A Framework for Automated Abstraction Class Detection for Event Abstraction

Chiao-Yun Li[1,2(✉)], Sebastiaan J. van Zelst[1,2], and Wil M.P. van der Aalst[2]

[1] Fraunhofer FIT, Birlinghoven Castle, Sankt Augustin, Germany
{chiao-yun.li,sebastiaan.van.zelst}@fit.fraunhofer.de
[2] RWTH Aachen University, Aachen, Germany
wvdaalst@pads.rwth-aachen.de

Abstract. *Process mining* enables companies to extract insights into the process execution from *event data*. Event data that are stored in information systems are often too fine-grained. When process mining techniques are applied to such system-level event data, the outcomes are often overly complex for human analysts to interpret. To address this issue, numerous (semi-)automated *event abstraction* techniques that "lift" event data to a higher level have been proposed. However, most of these techniques are *supervised*, i.e., the knowledge of which low-level event data or activities need to be abstracted into a high-level instance or concept is explicitly given. In this paper, we propose a fully *unsupervised* event abstraction framework for partially ordered event data, which further enables arbitrary levels of abstraction. The evaluation shows that the proposed framework is scalable and allows for discovering a more precise process model.

Keywords: Process mining · Event abstraction · Partial orders

1 Introduction

Nowadays organizations execute their processes with the support of information systems. Historical records of the process execution are stored as *event data* in the systems. *Process mining* [1] provides techniques to extract knowledge of such data to enhance the performance and the compliance of said processes. For example, *process discovery techniques* detect the relations of *activities* in a process, i.e., well-defined process steps, and represent the identified behavior as a process model [3,6].

Most process mining techniques are directly applied on event data as recorded in information systems. Figure 1 shows how a human analyst translates the system-level event data into activities performed at the business-level. Such system-level event data often present an overly fine granularity that is too detailed to understand a process at the business-level. In case of a large or flexible process, the complexity further causes challenges in interpreting process mining results. Therefore, the *principle of abstraction* is often applied to lift event data to a higher level, i.e., referred to as *event abstraction*, which aggregates activities or low-level event data to which process mining techniques are applied.

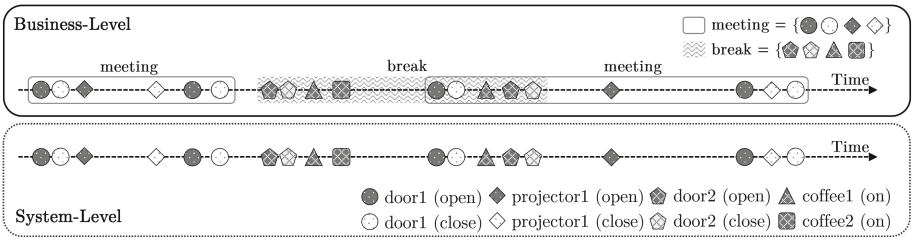


Fig. 1. A motivating example of interpreting sensor data, i.e., system-level event data, to business-level executions. The same shapes of event data at the system-level reflect the executions performed as time intervals, e.g., door 1 is opened from door1 (open) until door1 (close). The activities with the same pattern define a concept at the business-level.

Existing event abstraction techniques assume that event data are recorded and ordered chronologically. Meanwhile, most of the techniques operate in a semi- or fully supervised manner. In practice, both assumptions are often not applicable. Activities are often executed in time intervals and, thus, partially ordered. For example, an office worker can drink coffee *during* a meeting *after* a break. Furthermore, the application of event abstraction in real-life varies in the abstraction level required, which often depends on scenarios and stakeholders. One can apply the abstraction iteratively to construct a hierarchy of abstractions; however, it requires a technique to be able to deal with interleaving intervals as shown in the business-level executions in Fig. 1, where the break overlaps with the meeting in time. Moreover, a (semi-)supervised approach requires domain knowledge of event data to abstract; yet, such information is often not present and it takes a tremendous amount of time and effort to label event data. If a different level of abstraction is required, further domain knowledge must be provided and the event data need to be relabeled.

