Context-Based Activity Label-Splitting

Sebastiaan J. van Zelst[1,2(✉)], Jonas Tai[2], Moritz Langenberg[2], and Xixi Lu[3]

[1] Fraunhofer Institute for Applied Information Technology, Sankt Augustin, Germany
sebastiaan.van.zelst@fit.fraunhofer.de
[2] RWTH Aachen University, Aachen, Germany
{jonas.tai,moritz.langenberg}@rwth-aachen.de
[3] Utrecht University, Utrecht, The Netherlands
x.lu@uu.nl

Abstract. The information systems used in companies store event data describing the historical execution of the processes they support. Process mining covers the automated analysis of such data, generating insights that may ultimately lead to process improvement. A core branch of process mining is process discovery, dealing with event-data-based automated discovery of process models. In practice, the same activity may often be executed in a significantly different context, e.g., in a vaccination program, multiple vaccine doses are typically provided at different points in time. Process discovery algorithms assume that all executions of the same activity are to be mapped onto the same modeling element. Consequently, the presence of repeated activity executions under different contexts typically leads to underfitting discovered process models. To this end, activity label-splitting algorithms have been proposed to relabel the recordings of the same activity occurring in significantly different execution contexts. Yet, the state-of-the-art label-splitting algorithm adopts a trace-level-mapping strategy, yielding inferior results in the presence of loop constructs and infeasible computation time. Therefore, this paper proposes a novel label-splitting preprocessing technique that overcomes these issues. Our experiments confirm that our newly proposed label-splitting algorithm outperforms the state-of-the-art.

Keywords: Process mining · Process discovery · Label-splitting

1 Introduction

Most business processes executed in companies in various domains are supported by multiple, often interconnected, information systems. Among storing documents and artifacts, many such information systems, e.g., Enterprise Resource Planning (ERP) systems and Manufacturing Execution Systems (MES), store a digital representation of the historical execution of the processes they support.

C. Di Francescomarino et al. (Eds.): BPM 2023, LNCS 14159, pp. 232–248, 2023.
https://doi.org/10.1007/978-3-031-41620-0_14

Such data is referred to as *event data*. The intrinsic value of event data is confirmed by the successful application of event-data-driven analysis techniques in various domains, i.e., referred to as *process mining* [1].

In process mining, *process discovery*, which aims at the *automated discovery* of business process models describing the process as recorded, is one of the most prominent tasks [1]. Many process discovery algorithms have been proposed and studied in the literature [5]. While the discovery algorithms have made considerable progress and shown their values in real life, many other challenges remain largely unsolved. One of these challenges is the accurate handling *duplicated tasks* [1,5,19].

A *duplicated task* is manifested as a process task executed at different stages of the process, representing different activities. For example, the patient consultations at the beginning and end of a treatment trajectory are both called *consultations* but refer to different activities. When modeling such a process, a process analyst typically uses two task nodes to represent such duplicated tasks. However, when such tasks are executed, the corresponding events are recorded with the same label, i.e., *consultation*. Most existing discovery algorithms then consider these events to belong to the same activity and discover an overgeneralized loop to capture the behavior in the event log. To tackle this challenge, *activity label-splitting* techniques have been proposed [19,23]. Label-splitting techniques aim to detect the groups of events that refer to the same activity but are executed in a different context and, therefore, should be treated by the process discovery algorithm as conceptually different activities.

It is shown that label-splitting algorithms can significantly improve the quality of subsequently discovered process models [19], yet a relatively limited amount of work has been done in the area. The label-splitting technique proposed in [19] has shown that splitting the labels can lead to discovering more precise process models in some cases. However, the proposed approach uses a brute-force algorithm to find an optimal mapping of the events with the same labels between different traces, also called *trace mapping*, to detect candidate events for label-splitting. When two events have the same label in every trace in the log, this approach has to search 2^N possible mappings to find an optimal solution. As a result, the technique has a high time complexity (thus a poor running time) and has difficulties handling processes with loops. Other techniques use additional contextual information (such as the timestamps of the events) [23]. As a result, these approaches cannot handle an arbitrary log.

Therefore, we propose a novel label-splitting framework that is robust to looping behavior executed in the process and can handle real-life logs. The key artifact of our proposal is an *event graph* connecting all events that describe the same label and are candidates for label-splitting. Several techniques can be applied to detect clusters of equally labeled events with similar contexts, e.g., community detection [13]. The proposed event graph ignores the process instances in which the events occur and focuses on the execution context of the events (e.g., preceding and succeeding activities); the events of the same loop occur in a similar context and, thus, will be clustered automatically. As a result,

the approach can handle event logs from the processes with loops. We conducted an extensive range of quantitative experiments to assess our proposed framework. The results of our experiments show that our proposed approach consistently outperforms the state-of-the-art label-splitting preprocessing method.

The remainder of this paper is structured as follows. In Sect. 2, we discuss related work. Section 3 presents background concepts and the notation used in this paper. We present our main contribution in Sect. 4. In Sect. 5, we present the evaluation of our approach. Finally, Sect. 6 concludes this work.

2 Related Work

In this section, we discuss related work. We primarily focus on label-splitting. For a general overview of process mining, we refer to [1]. For an overview of existing process discovery algorithms, we refer to [5]. In terms of event data preprocessing techniques, next to label-splitting, we primarily identify two significant fields of study, i.e., *outlier and noise detection* [14], and *event abstraction* [27].

