Parallelism-based Session Creation to Identify High-Level Activities in Event Log Abstraction

Onur Dogan[1,2] and Massimiliano de Leoni[1]

[1] University of Padua, Department of Mathematics, 35121, Padua, Italy
onur.dogan@unipd.it, deleoni@math.unipd.it
[2] Izmir Bakircay University, Department of Management Information Systems, 35665, Izmir, Turkiye
onur.dogan@bakircay.edu.tr

Abstract. Process mining utilizes event data to gain insights into the execution of processes. While techniques are valuable, their effectiveness may be hindered when dealing with highly complex processes that have a vast number of variants. Additionally, because the recorded events in information systems are at a low-level, process mining techniques may not align with the higher-level concepts understood at the business level. Without abstracting event sequences to higher-level concepts, the outcomes of process mining, such as discovering a model, can become overly complex and challenging to interpret, rendering them less useful. Some research has been conducted on event abstraction, often requiring significant domain knowledge that may not be readily accessible. Alternatively, unsupervised abstraction techniques may yield less accurate results and rely on stronger assumptions. This paper introduces a technique that addresses the challenge of limited domain knowledge by utilizing a straightforward approach. The technique involves dividing traces into batch sessions, taking into account relationships between subsequent events. Each session is then abstracted as a single high-level activity execution. This abstraction process utilizes a combination of automatic clustering and visualization methods. The proposed technique was evaluated using a randomly generated process model with high variability. The results demonstrate the significant advantages of the proposed abstraction in effectively communicating accurate knowledge to stakeholders.

Keywords: Event log abstraction · Parallel relationships · Clustering · Visual analytics · Process discovery.

1 Introduction

In today's data-driven business landscape, organizations have access to vast amounts of event data generated by their information systems. This event data captures valuable insights into how processes are executed in the organization. Process mining techniques have emerged as powerful tools to analyze and leverage this event data to gain valuable insights into process execution. However, as processes become more complex and exhibit a large number of variants, the effectiveness of traditional process mining techniques is diminished [14]. Moreover, the low-level nature of the recorded events in information systems often fails to capture the higher-level concepts known at the business level, further complicating the analysis and interpretation of process mining results.

The need to abstract low-level event sequences into higher-level activities has been recognized as a crucial step in enhancing the usability and interpretability of process mining outcomes [11]. Traditional approaches to event abstraction often rely on significant domain knowledge, which may not be readily available or feasible to obtain. Thus, there is a pressing need to develop a technique that enables effective event abstraction without relying on extensive domain knowledge and with improved accuracy.

Existing approaches assume that each low-level sequence that is abstracted to one high-level activity does not interleave with other low-level sequences. In other words, there are no parallel high-level activities. This is a strong assumption because parallel activities are performed simultaneously, and the execution order changes in different traces.

This paper puts forward a technique that detects parallel low-level sequences by leveraging a combination of clustering and visualization methods. The proposed technique first identifies session creation rules by evaluating the time threshold and the parallel relationship between subsequent low-level events. Events having parallel relationships are assigned into different sessions to catch the parallelism. Then, it encodes the sessions as vectors, with each dimension representing an activity associated with a low-level event. A clustering algorithm groups the vectors of similar low-level sequences into clusters, where each cluster includes a set of parallel events. An abstracted model is generated using high-level event sequences. Then, each cluster, including sublogs, is discovered for sub-models, and sub-models are replaced with the corresponding high-level activity on the abstracted model. After this replacement, the model's quality is measured with different parameter values. The parameter values giving the best quality are chosen to use for the evaluation of the abstraction technique.

The effectiveness of the proposed technique was evaluated through different synthetic processes and event logs. The results showed that the abstraction technique enables to mining process models with a better balance of fitness and precision.

2 Motivation

This section motivates the importance to consider the interleaving of low-level sequences. The discussion is based on the work by de Leoni et al. [8], but a similar reasoning applies when employing other techniques that overlook the parallelism of high-level activities [14]. The event-log abstraction technique in [8] divides traces into sessions, where each session represents the smallest subsequences within a trace. A session is determined based on a user-defined threshold, which is the time difference between the last event in a session and the first event in the subsequent session.