In this paper, we propose a framework that tackles the two aforementioned issues, i.e., a *fully unsupervised framework* for *partially ordered event data*. Meanwhile, the support for partial orders inherently yields iterative applicability to construct a hierarchy of abstractions for various stakeholders. The framework detects concepts at the higher level, namely *abstraction classes*, which are defined by activities based on their observed execution context. The paper is compatible with the generic partial-order-based abstraction framework proposed in [9] and we further extend and define such context. The framework is evaluated using event data based on two real-life processes as in [9]. Additionally, we compare the proposed framework with two other event abstraction techniques. The experiments show that the proposed framework is the most applicable in practice for its scalability, robustness against infrequent activities, and effectiveness of abstraction, which is quantitatively and qualitatively evaluated and reasoned based on the process models discovered.

The rest of the paper is structured as follows. In Sect. 2, we discuss existing abstraction techniques in the field of process mining. We introduce the framework in Sect. 3, which is evaluated in Sect. 4. Finally, Sect. 5 summarizes the framework and the evaluation and concludes the paper with future work.

2 Related Work

This section reviews the applicability of existing event abstraction techniques from two perspectives: the level of domain knowledge required and whether a technique supports partially ordered event data.

Most existing event abstraction techniques require explicit domain knowledge as input. In general, the techniques expect either the behavioral knowledge of the activities in a process [11,15] or the mapping of activities to high-level concepts [5,7,14]. In [11], the authors explore all the possible behavior between activities, so-called *local process models* (LPMs), and extract event data at a higher level based on the ones chosen by domain experts. In [15], the authors search for instances of predefined patterns with a focus on the education field. Leemans et al. discover hierarchical process models based on the assumption that a hierarchical mapping of low-level instances to high-level concepts is annotated in event data [7]. In [5] and [14], statistical models are trained using labeled event data to predict concepts at a higher level of abstraction.

Unsupervised approach groups low-level concepts or instances by identifying their relationship in event data and allows for exploration through parameter tuning. Alharbi et al. discover statistical models to learn activities at a higher level [2]. Nguyen et al. decompose all activities in a process into sets of activities by maximizing the modularity measure [12]. In [8], the authors apply classical clustering techniques by extracting features using frequency- or duration-based encoding in a session, i.e., a segment of a sequence of event data.

Unsurprisingly, the more domain knowledge is provided, the more precise the abstraction is; however, this limits the practical applicability of a technique. Meanwhile, since activities are often performed as time intervals in practice, it is crucial that a technique supports partially ordered event data. Although partially ordered event data may be transformed into sequences of executions by introducing life cycle information, applying existing abstraction techniques may result in unreasonable outcomes. For example, the *start* of drink coffee is considered to be part of a meeting and the *complete* of drink coffee is an activity performed during a break. The method proposed in this paper is the first unsupervised technique focusing on abstracting partially ordered event data.

3 Framework

In this section, we present the proposed framework to detect concepts at a higher level, namely *abstraction classes*, from event data. Figure 2 presents an overview of the framework. First, event data is provided in the format of an *event log* L. Next, we extract a *preceding* and a *succeeding context* as categorical distributions

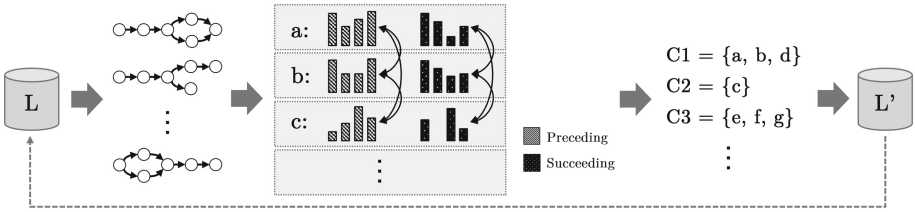


Fig. 2. Schematic overview of the framework. The abstraction classes are defined by activities with similar *preceding* and *succeeding* contexts. An event log at the higher level is then extracted and can be applied as input for the next iteration.

for every activity. The similarity of the activities is calculated accordingly and the activities that are *similar enough* define an *abstraction class*. Finally, an event log L' containing only the identified abstraction classes is extracted and the framework can be iteratively applied to the output to construct a hierarchy of abstraction.

First, we introduce the mathematical notations applied and define event data that is more generally applicable in practice. Then, we show how a context is constructed and the identification of abstraction classes.