Various label-splitting methods exist that refine imprecise labels as a *preprocessing step*. Lu et al. [18,19] propose a label-splitting algorithm that refines event labels based on their context similarity by creating a mapping between process traces. The goal is to maximize the pairs of mapped events with similar contexts. For this, they use a cost function based on various aspects like neighbors of the events and location of the events in the trace is used. However, mapping complete traces cannot express the relationship between events within the same trace and leads to various issues in practice, most significantly for traces with loops. Our approach proposes to use an event graph that connects all events of the same label. This allows our approach to cluster the events of a loop and, thus, tackle this limitation.

Tax et al. [23] propose using the timestamps of events to perform label-splitting. The assumption is that when the events occurred at different times of the day, this may suggest the events carry a different meaning (e.g., eating during the morning versus eating during the evening). This leads to good results on the event logs (such as smart devices) that satisfy this assumption, yet, it does not apply to arbitrary process event logs, as this method requires a correlation between the time and execution of events in different contexts. Our approach does not have such assumptions and is generally applicable to any event log. In addition, our approach can be extended to also take additional context (such as timestamps) into account as features, which are used as input for clustering or community detection algorithms.

Another type of approach to label-splitting is the extension of existing process discovery algorithms. For example, Fodina [7] is an extension of the heuristic miner algorithm [25] that introduces a simple label-splitting based on the local context of events. There has also been an effort to extend the α-algorithm [4] to enable it to deal with duplicate tasks [17]. Another class of process discovery algorithms that can apply label-splitting are genetic process discovery algorithms [2,9]. This class of algorithms uses an *evolutionary computational*

paradigm to gradually learn process models, naturally supporting adding several modeling elements with the same label. Some *region-based process mining algorithms* also support label-splitting [3,10]. However, the emphasis of label-splitting in these algorithms is to enable the discovered model to describe all observed behavior in the input data. Consequently, these approaches need to take more contexts into account, which can lead to excessive label refinement. Another technique suggested by Vazquez-Barreiros et al. [24] discovers duplicate tasks in an already mined heuristic net or causal net. Finally, Yang et al. [26] use hidden Markov models to discover workflow models and split states during discovery. A general downside of all methods that integrate label-splitting within process discovery is that these methods are tied to their respective discovery algorithms and can not be used as a general preprocessing method. In contrast, our approach is independent of any discovery algorithm and can be seen as a preprocessing step of the event log.

3 Background

This section presents the background concepts used in this paper. After briefly presenting notational conventions, we introduce the notion of *event data*.

A sequence σ of length n over a set X is a function $\sigma\colon \{1,\ldots n\}\rightarrow X$. We write $\sigma = \langle x_1, x_2, \ldots x_n\rangle$, where $x_i = \sigma(i)$ for $1 \leq i \leq n$. The length of a sequence σ is denoted as $|\sigma|$. Given $1 \leq i < j \leq |\sigma|$, we let $\texttt{sub}(\sigma, i, j) = \langle \sigma(i), \ldots, \sigma(j)\rangle$, i.e., the strict sub-sequence of σ ranging form index i to j. We let X^* denote the set of all possible sequences over X. Given a sequence $\sigma \in X^*$, we let $\texttt{elem}(\sigma) = \{x | 1 \leq i \leq |\sigma| \wedge (\sigma(i) = x)\}$ to return all elements in σ.

We define an *event log* as follows. Consider Table 1, presenting a simplified example of an event log. Each row refers to an *event*, recording the execution of an activity, e.g., the first row represents a recording of the "Open Expense Report" activity. Each event has a *unique event identifier*. Similarly, each event has a unique *case identifier*, representing the process instance for which the activity was executed. Finally, a *timestamp* is recorded, recording the activity execution time. We formally the notion of event logs as follows.

Definition 1 (Event, Case, Event Log). *Let $\mathcal{C}$ denote the* universe of cases*, let $\mathcal{E}$ denote the* universe of events*, and let Σ denote the* universe of activity labels*. An* event *$e \in \mathcal{E}$ is a data tuple, recording the historical execution of an activity. We assume that at minimum, an event describes:*

- *An* activity attribute*, accessed by $\pi_{act}(e) \in \Sigma$,*
- *A* timestamp attribute*, accessed by $\pi_{time}(e) \in \mathbb{R}^+$.*[1]

A case *$c \in \mathcal{C}$ records an instance of the process and describes a collection of events, i.e., $\pi_{events}(c) \subseteq \mathcal{E}$. An event log L is a collection of cases, i.e., $L \subseteq \mathcal{C}$.*

[1] We assume that, for $t_0, \Delta \in \mathbb{R}^+$, every timestamp t can be represented as $t = t_0 + \Delta$.

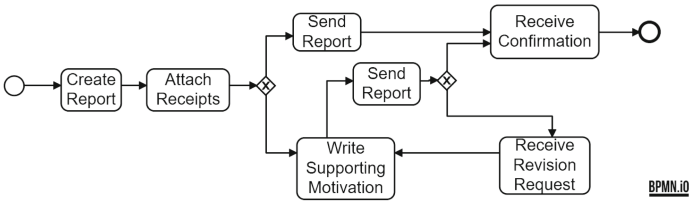
Table 1. Simple event log describing recorded process behavior. The event log captures at what point in time an activity was executed for a specific case.

