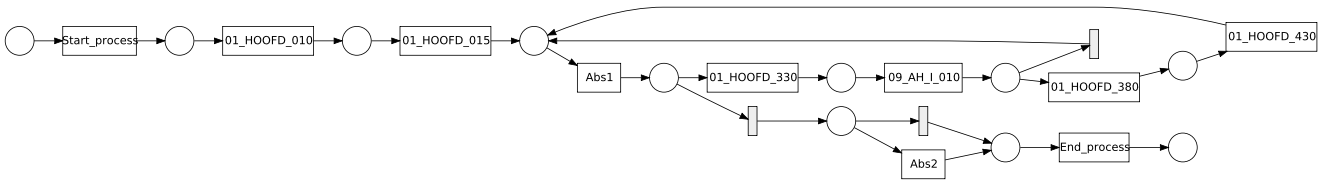
(a) Process model (fitness 0.98, precision 0.17, fscore 0.29) discovered with IMf with a threshold of 10% for the sepsis cases log.



(b) Process model (fitness 0.75, precision 0.32, fscore 0.45) discovered with IMf with a threshold of 40% for the sepsis cases log simplified with IBeA and a threshold of 40%.



(c) Process model (fitness 0.99, precision <0.05, fscore 0.1) discovered with IMf with a threshold of 40% for the BPIC15$_3$ log.



(d) Process model (fitness 0.24, precision 0.54, fscore 0.33) discovered with IMf with a threshold of 40% for the BPIC15$_3$ log simplified with IBeA and a threshold of 60%.

Figure 10: Visual examples of the difference in the discovered process models using the raw log or the log simplified with IBeA.

ities and connections among them, making impossible to understand what is happening in the process. In contrast, Figure 10d shows the structure of the frequent behavior: with a previous simplification of the log with IBeA and a threshold of 60% in the frequent behavior, a small process arises among the complex structure, encapsulating all the infrequent behavior in two abstracted activities (Abs_1, and Abs_2). In a process with hundreds of activities and a high trace variability such as BPIC15$_3$, the decrease in the fitness has to be higher in order to increase the precision and discover the frequent behavior in the process.

7. Conclusions and Future Work

In this paper, we have introduced the importance of a behavioral simplification in complex processes in order to understand what is happening in them. We have presented UBeA, a novel algorithm which, given an event log, a process model, and the events considered as core behavior, abstracts the remaining non-core behavior from the process model. We have also presented IBeA, a specific implementation of this algorithm to simplify process models by abstracting infrequent behavior, using WoMine [10] to detect the frequent behavior which is considered as core.

IBeA is able to detect the infrequent behavior which

obfuscates a process, abstracting it to produce simpler process models with a trade-off between fitness and precision, allowing to obtain an overall view of the process. The proposal also simplifies the event log, allowing to analyze and enhance the process with other process mining techniques. We have compared IBeA with other simplification approaches, using 11 logs from real scenarios, showing that IBeA is able to obtain, or allows to discover, a simpler and better model in complex processes.

As future work, the information provided by the abstracted activities can be exploited to enhance the process model. Each abstracted activity encapsulates a set of infrequent subtraces and, thus, different analysis might be applied to extract information from them. Therefore, that will allow to observe the main behavior of the process and, also, to better understand the process by inspecting the different infrequent subprocesses that were obfuscating its visualization.

Acknowledgments.

This research was funded by the Spanish Ministry of Economy and Competitiveness [TIN2017-84796-C2-1-R]; and the Galician Ministry of Education, Culture and Universities [ED431G/08]. These grants are co-funded by the European Regional Development Fund (ERDF/FEDER program). D. Chapela-Campa is supported by the Spanish Ministry of Education, under the FPU national plan (FPU16/04428).

References

[1] van der Aalst, W.M.P., 2016. Process Mining - Data Science in Action, Second Edition. Springer. doi:10.1007/978-3-662-49851-4.

[2] van der Aalst, W.M.P., Adriansyah, A., van Dongen, B.F., 2011. Causal nets: A modeling language tailored towards process discovery, in: Katoen, J., König, B. (Eds.), CONCUR 2011, Springer. pp. 28–42. doi:10.1007/978-3-642-23217-6_3.

[3] van der Aalst, W.M.P., Günther, C.W., 2007. Finding structure in unstructured processes: The case for process mining, in: Basten, T., Juhás, G., Shukla, S.K. (Eds.), ACSD 2007, IEEE Computer Society. pp. 3–12. doi:10.1109/ACSD.2007.50.

[4] Adriansyah, A., van Dongen, B.F., van der Aalst, W.M.P., 2011. Conformance checking using cost-based fitness analysis, in: EDOC 2011, IEEE Computer Society. pp. 55–64. doi:10.1109/EDOC.2011.12.