Notation. Given an arbitrary set X, $\mathcal{P}(X)=\{X'|X'\subseteq X\}$ denotes the powerset of X. A multiset is a generalization of a set where an element in X may appear multiple times. For example, $[a,b,b]=[a,b^2]$ is a multiset. $\mathcal{M}(X)$ denotes all the possible multisets over X. A sequence σ of length n over X is a function $\sigma\colon\{1,\ldots,n\}\rightarrow X$, denoted as $\sigma=\langle x_1,\ldots,x_n\rangle$, where $1\leq i\leq n, x_i=\sigma(i)$. We write $|\sigma|$ to denote the length of a sequence.

3.1 Event Data

A *case* is a process instance defined by a set of *activity instances*, i.e., records of the execution of *activities*. The collection of cases defines an *event log*. We define an activity instance, a case, and an event log as follows.

Definition 1 (Activity Instance). *An activity instance records the execution of an activity and is described by a set of attributes. $\mathcal{A}$ is the universe of activity instances, $\mathcal{N}$ is the universe of attribute names, and $\mathcal{U}_n$ denotes all the possible values of $n\in\mathcal{N}$. $\mathcal{U}_\mathcal{N}=\cup_{n\in\mathcal{N}}\mathcal{U}_n$ denotes the universe of the values of all the attributes. Given $n\in\mathcal{N}$ and $\mathbf{a}\in\mathcal{A}$, we define projection function $\pi_n\colon\mathcal{A}\nrightarrow\mathcal{U}_n$, where $\pi_n(\mathbf{a})\in\mathcal{U}_n$ if $\mathbf{a}$ has a value for n, else $\pi_n(\mathbf{a})=\perp$. $\mathcal{U}_{act}$ denotes the universe of activities. Given $\mathbf{a}\in\mathcal{A}$, the following attributes are assumed to be always defined: $\pi_{act}(\mathbf{a})\in\mathcal{U}_{act}$ for the activity of $\mathbf{a}$; $\pi_{st}(\mathbf{a})\in\mathbb{R}^+$ for the start time of $\mathbf{a}$; $\pi_{ct}(\mathbf{a})\in\mathbb{R}^+$ for the complete time of $\mathbf{a}$ where $\pi_{st}(\mathbf{a})\leq\pi_{ct}(\mathbf{a})$.*

Definition 2 (Case, Event Log). *A case is the collection of activity instances executed in the context of a process instance. Let $c\subseteq\mathcal{A}$ be a case. Given $\mathbf{a},\mathbf{a}'\in c$, we*

Table 1. An example event log L_0. An activity instance is represented in a row and is described by a set of attributes and the case it belongs to.

ID		Activity (Abbreviation)	Timestamp			Resource
Case	Activity Instance		Date	Start	Complete	
1	1	receive order (ro)	Sep. 6	14:32:21	14:32:21	Peter
1	2	pay in cash (pi)	Sep. 6	14:34:02	14:34:02	Mike
1	3	confirm payment (cp)	Sep. 6	14:35:40	14:36:14	Peter
1	4	bake pizza (bp)	Sep. 6	14:46:21	15:03:56	Sara
1	5	make salad (ms)	Sep. 6	14:49:32	15:00:11	Sara
1	6	pack & deliver (pd)	Sep. 6	15:15:16	15:27:48	Ethan
2	7	receive order (ro)	Sep. 7	15:08:00	15:08:00	Peter
2	8	pay by credit card (pb)	Sep. 7	15:09:27	15:09:27	Ben
2	9	confirm payment (cp)	Sep. 7	15:10:29	15:10:41	Ethan
2	10	bake pizza (bp)	Sep. 7	15:30:38	15:49:40	Sara
2	11	make salad (ms)	Sep. 7	15:47:56	15:57:10	Alex
2	12	pack & deliver (pd)	Sep. 7	15:55:02	16:17:59	Ethan