Event ID	Case ID	Activity	Timestamp
1	1	Open Expense Report	26-10-2022 9:40 AM
2	1	Attach Receipts	26-10-2022 9:42 AM
3	1	Send Report	26-10-2022 9:43 AM
4	2	Open Expense Report	26-10-2022 10:21 AM
5	2	Attach Receipts	26-10-2022 10:27 AM
6	2	Write Supporting Motivation	26-10-2022 10:35 AM
7	2	Send Report	26-10-2022 10:42 AM
8	2	Receive Revision Request	26-10-2022 5:25 PM
9	2	Write Supporting Motivation	27-10-2022 9:45 AM
10	2	Send Report	27-10-2022 9:53 AM
11	1	Receive Confirmation	27-10-2022 11:13 AM
12	1	Close Report	27-10-2022 11:14 AM
13	2	Receive Confirmation	28-10-2022 11:18 AM
14	3	Open Expense Report	29-10-2022 11:22 AM
15	3	Attach Receipts	29-10-2022 11:28 AM
16	3	Write Supporting Motivation	29-10-2022 11:36 AM
17	3	Send Report	29-10-2022 11:43 AM
18	3	Receive Revision Request	29-10-2022 3:20 PM
19	3	Write Supporting Motivation	31-10-2022 3:55 PM
20	3	Send Report	31-10-2022 4:27 PM
21	3	Receive Confirmation	31-10-2022 5:16 PM
⋮	⋮	⋮	⋮

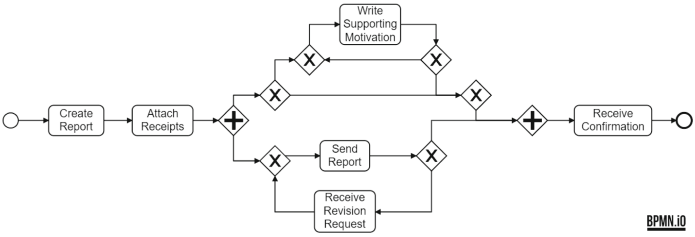
We write $\hat{c}$ as a shorthand notation for $\pi_{\texttt{events}}(c) \subseteq \mathcal{E}$. In the context of this paper, we assume that a total order is deterministically available for a case, i.e., $\texttt{seq}(c) \in \hat{c}^*$, s.t., $\texttt{elem}(\texttt{seq}(c)) = \hat{c}$, $|\texttt{seq}(c)| = |\hat{c}|$ and:
$\forall 1 \leq i < j \leq |\texttt{seq}(c)| \, (\pi_{\texttt{time}}(\texttt{seq}(c)(i)) \leq \pi_{\texttt{time}}(\texttt{seq}(c)(j)))$. We also assume that events occur uniquely in one case in an event log, i.e., $\forall c, c' \in L(\hat{c} \cap \hat{c'} \neq \emptyset \implies c = c')$.

4 Event-Graph-Based Label-Splitting

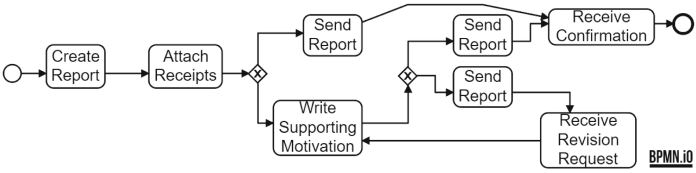
In this section, we present our novel proposed framework for activity label-splitting. In Sect. 4.1, we present a motivating example, which we use as a running example in the remainder of the paper. In Sect. 4.2, we present an overview of our proposed framework. Section 4.3 presents the construction of the event graph, i.e., the foundational artifact of our approach. Section 4.4 briefly discusses graph clustering. We present the relabeling mechanism in Sect. 4.5.



(a) Ground truth process model (in BPMN modeling formalism) describing the normative behavior of the running example process. Our proposed approach successfully leads to *rediscovery* of the ground truth model.



(b) Process model automatically discovered on the running example event data, using the Inductive Miner algorithm [15].



(c) Process model automatically discovered on the running example event data, using the Inductive Miner algorithm [15] in combination with the state-of-the-art label-splitting algorithm [19].

Fig. 1. Example used in this paper, describing a simple compensation request.

4.1 Motivating Example

To motivate our proposed approach and to ease the readability of this paper, we explain the steps of our framework using a motivating running example. We consider a (simplified) reimbursement process. Consider Fig. 1a, in which we depict a process model (using the BPMN modeling formalism) describing the reimbursement process. Firstly, a report is created. Then, receipts are attached to the report. If the total sum of the reimbursement claim is below $500, the report is directly submitted. Subsequently, the report is automatically accepted, and a confirmation is sent out to the applicant. If the sum of the claim is above $500, the applicant writes a supportive motivation, after which the report is submitted. The applicant either receives a confirmation or a revision request.

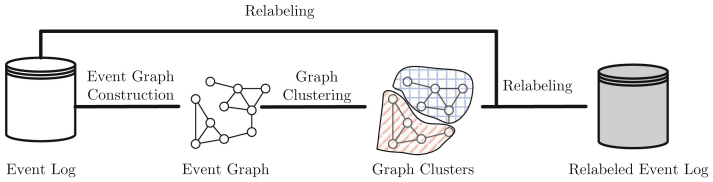


Fig. 2. Schematic overview of the proposed approach. The event log is converted to an event graph in which equally-labeled similar events are connected. Community detection is applied to detect events that depict similar behavioral contexts.

The applicant must revise and resubmit the supporting motivation if a revision request is received. Table 1 captures three executions of the process described.

State-of-the-art approaches for process discovery inaccurately handle event data describing the behavior of Fig. 1a. When applying the Inductive Miner algorithm [15] on a noise-free event log based on the model in Fig. 1a, we obtain the process model depicted in Fig. 1b. Since the discovery algorithm maps all occurrences of the `send report` activity on the same model element, the model is severely underfitting, i.e., it describes many more execution sequences than the process's reference model. When applying the state-of-the-art label-splitting algorithm [19], we obtain the process model depicted in Fig. 1c.[2] Whereas the model discovered by applying the label-splitting algorithm of [19] is *language equivalent* to the ground truth model (cf. Fig. 1a), it does have conceptual quality issues. The model falsely suggests that two distinct "Send Report" activities are possible after writing the supporting motivation, yielding a different outcome. As modeled in the ground truth model, the decision point of confirmation or requiring another revision is made after receiving the report. In contrast, our newly proposed algorithm can, when splitting the "Send Report" activity, rediscover *rediscover* the ground truth model (cf. Fig. 1a).