Let us consider the model in Figure 1 to motivate the importance of consider parallelism of high-level activities. The model consists of three parallel branches, each of which corresponds to one high-level activities. High-level activities A and B consists of the sequence of three low-level ones, in the model represented by the sequences $\langle A1, A2, A3\rangle$, $\langle B1, B2, B3\rangle$. Similarly, high-level activities C consists of the sequence of two low-level activities. The corresponding event log records the execution of these the low-level activities, and no information is stored about the high-level counterparts.

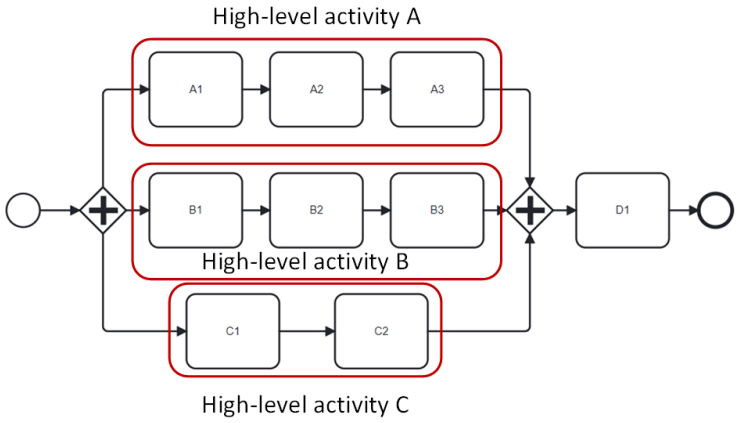

Fig. 1. An illustrative model including three parallel low-level sequences corresponding three different high-level activities

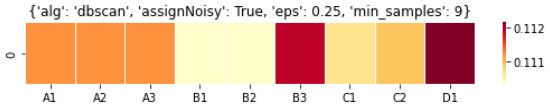

Fig. 2. Heatmap of clustering results for the technique in [8]. It highlights how each cluster has one or two evident dominant low-level activities (the cells of the darkest colors) according to the cluster centroids. There is only one cluster and the dominant low-level activities $D1$, $B3$, etc.

The timestamps of the low-level events are temporarily space out with a distance of one second from each other.

After applying the technique by de Leoni et al. [8] using a threshold of nine seconds, the outcome is one single cluster in which every low-level activity is added. This can be visualized through the heatmap in Figure 2, which illustrates the centroid of the created cluster. The most frequent low-level activities are given darker colours.

The expectation is to obtain four clusters, namely for the low-level activities related to the high-level A, B, C and D. However, since the low-level activities are given timestamps with distance of one second against a threshold session of nine seconds, the technique assigns every low-level activities into the same session, making it impossible to create the expected clusters. This can be solved by observing that low-level activities must not be in the same cluster if they can occur in any order, namely they are parallel. This idea motivates the research work reported in this paper. The remainder will detail the techniques. When parallelism is considered, we obtain the clusters depicted in Figure 3. The high-level activity A is covered via Cluster 0 where the most frequent low-level activities are indeed $A1$, $A2$ and $A3$. Clusters 1 and 3 incorporate the low-level activities related to the high-level activities C and B, respectively. Note the presence of clusters 2, 4 and 5, which are respectively related to high-level activities B, A and C: in these clusters, a predominant low-level activity is also $D1$, which indeed

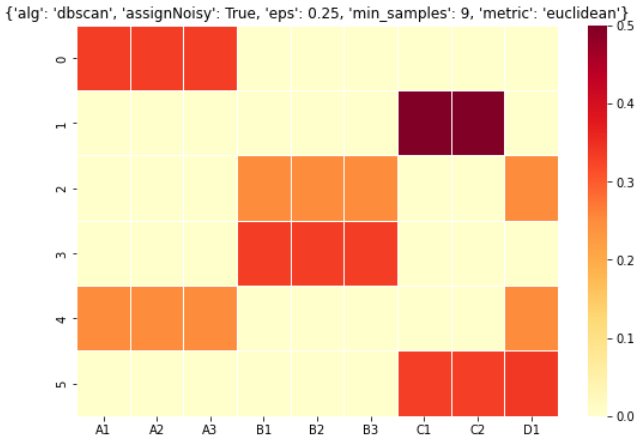

Fig. 3. Heatmap of clustering results for the proposed technique. Six clusters have different dominant low-level activities, and none of the clusters has low-level activities having a parallel relationship.

may follow $A3$, $B3$ or $C2$. In fact, the expected cluster for high-level activity D has been merged with those for A, B and C. The approach is thus not exactly achieving the four clusters, as expected, but yet is able to approximate the expected results.