write $\mathbf{a} \prec \mathbf{a}'$ *if and only if* $\pi_{ct}(\mathbf{a}) < \pi_{st}(\mathbf{a}')$. *The transitive reduction over* c, *written as* $\rho = (c, \lessdot)$, *is the minimum relation* $\lessdot$ *on* c *such that* $\forall \mathbf{a}_1, \mathbf{a}_2 \in c, \mathbf{a}_1 \lessdot \mathbf{a}_2$ *iff* $\mathbf{a}_1 \prec \mathbf{a}_2$ *and* $\nexists \mathbf{a}' \in c (\mathbf{a}_1 \prec \mathbf{a}' \wedge \mathbf{a}' \prec \mathbf{a}_2)$. *Given* $\mathbf{a}_1, \mathbf{a}_n \in c$, *a path from* $\mathbf{a}_1$ *to* $\mathbf{a}_n$ *in* $\rho = (c, \lessdot)$ *is* $\sigma = \langle (\mathbf{a}_1, \mathbf{a}_2), \ldots, (\mathbf{a}_{n-1}, \mathbf{a}_n) \rangle$ *s.t.* $\forall 1 \leq i < n (\mathbf{a}_i \lessdot \mathbf{a}_{i+1})$. $\mathbf{p}_\rho(\mathbf{a}_1, \mathbf{a}_n)$ *denotes all the paths between* $\mathbf{a}_1$ *and* $\mathbf{a}_n$ *in* ρ. $\mathcal{C} \subseteq \mathcal{P}(\mathcal{A})$ *denotes the universe of cases. An event log* $L \subseteq \mathcal{C}$ *is a collection of cases where* $\forall c, c' \in L (c \neq c') \implies c \cap c' = \emptyset$.

As an activity instance is described by a time interval, a case is considered as a strict partial order of activity instances which is *irreflexive*, *asymmetric*, and *transitive*. Figure 3 visualizes an event log L_0 shown in Table 1.

Definition 3 (Abstraction Class). *Let* $\mathcal{U}_{act}$ *be the universe of activities. An abstraction class* $C \in \mathcal{P}(\mathcal{U}_{act})$ *is a non-empty set of activities, i.e.,* $C \neq \emptyset$.

Note that Definition 3 does not limit an activity to be included in exact one abstraction class and vice versa. In the proposed framework, we assume that a *partition* of activities defines an abstraction class.

3.2 Constructing Preceding and Succeeding Patterns

A *preceding* and a *succeeding pattern* form a context of an activity instance. First, we select activity instances before, namely *preceding instances*, and after,

Fig. 3. Visualization of $L_0 = \{c_1, c_2\}$ in Table 1, where every circle represents an activity instance, which is labeled with the corresponding activity (abbreviation) and the identifier, and the arcs present the relation $\rho_1 = (c_1, \quad)$ and $\rho_2 = (c_2, \quad)$.

namely *succeeding instances*, every activity instance in a case. Next, we construct a *preceding* and a *succeeding pattern* from the instances selected. We define preceding and succeeding instances of an activity instance as follows.

Definition 4 (Preceding/Succeeding Instances). *Let $c \in \mathcal{C}$ be a case, where $\mathcal{C}$ is the universe of cases. Given $\mathbf{a} \in c$ and $\mathbf{A} \subset c$, we overload the notation $\prec$ such that $\mathbf{A} \prec \mathbf{a} \Leftrightarrow \forall \mathbf{a}' \in \mathbf{A}(\mathbf{a}' \prec \mathbf{a})$. We call $\mathbf{A}$ preceding activity instances of $\mathbf{a}$ and say that $\mathbf{A}$ precedes $\mathbf{a}$. Succeeding activity instances $\mathbf{A}' \subset c$ of $\mathbf{a} \in c$ is defined symmetrically, i.e., $\mathbf{a} \prec \mathbf{A}' \Leftrightarrow \forall \mathbf{a}' \in \mathbf{A}'(\mathbf{a} \prec \mathbf{a}')$.*

We propose two instantiations to *select* preceding and succeeding instances of an activity instance within a window of size $w \in \mathbb{R}^+$, namely the *distance-based* and the *time-based* selectors.

Definition 5 (Distance-Based Selector). *Let $c \in \mathcal{C}$ be a case, where $\mathcal{C}$ is the universe of cases s.t. $\rho = (c, \ll)$. $\overleftarrow{\phi}^{d}_{\rho} \colon \mathcal{A} \times \mathbb{N}^+ \to \mathcal{P}(\mathcal{A})$ defines the distance-based preceding selector, where $\overleftarrow{\phi}^{d}_{\rho}(\mathbf{a}, w) = \{\mathbf{a}' | \exists \sigma \in \mathbf{p}_\rho(\mathbf{a}', \mathbf{a}) \, (|\sigma| \leq w)\}$ denotes a set of preceding instances of $\mathbf{a}$. The succeeding selector $\overrightarrow{\phi}^{d}_{\rho}$ is defined symmetrically.*