[5] Baier, T., Ciccio, C.D., Mendling, J., Weske, M., 2018. Matching events and activities by integrating behavioral aspects and label analysis. Software and Syst. Model. 17, 573–598. doi:10.1007/s10270-017-0603-z.

[6] Bernard, G., Andritsos, P., 2018. Cjm-ab: Abstracting customer journey maps using process mining, in: Mendling, J., Mouratidis, H. (Eds.), CAiSE 2018, Springer. pp. 49–56. doi:10.1007/978-3-319-92901-9_5.

[7] vanden Broucke, S.K.L.M., Weerdt, J.D., Vanthienen, J., Baesens, B., 2013. A comprehensive benchmarking framework (cobefra) for conformance analysis between procedural process models and event logs in prom, in: IEEE Symposium on Computational Intelligence and Data Mining, CIDM 2013, IEEE. pp. 254–261. doi:10.1109/CIDM.2013.6597244.

[8] vanden Broucke, S.K.L.M., Weerdt, J.D., Vanthienen, J., Baesens, B., 2014. Determining process model precision and generalization with weighted artificial negative events. IEEE Trans. Knowl. Data Eng. 26, 1877–1889. doi:10.1109/TKDE.2013.130.

[9] Chapela-Campa, D., Mucientes, M., Lama, M., 2017. Discovering infrequent behavioral patterns in process models, in: Carmona, J., Engels, G., Kumar, A. (Eds.), BPM 2017, Springer. pp. 324–340. doi:10.1007/978-3-319-65000-5_19.

[10] Chapela-Campa, D., Mucientes, M., Lama, M., 2019. Mining frequent patterns in process models. Inf. Sci. 472, 235–257. doi:10.1016/j.ins.2018.09.011.

[11] Conforti, R., Rosa, M.L., ter Hofstede, A.H.M., 2017. Filtering out infrequent behavior from business process event logs. IEEE Trans. Knowl. Data Eng. 29, 300–314. doi:10.1109/TKDE.2016.2614680.

[12] Desel, J., Reisig, W., 1996. Place or transition petri nets, in: Reisig, W., Rozenberg, G. (Eds.), Lectures on Petri Nets I: Basic Models, Advances in Petri Nets, Springer. pp. 122–173. doi:10.1007/3-540-65306-6_15.

[13] Diamantini, C., Genga, L., Potena, D., 2016. Behavioral process mining for unstructured processes. J. Intell. Inf. Syst. 47, 5–32. doi:10.1007/s10844-016-0394-7.

[14] van Dongen, B.F., de Medeiros, A.K.A., Verbeek, H.M.W., Weijters, A.J.M.M., van der Aalst, W.M.P., 2005. The prom framework: A new era in process mining tool support, in: Ciardo, G., Darondeau, P. (Eds.), ICATPN 2005, Springer. pp. 444–454. doi:10.1007/11494744_25.

[15] Fahland, D., van der Aalst, W.M.P., 2013. Simplifying discovered process models in a controlled manner. Inf. Syst. 38, 585–605. doi:10.1016/j.is.2012.07.004.

[16] Fazzinga, B., Flesca, S., Furfaro, F., Pontieri, L., 2018. Process discovery from low-level event logs, in: Krogstie, J., Reijers, H.A. (Eds.), CAiSE 2018, Springer. pp. 257–273. doi:10.1007/978-3-319-91563-0_16.

[17] Leemans, M., van der Aalst, W.M.P., 2014. Discovery of frequent episodes in event logs, in: Ceravolo, P., Russo, B., Accorsi, R. (Eds.), International Symposium on Data-Driven Process Discovery and Analysis (SIMPDA 2014), Springer. pp. 1–31. doi:10.1007/978-3-319-27243-6_1.

[18] Leemans, S.J.J., Fahland, D., van der Aalst, W.M.P., 2013a. Discovering block-structured process models from event logs - A constructive approach, in: Colom, J.M., Desel, J. (Eds.), PETRI NETS 2013, Springer. pp. 311–329. doi:10.1007/978-3-642-38697-8_17.

[19] Leemans, S.J.J., Fahland, D., van der Aalst, W.M.P., 2013b. Discovering block-structured process models from event logs containing infrequent behaviour, in: Lohmann, N., Song, M., Wohed, P. (Eds.), BPM 2013 International Workshops. Revised Papers, Springer. pp. 66–78. doi:10.1007/978-3-319-06257-0_6.