4.2 Overview

This section presents an overview of our proposed framework for activity label-splitting. Consider Fig. 2, in which we schematically present the basic steps of our framework. The input of our framework is an *event log*. We assume that some activity label $a \in \Sigma$, i.e., we aim to split label a, has been determined in advance by a domain expert. As a first step, the event log is converted into an *event graph* where all events e in the log that describe activity a ($\pi_{\texttt{act}}(e) = a$) form the vertices of the graph. If they are significantly similar, given some arbitrary context, two vertices are connected. Generally, the context and similarity function are parameters of the approach. Examples include, among others, the resource executing the event, the activities preceding and succeeding the event,

[2] The default implementation of the algorithms falsely splits the event data on "Write Supporting Motivation". The model in Fig. 1c is closest to the ground truth model (Fig. 1a) and is obtained by using a custom parameterization of the algorithm.

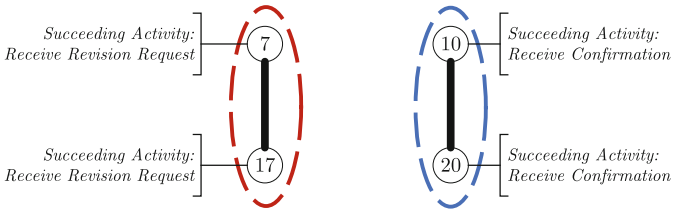


Fig. 3. Example event graph, based on the running example event log (cf. Table 1). The vertices contain the event ids, and the similarity context used is the succeeding activity within the case that the event belongs to. Vertices 7 and 17 have a similar context. Similarly, vertices 10 and 20 have a similar context.

etc. We apply graph clustering on the graph to detect groups of equally labeled events that occur in a different context. All events in the same cluster obtain a "fresh" activity label.

4.3 Event Graph Construction

In this section, we describe the first step of our approach, i.e., *event graph construction*. Events that have the same activity label, i.e., $e, e' \in \mathcal{E}$ s.t. $\pi_{\texttt{act}}(e) = \pi_{\texttt{act}}(e')$, form the vertices of the graph. Two events vertices are connected if, for some *context-based symmetrical similarity function*, their corresponding events are significantly similar. Generally, such a similarity function can be any contextual data feature recorded for the events, e.g., the two activities may be executed by the same resource, the two activities may require the same input document, etc. From a formal perspective, we require the similarity metric to be symmetric.

As a simple example, reconsider Table 1. Observe that the activities "Write Supporting Motivation" and "Send Report" are executed twice for both case 2 and case 3. We decide to apply label-splitting on the "Send Report" activity, hence, the events describing said activity form the vertices in the event graph (events 7, 10, 17, and 20). For simplicity, assume that we use each event's direct succeeding activity as a context (within the same case). For events 7 and 17, followed by events 8 and 18, respectively, the succeeding activity is "Receive Revision Request". Similarly, events 10 and 20 are followed by events 13 and 21, respectively, which both describe the "Receive Confirmation" activity. If we only connect those events in the event graph with the exact same context, i.e., succeeding activity, we obtain the graph depicted in Fig. 3. We define the notion of an event graph as follows.

Definition 2 (Event Graph). *Let $V \subseteq \mathcal{E}$ be a collection of events. Let $\varphi\colon \mathcal{E}\times\mathcal{E}\to[0,1]$ be a* context-based symmetric similarity function *on the universe of events and let $t_s \in [0,1]$. We let $G_{(\varphi,t_s)} = (V,E)$ be an undirected graph, referred to as the (φ,t_s)-driven event graph of L, where $\{v,v'\} \in E$ iff $\varphi(v,v') \geq t_s$.*

The exact characterization of the similarity function φ is a parameter of our approach, i.e., it depends on the attributes available in the event data as well as the nature of the process and its corresponding logging. Hence, we refrain from providing formal definitions of instantiations of φ, yet, since we assume that at least an activity and timestamp attribute are available, we provide an example instantiation based on control flow. Additionally, note that, in certain scenarios, the φ-value can be used as a weight function on the edges of the event graph.

Let $\sigma = \texttt{seq}(c)$ for some $c \in \mathcal{C}$ denote a sequence of events (recall $\texttt{seq}(c) \in \mathcal{E}^*$), let $1 \leq i \leq |\sigma|$ and let $k \in \mathbb{N}$. The sequence $\texttt{sub}(\sigma, \max(1, i-k), \max(1, i-1))$ describes the preceding k events of the i^{th} event in σ. Similarly, $\texttt{sub}(\sigma, \min(i+1, |\sigma|), \min(i+k, |\sigma|))$ describes the succeeding k events of the i^{th} event in σ. Clearly, given some $c' \in \mathcal{C}$ with $\sigma' = \texttt{seq}(c')$ and $1 \leq j \leq |\sigma'|$, we compare the k preceding events of event $\sigma(i)$ in σ with the k preceding events of event $\sigma'(j)$ in σ', e.g., by computing the *edit distance* between $\texttt{sub}(\sigma, \max(1, i-k), \max(1, i-1))$ and $\texttt{sub}(\sigma', \max(1, j-k), \max(1, j-1))$. The same can be applied for the k succeeding events. Both distances can subsequently normalized and a weighted average can be computed. Several variations of the above scheme are possible. For example, the event graph in Fig. 3 uses $k = 1$ and ignores the preceding activities.