Definition 6 (Time-Based Selector). *Let $c \in \mathcal{C}$ be a case, where $\mathcal{C}$ is the universe of cases s.t. $\rho = (c, \ll)$. $\overleftarrow{\phi}^{t}_{\rho} \colon \mathcal{A} \times \mathbb{R}^+ \to \mathcal{P}(\mathcal{A})$ defines the time-based preceding selector, where $\overleftarrow{\phi}^{t}_{\rho}(\mathbf{a}, w) = \{\mathbf{a}' | \mathbf{a}' \prec \mathbf{a} \wedge \pi_{st}(\mathbf{a}') \geq \pi_{st}(\mathbf{a}) - w\}$ denotes a set of preceding instances of $\mathbf{a}$. The succeeding selector $\overrightarrow{\phi}^{t}_{\rho}$ is defined symmetrically.*

Given a set of activity instances in a case, we construct patterns as below.

Definition 7 (Preceding/Succeeding Patterns). *Let $c \in \mathcal{C}$ be a case, where $\mathcal{C}$ is the universe of cases s.t. $\rho = (c, \ll)$. A pattern is a data structure defined by a set of activity instances $\mathbf{A} \subseteq c$ and an attribute $n \in \mathcal{N}$, where $\mathcal{N}$ is the universe of attribute names. Given $\mathbf{A} \subseteq c$ and $n \in \mathcal{N}$, we define the instantiation functions:*

- $\lambda^{\mathbf{s}}_{\rho} \colon \mathcal{P}(\mathcal{A}) \times \mathcal{N} \to \mathcal{P}(\mathcal{U}_{\mathcal{N}})$, *where* $\lambda^{\mathbf{s}}_{\rho}(\mathbf{A}, n) = \{\pi_n(\mathbf{a}) | \mathbf{a} \in \mathbf{A}\}$ *is a pattern of type set;*
- $\lambda^{\mathbf{m}}_{\rho} \colon \mathcal{P}(\mathcal{A}) \times \mathcal{N} \to \mathcal{M}(\mathcal{U}_{\mathcal{N}})$, *where* $\lambda^{\mathbf{m}}_{\rho}(\mathbf{A}, n) = [\pi_n(\mathbf{a}) | \mathbf{a} \in \mathbf{A}]$ *is a pattern of type multiset;*
- $\lambda^{\mathbf{g}}_{\rho} \colon \mathcal{P}(\mathcal{A}) \times \mathcal{N} \to \mathcal{P}(\mathcal{A} \times \mathcal{U}_{\mathcal{N}})$, *where* $\lambda^{\mathbf{g}}_{\rho}(\mathbf{A}, n) = \{(\mathbf{a}, \pi_n(\mathbf{a})) | \mathbf{a} \in \mathbf{A}\}$. *Given* $\mathbf{a}_1, \mathbf{a}_2 \in \mathbf{A}$ *and* $v_1 = (\mathbf{a}_1, \pi_n(\mathbf{a}_1)), v_2 = (\mathbf{a}_2, \pi_n(\mathbf{a}_2)) \in \lambda^{\mathbf{g}}_{\rho}(\mathbf{A}, n)$, *we overload* $\lll$ *s.t.* $v_1 \lll v_2 \Leftrightarrow \mathbf{a}_1 \lll \mathbf{a}_2$. *We call such pattern, together with the relation, as type graph.*

We call the attribute to construct a pattern a pattern attribute. We name a pattern initiated from preceding instances a preceding pattern and, likewise, a pattern initiated from succeeding instances a succeeding pattern.

In this section, we showed how to construct a *context*, i.e., a preceding and a succeeding pattern, of an activity instance. We proposed two instantiations to select the surrounding instances and three instantiations to construct a pattern from a set of activity instances.

3.3 Computing Activity Similarity

We evaluate the similarity of two activities by comparing how similar their contexts are, which are defined as the distributions of preceding and succeeding patterns of an activity in an event log.

Preceding/Succeeding Context. A preceding context of an activity is a collection of preceding patterns of the activity instances of an activity in an event log. We transform such a collection into a categorical distribution of patterns where every pattern is weighted with its frequency. A succeeding context is constructed likewise. A context of an activity is defined together by a preceding and a succeeding context of an activity.