[20] de Leoni, M., Dündar, S., 2020. Event-log abstraction using batch session identification and clustering, in: Hung, C., Cerný, T., Shin, D., Bechini, A. (Eds.), ACM SAC 2020, ACM. pp. 36–44. doi:10.1145/3341105.3373861.

[21] Mannhardt, F., de Leoni, M., Reijers, H.A., van der Aalst, W.M.P., Toussaint, P.J., 2016. From low-level events to activities - A pattern-based approach, in: Rosa, M.L., Loos, P., Pastor, O. (Eds.), BPM 2016, Springer. pp. 125–141. doi:10.1007/978-3-319-45348-4_8.

[22] Mannhardt, F., de Leoni, M., Reijers, H.A., van der Aalst, W.M.P., Toussaint, P.J., 2018. Guided process discovery - A pattern-based approach. Inf. Syst. 76, 1–18. doi:10.1016/j.is.2018.01.009.

[23] Mannhardt, F., Tax, N., 2017. Unsupervised event abstraction using pattern abstraction and local process models, in: Gulden, J., Nurcan, S., et al. (Eds.), BPMDS 2017, CEUR-WS.org. pp. 55–63. URL: http://ceur-ws.org/Vol-1859/bpmds-06-paper.pdf.

[24] Mannhardt, F. (Felix), 2016. Sepsis cases - event log. doi:10.4121/UUID:915D2BFB-7E84-49AD-A286-DC35F063A460.

[25] Polyvyanyy, A., García-Bañuelos, L., Dumas, M., 2012. Struc-

turing acyclic process models. Inf. Syst. 37, 518–538. doi:10.1016/j.is.2011.10.005.

[26] de San Pedro, J., Carmona, J., Cortadella, J., 2015. Log-based simplification of process models, in: Motahari-Nezhad, H.R., Recker, J., Weidlich, M. (Eds.), BPM 2015, Springer. pp. 457–474. doi:10.1007/978-3-319-23063-4_30.

[27] Sani, M.F., van Zelst, S.J., van der Aalst, W.M.P., 2017. Improving process discovery results by filtering outliers using conditional behavioural probabilities, in: Teniente, E., Weidlich, M. (Eds.), BPM 2017 International Workshops. Revised Papers, Springer. pp. 216–229. doi:10.1007/978-3-319-74030-0_16.

[28] Steeman, W., 2013. BPI Challenge 2013. doi:10.4121/uuid:a7ce5c55-03a7-4583-b855-98b86e1a2b07.

[29] Tax, N., Sidorova, N., van der Aalst, W.M.P., 2019. Discovering more precise process models from event logs by filtering out chaotic activities. J. Intell. Inf. Syst. 52, 107–139. doi:10.1007/s10844-018-0507-6.

[30] Tax, N., Sidorova, N., Haakma, R., van der Aalst, W.M.P., 2016. Mining local process models. J. Innovation Digital Ecosyst. 3, 183–196. doi:10.1016/j.jides.2016.11.001.

[31] Tello, G., Gianini, G., Mizouni, R., Damiani, E., 2019. Machine learning-based framework for log-lifting in business process mining applications, in: Hildebrandt, T.T., van Dongen, B.F., Röglinger, M., Mendling, J. (Eds.), BPM 2019, Springer. pp. 232–249. doi:10.1007/978-3-030-26619-6_16.

[32] Van Dongen, B., 2011. Real-life event logs - hospital log. doi:10.4121/uuid:d9769f3d-0ab0-4fb8-803b-0d1120ffcf54.

[33] Van Dongen, B., 2012. BPI Challenge 2012. doi:10.4121/uuid:3926db30-f712-4394-aebc-75976070e91f.

[34] Van Dongen, B., 2015. BPI Challenge 2015. doi:10.4121/uuid:31a308ef-c844-48da-948c-305d167a0ec1.

[35] Weijters, A.J.M.M., Ribeiro, J.T.S., 2011. Flexible heuristics miner (FHM), in: IEEE Symposium on Computational Intelligence and Data Mining, CIDM 2011, IEEE. pp. 310–317. doi:10.1109/CIDM.2011.5949453.

[36] van Zelst, S.J., van Dongen, B.F., van der Aalst, W.M.P., 2015. Ilp-based process discovery using hybrid regions, in: van der Aalst, W.M.P., Bergenthum, R., Carmona, J. (Eds.), International Workshop on Algorithms & Theories for the Analysis of Event Data, ATAED 2015, CEUR-WS.org. pp. 47–61. URL: http://ceur-ws.org/Vol-1371/paper04.pdf.

[37] van Zelst, S.J., Mannhardt, F., de Leoni, M., Koschmider, A., 2020. Event abstraction in process mining: literature review and taxonomy. Granular Comput. doi:10.1007/s41066-020-00226-2.