3 Related Work

The problem of event-log abstract is currently gaining momentum, but most of the techniques do not consider the parallelism among high-level activities. Bakullari and van der Aalst [3] developed a framework for generating high-level events by dividing events into time windows. Van Eck et al. [12] presented an approach for creating high-level event logs from sensor data of smart products using segmentation and clustering techniques. Similarly, Brzychczy et al. [4] employed a combination of hierarchical clustering and domain knowledge to identify process stages. Michael Mayr et al. [10] extended the approach proposed by van Eck et al. [12] by incorporating time-series-based process-state and product-state data as additional attributes in the event log. Augusto et al. [1] applied the *k*-th order Markovian abstraction technique to create high-level activities and then discovered process models to compare the quality of the models based on a family of fitness and precision measures.

Some techniques support high-level activities parallelism, but they rely on manual effort and domain expertise. Mannhardt and Tax [9] suggested first discovering local process models from the event log to capture the behavior of specific subsets of events. The local process models were then used to lift the event log to a higher level of abstraction. Van Houdt et al. [13] employed a method that combines local process models with behavioral pattern mining to identify event abstraction patterns, leveraging on the event-log transformation by Mannhardt and Tax [9]. Baier et al. [2] leverage on domain knowledge to guide the event abstraction algorithm that also supports parallelism. By

incorporating domain knowledge, the algorithm achieves a more precise and contextually relevant event abstraction.

However, domain knowledge is often not readily available in real life. This paper presents a novel approach that considers parallelism, while not requiring domain knowledge.

4 Preliminaries

The initial step of an abstraction technique involves utilizing an *event log*, which comprises a collection of *traces*. Each trace represents a *process instance* (i.e., a *case*) manipulated in terms of *activities* performed and *process attributes* that are influenced.

Definition 1 (Events). *Let A be the set of activity names, and T be the universe of timestamps, and I be the universe of event identifiers. An event is a triple $e = (a, t, i) \in A \times T \times I$ and represents that the event with identifier i refers to the execution of an instance of activity a for at timestamp t.*

We use shortcuts $\lambda_A(e) = a, \lambda_T(e) = t, \lambda_I(e) = i$ to extract the activity name a, the timestamp t, and identifier i of any event e. Events may define several additional attributes, but they are not formalized here, to keep the notation simple. A trace refers to a sequence of events:

Definition 2 (Traces and Event Logs). *A trace σ is a sequence of events, i.e., $\sigma \in \mathcal{E}^*$. An event log L is a set of traces, i.e., $\mathcal{L} \subset \mathcal{E}^*$.*

In the remainder, for each pair (a, b) of activities and each event log $\mathcal{L}$, $|a >_{\mathcal{L}} b|$ is used to denote the number of occurrences of events for activity a followed by events for b in $\mathcal{L}$. The concept of parallelism used here is based on the dependency measured introduced within the Heuristic Miner [11]:

Definition 3 (Dependency Measure). *Let A be the set of activity names, and T be the universe of timestamps, and I be the universe of event identifiers. Let $\mathcal{E} = A \times T \times I$ be the university of events. Let $\mathcal{L} \subset \mathcal{E}^*$ be an event log. The dependency measure for each pair $(a, b) \in A \times A$ is defined as follows:*

$$a \Rightarrow_{\mathcal{L}} b = \begin{cases} \frac{|a>_{\mathcal{L}} b| - |b>_{\mathcal{L}} a|}{|a>_{\mathcal{L}} b| + |b>_{\mathcal{L}} a| + 1} & \text{if } a \neq b \\ \frac{|a>_{\mathcal{L}} a|}{|a>_{\mathcal{L}} a| + 1} & \text{if } a = b \end{cases}$$

The value of $a \Rightarrow_{\mathcal{L}} b$ is always between -1 and 1. If the absolute value of $a \Rightarrow_{\mathcal{L}} b$ is lower than a user-defined threshold, and at least once a is followed by b in $\mathcal{L}$ or the other way around, activities a and b are considered in parallel:

Definition 4 (Parallelism). *Let A be the set of activity names, and T be the universe of timestamps, and I be the universe of event identifiers. Let $\mathcal{E} = A \times T \times I$ be the university of events. Let $\mathcal{L} \subset \mathcal{E}^*$ be an event log. Given two activities $a, b \in A$ and a dependency threshold Θ, activities a and b are consider parallel:*

$$a \parallel_{\mathcal{L},\Theta} b \Leftrightarrow |a >_{\mathcal{L}} b| + |b >_{\mathcal{L}} a| > 0 \wedge |a \Rightarrow_{\mathcal{L}} b| < \Theta$$

The use of a threshold is to be tolerant to noise. In an ideal world, activities are parallel if we see once a followed by b, and also b followed by a.