Activity Similarity. The similarity between two activities is quantified based on their contexts. We initiate the framework using *normalized earth mover's distance* (*EMD*) to compute the similarity between two distributions [13]. The metric requires the distance between two patterns. Suppose a context of an activity is constructed using patterns of type set or multiset, we apply *Jaccard distance* for the distance between two patterns. For patterns of type graph, we apply *graph edit distance* based on the pattern attribute. The preceding and the succeeding similarity are the similarities computed based on preceding and succeeding contexts of two activities, correspondingly. The similarity between two activities is the average of the preceding and the succeeding similarity.

Identifying Abstraction Classes. The activities that are *similar enough* define an abstraction class. The similar activities are the activities whose similarity is above a given *similarity threshold*. Various grouping strategies can be applied. Figure 4 visualizes the abstraction process initiated with *binary grouping* strategy for L_0, i.e., only a pair of the most similar subset of activities in an abstraction class is identified, e.g., the abstraction class C is abstracted by first grouping pb and pi as $\{\mathsf{pb}, \mathsf{pi}\} \subseteq C$ and merging cp into $\{\mathsf{pb}, \mathsf{pi}\}$ such that $C = \{\mathsf{pi}, \mathsf{pb}, \mathsf{cp}\}$. In each iteration of grouping, we update the preceding and the succeeding contexts by summing up the weight of every pattern. The similarity between a set of activities and another activity or set of activities can be calculated using EMD likewise. We repeat the procedure until no activities can be grouped and the final sets of activities define the abstraction classes.

The framework is generally applicable to partially ordered event data. By applying the framework iteratively to the output of the previous iteration, one can construct a hierarchy of abstractions, which allows various stakeholders to flexibly adjust the level of abstraction required.

Table 2. Quality metrics of flatten process models using the proposed framework, session-based, and LPM-based techniques. The metrics of the models without abstraction (represented with -) are provided as baselines.

Event Log	Sepsis [10]				BPIC12 [4]			
Technique	–	Proposed	Session	LPM	–	Proposed	Session	LPM
Fitness	**0.9979**	0.8751	0.9538	0.8852	**0.9981**	0.9962	0.9556	N/A
Precision	0.2140	0.2877	0.2657	**0.5090**	0.1297	**0.1685**	0.0939	N/A
F1-Score	0.3524	0.4330	0.4156	**0.6463**	0.2296	**0.2883**	0.1710	N/A
Simplicity	0.5584	0.5833	0.5804	**0.6634**	0.6175	**0.6531**	0.5245	N/A

*The implementation of the LPM-based technique cannot discover an abstract model within a reasonable amount of time with BPIC12. Hence, the metrics of the flatten model are unavailable.

4 Evaluation

We compare the proposed framework with an unsupervised [8] and a semi-supervised [11] technique on two real-life event logs Sepsis [10] and BPIC12 [4].[1] We evaluate the quality of abstraction using process models.[2] Apparently, the framework provides an easier means for human analysts by grouping activities into disjoint sets. Hence, to fairly compare the effectiveness of different abstraction techniques, we "flatten" an *abstract model* discovered from an abstracted event log to the original level of granularity. The flattening is achieved by replacing every transition labeled with an abstraction class in an abstract model with a model representing the behavior of the activities defining the corresponding abstraction class. We name such models as *flatten models*. Meanwhile, we discover and provide models without applying abstraction as baselines.

Table 2 shows the quality metrics, i.e., fitness, precision, F1-score, the harmonic mean of the fitness and the precision, and simplicity, of the flatten models.

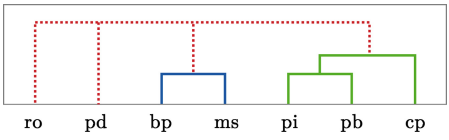


Fig. 4. Abstraction process using binary merging strategy based on L_0. The activities below the dashed lines define abstraction classes.