4.4 Graph Clustering

The second step of our approach entails *global graph clustering* [21]. Any algorithm that computes a *partitioning* of the vertices of an undirected graph based on the graph's topological structure is applicable. For example, *connected components* is used by the approach of [19]. However, we found that *community detection* leads to better results which is why we decided to focus on this clustering method.

Community Detection; Community detection algorithms [13] detect clusters in which the intra-connectivity of vertices of a cluster is high, and the inter-connectivity to vertices in a different cluster is low. Applying community detection allows for detecting clusters, even if the graph is connected. For example, consider the schematic example graph in Fig. 2. The graph itself is connected, yet, two separate communities are identifiable.

Observe that, in the context of our running example, most clustering techniques find the two vertex clusters visualized in Fig. 3, i.e., events 7 and 17, and, events 10 and 20 are grouped together.

4.5 Relabeling

In the final step of our framework, we *relabel* the events that form the clusters in the event graph. In particular, all events belonging to the same cluster obtain the same activity label. Let $G_{(\varphi,k)} = (V, E)$ be a (φ, k)-driven event graph of some event log L, similarity function φ, and threshold k. Further, assume that a clustering algorithm of choice resulted in a partitioning $\mathcal{V} = \{V_1, V_2, \ldots, V_n\}$. To relabel the events in the event log, we return a relabeling

function $\lambda: V \rightarrow \Sigma$, s.t., $\lambda(e) \neq$ '$\pi_{\texttt{act}}(e)$, $\forall 1 \leq i \leq n\,(\forall e, e' \in V_i\,(\lambda(e) = \lambda(e')))$, and $\forall 1 \leq i < j \leq n(\forall e \in V_i e' \in V_j(\lambda(e) \neq \lambda(e')))$. The process discovery algorithm that is used on the event log L uses $\lambda(e)$ as a replacement for $\pi_{\texttt{act}}(e)$. As an example instantiation of λ, assume the activity label that we aim to split is label $a \in \Sigma$. Given the partitioning $\mathcal{V} = \{V_1, V_2, \ldots, V_n\}$, for $e \in V_1$, we let $\lambda(e) = a_1$, for $e' \in V_2$, we let $\lambda(e') = a_2$, etc.

In practice, the label-splitting algorithm typically outputs two artifacts, i.e., an event log L' and the label function λ. In event log L', the $\pi_{\texttt{act}}(e)$ values (e.g., `Activity` column in Table 1) are simply overwritten by the λ function. After process model discovery is applied on L', the $\lambda(e)$ values occurring in the discovered model are replaced by their original $\pi_{\texttt{act}}$ value. The framework can be iteratively applied, i.e., if the new activity labels used for the identified event clusters are unique.

5 Evaluation

In this section, we present the evaluation of our approach. In Sect. 5.1, we discuss the implementation of our approach, followed by the experimental setup in Sect. 5.2. The results are presented in Sect. 5.3.

5.1 Implementation

A public implementation of our framework is available[3]. The implementation supports three types of contexts based on the k preceding/succeeding activities (cf. Sect. 4.3). Based on the length-k preceding and succeeding activity sequences, we support the computation of *normalized edit distance*. Additionally, the sequences can be further abstracted using either the set abstraction (`elem`-function defined in Sect. 3) or the Parikh vector representation [20] (multi-set representation counting the occurrences of each activity in the activity sequences). The implementation uses *the Louvain community detection algorithm* [6] for community detection.

The implementation supports *variant compression*. In the compression, all events occurring in cases that describe the same sequence of activities, e.g., cases 2 and 3 in Table 1, are represented by a single unique vertex (observe that all these events have the same control-flow context). The arc weights between the vertices are equal to the sum of the arc weights in the uncompressed event graph. Additionally, self-loops are added to the vertex to represent the similarity of the equal events. Consider Fig. 4, showing a visual example of the application of variant compression. The compression equals the initial data structure used in the Louvain community detection algorithm and is, as such, particularly useful in combination with said community detection algorithm.

[3] https://github.com/jonas-tai/python-label-refinement.

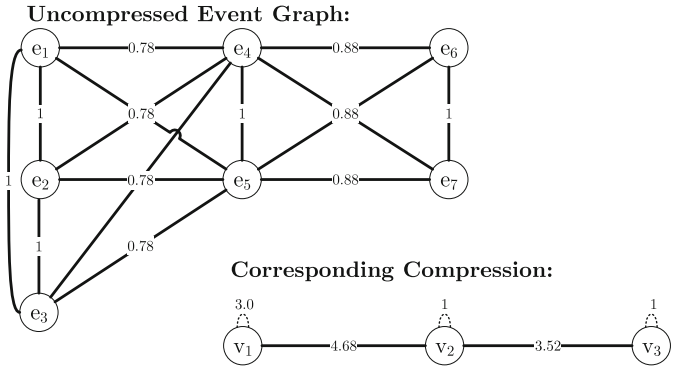


Fig. 4. Example of variant compression. Events $\{e_1, e_2, e_3\}$, $\{e_4, e_5\}$, and $\{e_6, e_7\}$ are part of the same *case variant*, respectively. Vertex v_1 represents $\{e_1, e_2, e_3\}$, vertex v_2 represents $\{e_4, e_5\}$, and vertex v_3 represents $\{e_6, e_7\}$.

5.2 Experimental Setup

In this section, we describe the experimental setup of our experiments. We conduct two sets of experiments, i.e., an experiment using several *synthetic event logs* (part of [19]) with a known ground-truth and an experiment using several *real event logs*, i.e., without any known ground-truth.