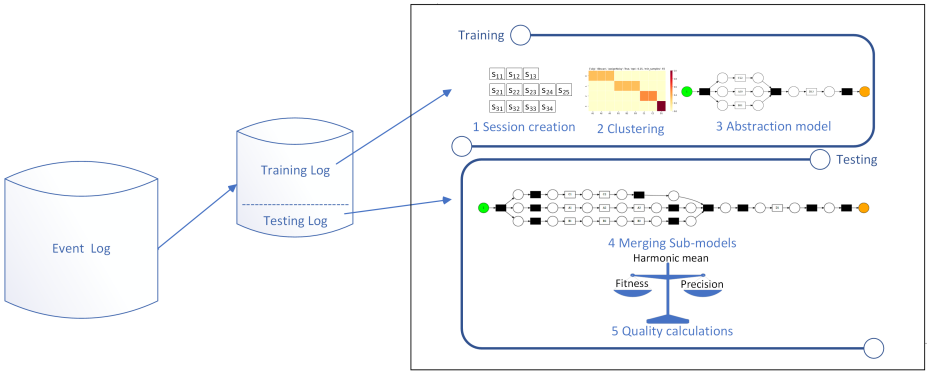

Fig. 4. Abstraction technique framework

5 Abstraction Technique

Figure 4 depicts the steps of the proposed abstraction technique. The input consists of an event log, which is split into training and testing. The training event log is used to create sessions at step 1: the events of each trace of the event log are divided into sessions. A session is a sequence of events. At step 2, sessions are clustered, where each cluster represents a high-level activities. The sessions of each cluster are the sequence of low-level activities that are mapped to the high-level activity corresponding to that cluster. At step 3, the low-level event sequences are replaced by high-level events to obtain an abstracted event log: this abstracted event log can then be used, e.g., to mine a high-level model. The sessions of each cluster can be seen as a sublog that can be used as input for process discovery: this would discover a low-level process model of how each high-level activity is mapped to the executions of low-level activities. The low-level process models can then be merged with the abstract process model to in fact obtain a hierarchical model: this is step 4. At step 5, a testing event log is used to compute the harmonic mean of fitness and precision, and thus to assess the quality of the hierarchical model.

This procedure relies on the DBSCAN clustering algorithm: the above procedure is repeated for different values of the algorithm's parameters ϵ and $min\_samples$, aiming to find those that allow mining a hierarchical model that maximizes the harmonic mean of fitness and precision.

The remainder of this section provides further details of the different steps of the proposed techniques.

5.1 Parallelism-based Session Creation

Algorithm 1 illustrates how sessions are created. The input is the event log $\mathcal{L}$, a time threshold Δ and the dependency threshold Θ (cf. Equation 4). Values observed for Θ in [11] are usually between 0.7 and 0.9.

As illustrated in Algorithm 1, the creation is done by trace (see line 2). For each trace $\sigma \in \mathcal{L}$, a set S_σ of the sessions of σ is created to include a session that consists

Algorithm 1: Session Creation based on Parallelism

Input : An event log $\mathcal{L}$, A session time-threshold Δ, A dependency threshold Θ
Output: A set S of sessions

```
1  S ← ∅
2  foreach trace σ = ⟨e_1, ··· , e_n⟩ ∈ L do
3      Trace-session set S_σ ← {⟨e_1⟩}
4      for event index i=2 to n do
5          event_assigned ← False
6          foreach session s = ⟨f_1, ··· , f_m⟩ ∈ S_σ do
7              if (a ‖_{L,Θ} b ∧ (λ_T(e_i) − λ_T(f_m) < Δ)) then
8                  S_σ ← (S_σ \ s) ⋃ {⟨f_1, ··· , f_m, e_i⟩}
9                  event_assigned ← True
10                 break
11             end
12         end
13         if event_assigned = False then
14             S_σ ← S_σ ⋃ {⟨e_i⟩}
15         end
16     end
17     S ← S ⋃ S_σ
18 end
19 return S
```