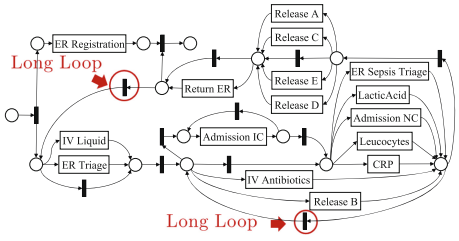
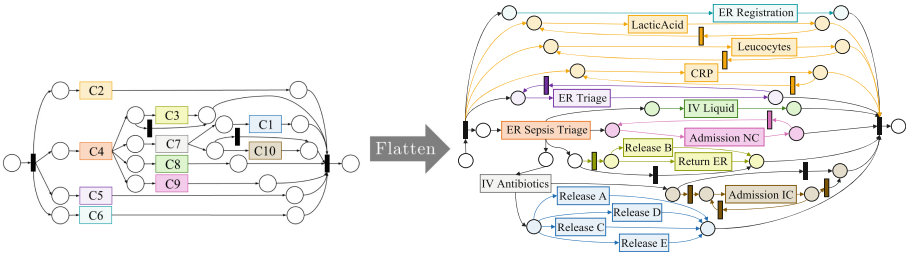


Fig. 5. Model without abstraction.

[1] We apply DBSCAN with the default setting in the implementation of [8]. For [11], we select the LPM with the highest score in every group based on the maximal grouping strategy.

[2] We apply Inductive Miner - infrequent & life cycle with noise threshold 0.2 to discover process models represented using Petri nets [1].



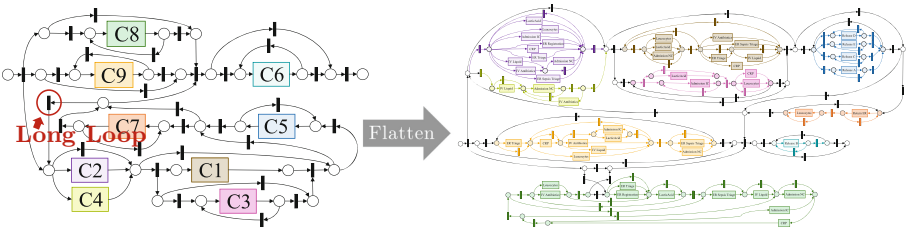
(a) Abstract model. (b) Flatten model.

Fig. 6. Models discovered using the proposed framework.

The models without abstraction guarantee the highest fitness, which, however, compromise the precision. LPM-based technique outperforms other techniques in terms of other metrics using Sepsis; nevertheless, it is not scalable to BPIC12. In contrast, the proposed framework strikes a balance between the scalability and the quality of abstraction based on the metrics.

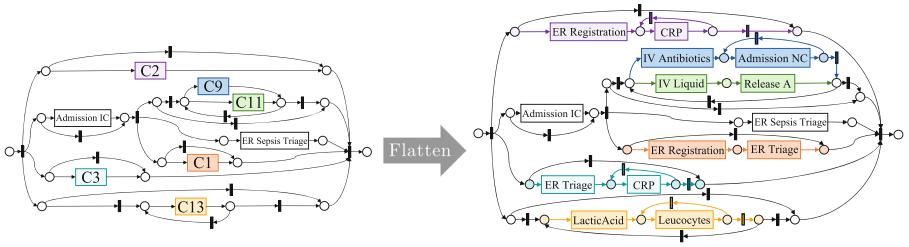
To qualitatively compare and reason the outcomes of different abstraction techniques, Figs. 5 to 8 demonstrate the process models discovered based on Sepsis (as the outcomes are available for all the techniques compared). We color an abstraction class in an abstract model and the behavior of the activities defining the abstraction class in the corresponding flatten model the same. Figure 5 presents the model without abstraction. With skipping of activities and long loops (more than one activity being repetitively executed), the model provides little to no insights with a high degree of "flower-like" behavior, i.e., the activities can be executed an unlimited number of times in any order. We highlighted some constructs that can cause such long loops in Fig. 5. The same flower-like behavior is also identified in the models using the session-based technique in Fig. 7. In some scenarios, the abstract model in Fig. 7a even allows for none of the abstraction classes and, thereby, the activities defining the abstraction classes, being executed. Moreover, the flatten model using the session-based technique in Fig. 7b exhibits a rather complex behavior, which contradicts the motive of applying abstraction. In contrast, we do not observe long loops in the models using the proposed framework while the flatten model contains the same number of activities as in the event log (see Fig. 6). Hence, we show the effectiveness of the proposed framework based on the simplicity and the precision of the flatten models, which is also reflected in Table 2.

Figure 8 displays the abstract and flatten models using the LPM-based technique. The quantitative evaluation in Table 2 suggests a better outcome. However, despite the same noise threshold of the discovery algorithm applied, some activities are not identified as shown in the flatten model (see Fig. 8b), revealing another limitation of the technique that it is more sensitive to frequent behavior. The proposed approach (see Fig. 6), instead, is able to identify abstraction



(a) Abstract model. (b) Flatten model.