We are primarily interested in the *precision* of the discovered process model after applying label-splitting. The precision of the discovered models describes the additional amount of behavior described by the model compared to an event log. Typically, models discovered on the raw data are underfitting and have low precision. As such, we investigate the increased precision of the discovered process models due to label-splitting.

For the synthetic data, we know precisely which events belong to the same "activity cluster". Therefore, we can use *Adjusted Rand Index* (ARI) of the discovered event clustering and the ground truth clustering to measure the general quality of the detection mechanism (i.e., the ARI measures the similarity of two clusterings).

Experiments with Synthetic Data. We compare our approach with and without variant compression to the state-of-the-art approach presented in [19] (we use the same set of data as presented in [19]). In our experiments, we found that the use of *community detection* instead of *connected components* leads to better results. To show that our method outperforms the approach of [19] independent of the clustering method, we substitute the use of *connected components* by *community detection* in their implementation. In our approach, we use 11 different similarity thresholds, varying from 0.0 to 1.0, for including edges in the constructed graph, five context sizes, and three metrics to measure the similarity of events. We evaluate the approach of [19] with similar configurations of their unfolding threshold t_u, used to determine if two events with the same

Table 2. Parameter space for experiments on the synthetic event logs.

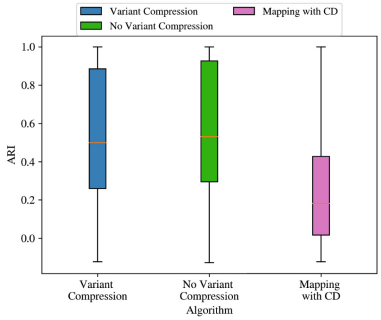
Algorithm	Parameters
Context-based with Variant Compression	Similarity threshold $t_s = \{0, 0.1, \ldots, 0.9, 1\}$ Context size $k = \{1, 2, 3, 4, 5\}$ Distance metric $dm =$ {edit distance, set distance, multi-set distance}
Case-Mapping-based with Community Detection	Unfolding threshold $t_u = \{0, 0.1, \ldots, 0.9, 1\}$ Variant threshold $t_v = \{0, 0.1, \ldots, 0.9, 1\}$

Table 3. Parameter space used for the experiments on the real life event logs.

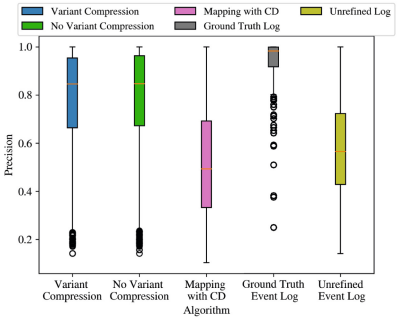
Algorithm	Parameters
Context-based with Variant Compression	Similarity threshold $t_s = \{0, 0.25, 0.5, 0.75, 1\}$ Context size $k = \{1, 3, 5\}$ Distance metric $dm =$ {edit distance, set distance, multi-set distance}
Case-Mapping-based with Community Detection	Unfolding threshold $t_u = \{0, 0.25, 0.5, 0.75, 1\}$ Variant threshold $t_v = \{0, 0.25, 0.5, 0.75, 1\}$

label in one case get different labels, i.e., if they are part of a loop or not, and variant threshold t_v, used to prune edges on the graph structure created by the algorithm to compare case mappings, parameters. A detailed list of the used parameter space for each of the algorithms is depicted in Table 2. We use the *Inductive Miner* [15] algorithm for process discovery without embedded noise filtering. As such, the algorithm guarantees that the process model describes all event data in the input (referred to as *perfect fitness*). Since a ground truth is available, we know what labels are candidates to be used for the label-splitting approach.

Experiments with Real-Life Event Data. We use publicly available event logs in combination with our proposed algorithm in experiments with real-life event data. We use 5 different event logs, i.e., the *BPI Challenge logs* from 2012 [11], 2013 (*Closed Problems log*) [22] and *2017* [12] (referred to as BPIC12, BPIC13 and BPIC17, respectively). Additionally, we use the *road fines* event log [16] and the environmental permits event log [8]. Due to the size of the event graphs, we primarily focus on the results of the variant compression, i.e., as presented in Sect. 5.1. The values for the parameters are listed in Table 3. We again use the *Inductive Miner* [15] algorithm for process discovery, in this case, with embedded noise filtering. We investigated several activity labels as



(a) The ARI scores obtained. Our approach (with and without compression) and the existing technique (Mapping with CD) [19]. The effect of compression is negligible, our approach achieves a significantly higher average ARI score.



(b) The precision scores obtained. Our approach tends to outperform the existing technique (Mapping with CD) [19] as well as applying process discovery without label-splitting.

Fig. 5. Results obtained for the experiments with synthetic data.

a candidate for splitting. We tested our approach on the three most frequently occurring activities. We selected the best-performing candidate, as we did not have a domain expert to pick the best candidate for every event log manually. For the road fines log, we selected the *Payment* activity for splitting due to insides from a manually created model [16]. This model shows that the *Payment* activity can be executed in different contexts, making it a prime candidate for label-splitting.

5.3 Results

Here, we present the results of the experiments. We further divide this section based on the results on *synthetic event data* and *real event data*, respectively.

Results on Synthetic Event Data. As indicated, in total, we used 270 different event logs from [18].