of the sequence with only the first event (see line 3). In fact, a session for a trace σ is a sequence of events of σ. The loop from line 4 to 16 iterates over all events of the trace of length n, and adds each event e_i either to an existing session or to a new session. The inner loop from line 6 to 12 iterates over the existing sessions: if any session s concludes with an event f_m that is not parallel with e_i and also the time difference between the two events is lower than the time threshold Δ (conditions at line 7), e_i is added to f_m (see line 8) and no more sessions are considered (see break of the inner loop at line 10). A special case is when the event is not added to any of the existing session because the conditions at line 7 are never both satisfied. In this case, the inner loop between 6 and 12 concludes. The boolean variable $event\_assigned$ stays set to false: it would become true if and only if it is added to some session (cf. line 9 where the variable is set to true). The condition at line 13 refers to this case: if the event is assigned to no session, a new session is created and added to the existing set S_δ At the end of this step, every trace is broken down into the constituent sessions, then clustered as discussed in the next step.

5.2 Clustering

Each session *s* is converted into a vector via an encoding function $\text{ENCODE}(s)$ that abstract the behavior observed in the session. All these vectors (e.g. the sessions) are clustered. The encoding can be obtained in many ways. Here we consider $\text{ENCODE}(s) = (v_{x_1}, \ldots, v_{x_n})$, where $x_1, \ldots, x_n$ are the activities observed in the log and any v_{x_i} is obtained in one of the two following alternatives:

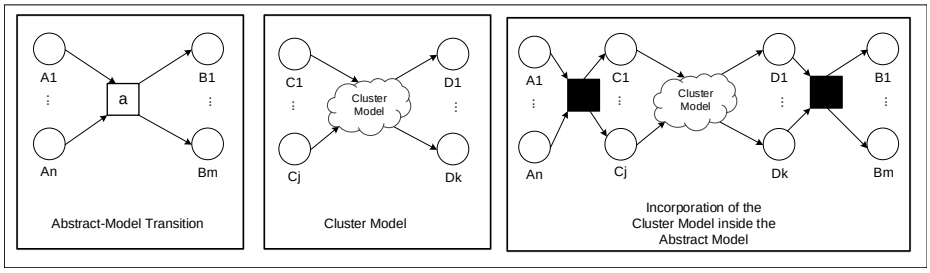

Fig. 5. Merging one Cluster model into the Abstract Model (source [8]).

.

Frequency-based encoding. The value of dimension v_{x_k} is the number of events for activity x_k in s. If both starting and completion events are observed for an activity x_k, the number of events is ultimately divided by two, to prevent double counting.

Duration-based encoding. The value of dimension v_{x_k} is the average duration of instances of activity x_k in the session s (a zero value is given if x_k does not occur in s).

Once sessions are converted into vectors, they are clustered. Here, we opt for the clustering algorithm DBSCAN, which are shown to lead to better results [7, 8] Clusters are visualized on the heatmap for insights into how sessions were clustered. The x-axis refers to low-level activities, whereas the y-axis indicates clusters. The intersection of the axes is colored according to the value of the dimension a in cluster c. A value of 0 is represented as white, while 1 is given as the most intense red. The cluster centroids $(c_1, \cdots, c_k)$ are normalized by dividing the sum of the centroid values $(\frac{c_1}{\sum_{j=1}^{k} c_j}, \cdots, \frac{c_1}{\sum_{j=1}^{k} c_k})$. Varying intensities of the color red represent the centroids on the heat maps.

5.3 Creation of Abstracted Model

The third step focuses on the mining of an abstract model, namely the model consisting of the the high-level activities. The input of this step is the set C of clusters of sessions. Each cluster $c \in C$ is associated with a high-level activity $\text{NAME}(c)$. Since high-level activity names are not resulting from the clusters, we visualize the centroids of the clusters on a heatmap, which provides insights to process analysts into the names to assign to clusters, and consequently to high-level activities, in line with what discussed in [7, 8]. This enables the creation of the abstract event log. For each log trace σ, we compute the sessions' sequence $Ⓢ_{\Delta}(\sigma) = \langle s_{\sigma_1}, \ldots, s_{\sigma_m} \rangle$ and then the abstract trace $\sigma_{\text{ABST}} = \langle f^{st}_{\sigma_1}, f^{co}_{\sigma_1} \ldots, f^{st}_{\sigma_n}, f^{co}_{\sigma_n} \rangle$ where, for every $1 \leq i \leq m$, $f^{st}_{\sigma_i}$ and $f^{co}_{\sigma_i}$ are the abstract events for session s_{σ_i}: the activity names of $\lambda_A(f^{st}_{\sigma_i})$ and $\lambda_A(f^{co}_{\sigma_i})$ are equal to the name $\text{NAME}(c_i)$ of the cluster c_i to which s_{σ_i}. The so-constructed abstract event log can then be used to discover an abstract model.