Fig. 7. Models discovered using the session-based technique.



(a) Abstract model. (b) Flatten model.

Fig. 8. Models discovered using the LPM-based technique.

classes defined by less frequent activities. Hence, we argue that the proposed framework is more robust compared to the LPM-based technique.

We evaluated and compared the proposed framework with other abstraction techniques based on two real-life event logs. The quantitative evaluation showed that applying abstraction not only provides an easier means for humans to analyze a process but also allows for discovering a more precise model. By further reasoning the outcomes based on the models, we argue that the proposed framework is the most applicable in practice as it provides more insights with less "flower-like" behavior while being scalable and robust to less frequent activities.

5 Conclusion

We proposed a framework to identify concepts at the higher level, i.e., *abstraction classes*, for event abstraction based on partially ordered event data. The framework computes the similarity of activities based on their surrounding contexts and discovers abstraction classes by grouping the activities whose similarity is higher than a threshold. We evaluated the framework using two real-life event logs. The experiments showed that the proposed framework outperforms the other techniques for its robustness, scalability, and effectiveness of abstraction. For future work, different instantiations will be compared and investigated.

References

1. van der Aalst, W.M.P.: Process mining - data science in action (2016)
2. Alharbi, A., Bulpitt, A., Johnson, O.A.: Towards unsupervised detection of process models in healthcare. Stud. Health Technol. Inform. **247**, 381–385 (2018)
3. Augusto, A., Conforti, R., Dumas, M., La Rosa, M., Polyvyanyy, A.: Split miner: automated discovery of accurate and simple business process models from event logs. Knowl. Inf. Syst. **59**(2), 251–284 (2019)
4. van Dongen, B.: BPI challenge 2012 (2012)
5. Fazzinga, B., Flesca, S., Furfaro, F., Pontieri, L.: Process discovery from low-level event logs. In: Krogstie, J., Reijers, H.A. (eds.) CAiSE 2018. LNCS, vol. 10816, pp. 257–273. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-91563-0_16
6. Leemans, S.J.J., Fahland, D., van der Aalst, W.M.P.: Using life cycle information in process discovery. In: Reichert, M., Reijers, H.A. (eds.) BPM 2015. LNBIP, vol. 256, pp. 204–217. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-42887-1_17
7. Leemans, S.J.J., Goel, K., van Zelst, S.J.: Using multi-level information in hierarchical process mining: balancing behavioural quality and model complexity. In: International Conference on Process Mining, pp. 137–144. IEEE (2020)
8. de Leoni, M., Dündar, S.: Event-log abstraction using batch session identification and clustering. In: Proceedings of the 35th Annual ACM Symposium on Applied Computing, pp. 36–44 (2020)
9. Li, C., van Zelst, S.J., van der Aalst, W.M.P.: An activity instance based hierarchical framework for event abstraction. In: International Conference on Process Mining, pp. 160–167. IEEE (2021)
10. Mannhardt, F.: Sepsis cases - event log (2016)
11. Mannhardt, F., de Leoni, M., Reijers, H.A., van der Aalst, W.M.P., Toussaint, P.J.: Guided process discovery-a pattern-based approach. Inf. Syst. **76**, 1–18 (2018)
12. Nguyen, H., Dumas, M., ter Hofstede, A.H.M., La Rosa, M., Maggi, F.M.: Stage-based discovery of business process models from event logs. Inf. Syst. **84**, 214–237 (2019)
13. Rubner, Y., Tomasi, C., Guibas, L.J.: The earth mover's distance as a metric for image retrieval. Int. J. Comput. Vision **40**(2), 99–121 (2000)
14. Tax, N., Sidorova, N., Haakma, R., van der Aalst, W.M.P.: Event abstraction for process mining using supervised learning techniques. In: Bi, Y., Kapoor, S., Bhatia, R. (eds.) IntelliSys 2016. LNNS, vol. 15, pp. 251–269. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-56994-9_18
15. Trcka, N., Pechenizkiy, M.: From local patterns to global models: towards domain driven educational process mining. In: Ninth International Conference on Intelligent Systems Design and Applications, pp. 1114–1119. IEEE (2009)