Consider Fig. 5, in which we present the corresponding results. In terms of ARI score (Fig. 5a), our approach outperforms the existing state-of-the-art label-splitting algorithm. Secondly, applying variant-based compression has a negligible effect on the overall ARI score. In Fig. 5b, the precision scores of the discovered process models is presented. The results are in line with the results obtained for the ARI score, i.e., in general, our technique outperforms the approach presented in [19]. Clearly, using the ground-truth event log generally leads to models of near-perfect precision. Yet, the median result of applying process discovery with our proposed label-splitting as a preprocessing step is well over 0.8. On average, the approach presented in [19] seems to have little effect compared to applying discovery on the unrefined event data.

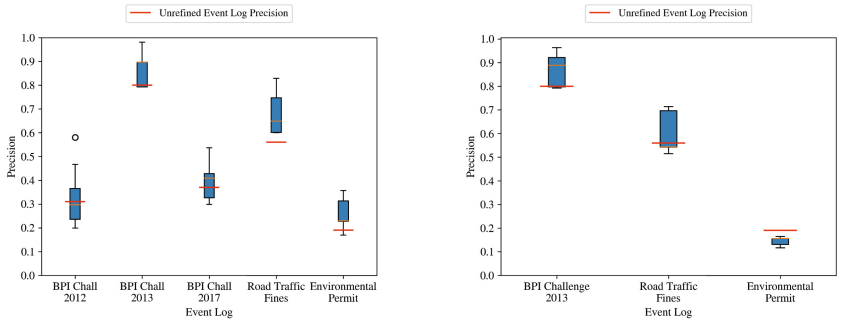
Table 4. An overview of the results of the experiments with real event logs.

			Unrefined Log		Variant Compression				
Event Log	IM Noise Threshold	Min Cases per Variant	Precision	F1-Score	Max Precision	Average Precision	Max F1-Score	Average F1-Score	Average Runtime
BPI12	0.1	3	0.31	0.48	0.58	0.33	0.73	0.48	36 s
BPI13	–	1	0.8	0.89	0.98	0.87	0.99	0.93	49 s
BPI17	0.1	3	0.37	0.53	0.54	0.4	0.68	0.56	492 s
Road Fines	0.1	1	0.56	0.72	0.83	0.68	0.89	0.79	2 s
Permit	0.1	1	0.19	0.31	0.36	0.25	0.52	0.4	0.2 s

Results on Real Event Data. Here, we present the results of applying our proposed label-splitting algorithm on real event data. The variant-compression-based version of our framework is the only algorithm that finished within a reasonable time for all event logs. The algorithm presented in [19] only finished within the time-out set for the *permits event log* (13 s vs. 0.2 s of our approach, lower precision results: 0.15 average precision vs. 0.25 of our approach), *road traffic fines event log* (2823 s seconds vs. 2 s of our approach, lower precision results: 0.6 average precision vs. 0.68 of our approach) and *BPI challenge 2013 event log* (575 s vs. 49 s of our approach, equal precision results: 0.87 average precision vs. 0.87 of our approach).

In Table 4, we present an overview of the results of our algorithm (using different parameter configurations) and compare them with the results applied to the raw event data. To reduce the number of events in the event data, for BPI12 and BPI13, we enforce a minimum of 3 cases describing the same variant to be included in the event log. We use the Inductive Mining algorithm as a discovery algorithm with a noise threshold of 0.1 (except for BPI13, where no threshold is used). The noise threshold allows for ignoring small portions of noise, i.e., generally a large share of the behavior in the event log. We observe that our algorithm increases the precision scores and, similarly, the $F1$-score. The increase in precision is most significant in the Road Fines event log, where, on average, an increase of 0.12 is measured. The compression yields a relatively small graph for some logs, and the algorithm terminates within a few seconds. Clearly, we observe higher runtime values for larger graphs (e.g., BPI17).

Finally, consider Fig. 6, in which we present box plots of the precision obtained for the different instantiations of our algorithm on the real event logs. We also present the result of [19] for the *permits event log* ("Case Mapping" in the figure). We observe that the median value of our results outperforms the results of the process discovery algorithm on the unrefined event log for all logs except for BPI12, where it is slightly lower. For the road fines log, all obtained precision values exceed the result on the raw event log. Notably, the approach presented in [19] does not always improve the quality of discovered process models, compared to using the raw event data, inline with the conclusion in [19].



(a) The precision scores obtained with our approach with variant compression.

(b) The precision scores obtained with the state-of-the-art technique from [19].

Fig. 6. Results obtained for the experiments with real life event data. The red line indicates the precision on the unrefined event log. (Color figure online)

6 Conclusion and Discussion

Label-splitting, i.e., an established preprocessing technique in process mining, generally allows for better results from process discovery algorithms. However, the state-of-the-art label-splitting algorithm performs poorly on real event data and often has infeasible runtime. Therefore, this paper presented a novel label-splitting framework based on *event similarity graphs*. Our experiments show that, compared to the state-of-the-art label-splitting algorithm, our approach yields process models with higher precision and has better runtime performance.

One interesting finding of our evaluation is that the best parameter configuration for our algorithms highly depends on the input event log. One solution is to run our algorithm over a range of parameter configurations. By using various optimizations in the implementation, e.g., only recalculating the graph edges after the first iteration, we found this to be a feasible solution in our evaluation of real event data.

Future Work. Using variant-compression allows us to obtain a feasible runtime for the algorithm. We plan to perform experiments with different community detection algorithms and investigate whether the compression can be adopted. Secondly, we aim to investigate *detection mechanisms* for label-splitting, i.e., indicating what activity labels may be good candidates for splitting.