5.4 Merging Sub-cluster Models

Each cluster $c \in C$ consists of a set of sessions, which is in fact an event log to be used as input to mine a model of the behaviour of the sessions within the cluster. Here we also assume to employ a miner that returns strongly connected Petri nets. Figure 5 illustrates how one cluster model can be incorporate into the abstract model to create the hierchical model: the leftmost part shows the portion related to the high-level activity a where the center part depicts the structure of the model of the cluster referring to high-level activity a. The incorporation of the cluster model into the abstract model is illustrated in the rightmost part of the figure, where two invisible transitions are introduced (the black squares) in the abstract model along with the cluster model, which is connected as shown in figure. This procedure is repeated for each cluster, thus finally producing the hierarchical model. Note that often those invisible transitions are not necessary (see, e.g., the case study): in that case, the invisible transitions can finally be removed.

5.5 Computing Model Quality

Quality computing for a process model involves evaluating the discovered process model's accuracy, reliability, and usefulness. Fitness reflects how well the model aligns with the actual execution of the process, with higher fitness values indicating a closer match between the observed and modeled behavior [11]. Precision evaluates how effectively the model represents the actual process without oversimplifying or omitting essential details [11]. In particular, fitness is computed on the test event log, whereas precision considers train and test event log together for more reliability. This computation of the model quality is then returned as the harmonic means of fitness and precision, which heavily penalizes configurations where the fitness and precision values are not well balanced.

6 Evaluation

To evaluate the effectiveness of the proposed technique, we employed two artificial processes with synthetic event logs. In particular, we confronted the quality of the hierarchical model obtained using the abstraction technique introduced in this paper with the models obtained using no event-log abstractions and the technique by de Leoni et al. [8], which overlooks parallelism.

Processes and Logs Generator 2 (PLG2) was used to create the artificial models and accordant event logs [5]. PLG2 allows to generate random process models and to create accordant event logs. PLG2 also allows noise to be added, in terms of percentage of events that are randomly inserted, deleted or swapped.

Two processes were randomly generated, containing 109 and 119 activities for model 1 and 2, respectively. Initially, we generated logs with no noise: The average number of events per trace is respectively 107.41 and 105.01. Then, we also generated new event logs with 5%, 10%, 15%, 20% of noise, which were used to define the noise's probabilities of three types: trace missing head probability, trace missing tail probability, and trace missing episode probability. Because PLG2 generates a log with fixed

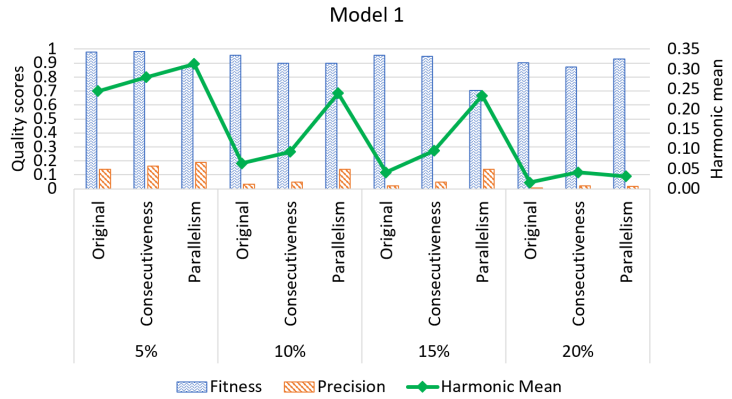

Fig. 6. Quality of Model 1 in terms of the harmonic mean of fitness and precision