References

1. van der Aalst, W.M.P.: Process Mining - Data Science in Action, 2nd edn. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-49851-4
2. van der Aalst, W.M.P., de Medeiros, A.K.A., Weijters, A.J.M.M.: Genetic process mining. In: Ciardo, G., Darondeau, P. (eds.) ICATPN 2005. LNCS, vol. 3536, pp. 48–69. Springer, Heidelberg (2005). https://doi.org/10.1007/11494744_5

3. van der Aalst, W.M.P., Rubin, V.A., Verbeek, H.M.W., van Dongen, B.F., Kindler, E., Günther, C.W.: Process mining: a two-step approach to balance between underfitting and overfitting. Softw. Syst. Model. **9**(1), 87–111 (2010)
4. van der Aalst, W.M.P., Weijters, T., Maruster, L.: Workflow mining: discovering process models from event logs. IEEE Trans. Knowl. Data Eng. **16**(9), 1128–1142 (2004)
5. Augusto, A., et al.: Automated discovery of process models from event logs: review and benchmark. IEEE Trans. Knowl. Data Eng. **31**(4), 686–705 (2019)
6. Blondel, V.D., Guillaume, J.L., Lambiotte, R., Lefebvre, E.: Fast unfolding of communities in large networks. J. Stat. Mech. Theory Exp. **2008**(10), P10008 (2008)
7. vanden Broucke, S.K.L.M., Weerdt, J.D.: Fodina: a robust and flexible heuristic process discovery technique. Decis. Support Syst. **100**, 109–118 (2017)
8. Buijs, J.: Receipt phase of an environmental permit application process (WABO), CoSeLoG project, March 2022. https://doi.org/10.4121/12709127.v2
9. Buijs, J.C.A.M., van Dongen, B.F., van der Aalst, W.M.P.: Quality dimensions in process discovery: the importance of fitness, precision, generalization and simplicity. Int. J. Cooperative Inf. Syst. **23**(1), 1440001 (2014)
10. Carmona, J., Cortadella, J., Kishinevsky, M.: A region-based algorithm for discovering Petri nets from event logs. In: Dumas, M., Reichert, M., Shan, M.-C. (eds.) BPM 2008. LNCS, vol. 5240, pp. 358–373. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-85758-7_26
11. van Dongen, B.: BPI Challenge 2012, April 2012. https://doi.org/10.4121/uuid:3926db30-f712-4394-aebc-75976070e91f
12. van Dongen, B.: BPI Challenge 2017, February 2017. https://doi.org/10.4121/uuid:5f3067df-f10b-45da-b98b-86ae4c7a310b
13. Fortunato, S.: Community detection in graphs. Phys. Rep. **486**(3), 75–174 (2010)
14. Koschmider, A., Kaczmarek, K., Krause, M., van Zelst, S.J.: Demystifying noise and outliers in event logs: review and future directions. In: Marrella, A., Weber, B. (eds.) BPM 2021. LNBIP, vol. 436, pp. 123–135. Springer, Cham (2022). https://doi.org/10.1007/978-3-030-94343-1_10
15. Leemans, S.J.J., Fahland, D., van der Aalst, W.M.P.: Discovering block-structured process models from event logs - a constructive approach. In: Colom, J.-M., Desel, J. (eds.) PETRI NETS 2013. LNCS, vol. 7927, pp. 311–329. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-38697-8_17
16. de Leoni, M.M., Mannhardt, F.: Road Traffic Fine Management Process, February 2015. https://doi.org/10.4121/uuid:270fd440-1057-4fb9-89a9-b699b47990f5
17. Li, J., Liu, D., Yang, B.: Process mining: extending α-algorithm to mine duplicate tasks in process logs. In: Chang, K.C.-C., et al. (eds.) APWeb/WAIM -2007. LNCS, vol. 4537, pp. 396–407. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-72909-9_43
18. Lu, X., Fahland, D., van den Biggelaar, F.J.H.M., van der Aalst, W.M.P.: Detecting deviating behaviors without models. In: Reichert, M., Reijers, H.A. (eds.) BPM 2015. LNBIP, vol. 256, pp. 126–139. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-42887-1_11
19. Lu, X., Fahland, D., van den Biggelaar, F.J.H.M., van der Aalst, W.M.P.: Handling duplicated tasks in process discovery by refining event labels. In: La Rosa, M., Loos, P., Pastor, O. (eds.) BPM 2016. LNCS, vol. 9850, pp. 90–107. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-45348-4_6
20. Parikh, R.: On context-free languages. J. ACM **13**(4), 570–581 (1966)

21. Schaeffer, S.E.: Graph clustering. Comput. Sci. Rev. **1**(1), 27–64 (2007)
22. Steeman, W.: BPI Challenge 2013, closed problems, April 2013. https://doi.org/10.4121/uuid:c2c3b154-ab26-4b31-a0e8-8f2350ddac11
23. Tax, N., Alasgarov, E., Sidorova, N., Haakma, R., van der Aalst, W.M.P.: Generating time-based label refinements to discover more precise process models. J. Ambient Intell. Smart Environ. **11**(2), 165–182 (2019)
24. Vázquez-Barreiros, B., Mucientes, M., Lama, M.: Enhancing discovered processes with duplicate tasks. Inf. Sci. **373**, 369–387 (2016)
25. Weijters, A., van der Aalst, W.M.P., De Medeiros, A.: Process mining with the heuristicsminer algorithm. BETA publicatie: working papers, Technische Universiteit Eindhoven (2006)
26. Yang, S., et al.: Medical workflow modeling using alignment-guided state-splitting HMM. In: 2017 IEEE International Conference on Healthcare Informatics, ICHI 2017, Park City, UT, USA, 23–26 August 2017, pp. 144–153. IEEE Computer Society (2017)
27. van Zelst, S.J., Mannhardt, F., de Leoni, M., Koschmider, A.: Event abstraction in process mining: literature review and taxonomy. Granul. Comput. **6**(3), 719–736 (2021)