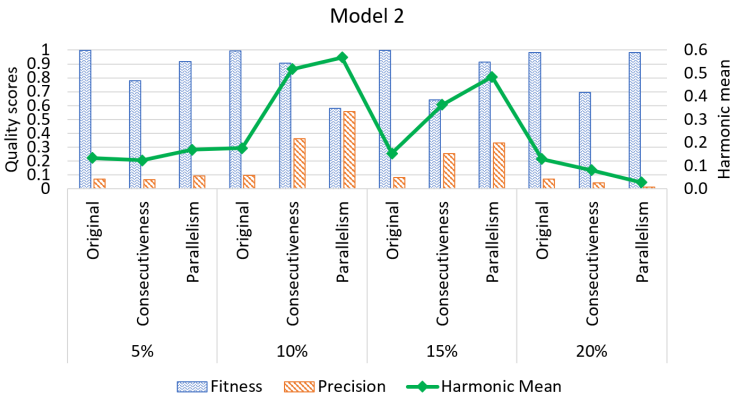

Fig. 7. Quality of Model 2 in terms of the harmonic mean of fitness and precision

1-hour timestamps for each consecutive activity, we changed the timestamps randomly, considering relationships among activities.

Our technique has been implemented in in Python, whose source code is available at https://github.com/onurdogannet/ParallelismbasedEventLogAbstraction, where we also uploaded the generated models and logs.

Our technique leverages on a process discovery: we used Inductive Miner with a noise threshold of 0.2 [6]. The session threshold has been set to be the average time between consecutive events: this is used both for the technique proposed in this paper and for the technique reported in [7, 8]. Figures 6 and 7 provide an evaluation of process models based on their fitness, precision, and harmonic mean scores for the increasing level of noise in the event log. The models include three technique: *Original*, namely without using event-log abstraction, *Consecutiveness*, which is the technique reported in [7, 8], and *Parallelism*, which is what proposed in this paper.

The original models show high fitness scores but low precision. On the other hand, abstraction techniques based on consecutiveness and parallelism result in lower fitness scores but improved precision, in general.

Across different session creation methods (Original, Consecutiveness, Parallelism), there are variations in fitness. For example, in Model 1, the Parallelism variation tends to have lower fitness compared to the Original or Consecutiveness, especially at higher noise levels. In Model 2, the fitness of the Consecutiveness variation varies significantly with noise levels. It's important to note that Model 1 generally exhibits higher fitness scores compared to Model 2 under the same conditions and noise levels.

Precision varies across different noise levels and model names. In Model 1, the Parallelism variation tends to have higher precision compared to the other two variations across noise levels. In Model 2, the precision of the Parallelism variation is notably high, especially at 10% noise level, suggesting it is better at avoiding false positives under certain conditions.

The parallelism-based abstraction achieves a better balance between fitness and precision compared to the consecutiveness-based abstraction for both event logs, except the event logs with 20% noise. Because both abstraction methods consider noises by assigning them to the nearest cluster before creating the abstracted model, they gave better results until a certain noise level. However, after a certain level of noise, abstraction techniques may struggle to cope with the increased variation introduced by the noise, leading to reduced model quality. Intuitively, as the noise level continues to increase beyond 20%, the model quality is expected to deteriorate further. At 25% or 30% noise levels, the proportion of irrelevant events becomes even larger, making it more challenging for the abstraction techniques to identify relevant patterns and relationships. This can lead to a higher number of incorrect or misleading abstractions, resulting both fitness and precision to decline.

7 Conclusion and Future Research

This paper addresses the challenges traditional process mining techniques face in dealing with complex processes and low-level event data. By introducing a novel event abstraction technique that requires limited domain knowledge, the paper offers an approach to enhance the interpretability and usability of process mining outcomes. The proposed technique's effectiveness is validated through different noise levels, emphasizing its ability to provide accurate knowledge to stakeholders. The improved interpretability and usability of the abstracted process model enable users to gain valuable insights and make informed decisions regarding process analysis, optimization, and improvement.

For noise levels up to 20%, the abstraction techniques perform better than the not abstracted model because they can effectively filter out some noise and separate the essential patterns. The fact that parallelism is the best abstraction technique at noise levels less than 20% can be attributed to its ability to handle more diverse and complex event patterns. Parallelism captures relationships among high-level activities that can occur simultaneously. The choice of the best abstraction technique depends on the specific characteristics and noise level of the event log being analyzed.

As future work, we aim to experience alternative clustering algorithms, such as hierarchical clustering, which may lead to better clustering results, and ultimately to hierarchical models of higher quality. So do we plan to work on automatically finding a suitable time threshold for session creation.

Acknowledgements. This research is financially supported by the Department of Mathematics of the University of Padua, through the BIRD project "Web-site Interaction Discovery" (code `BIRD219730/21`).

References

1. Augusto, A., Armas-Cervantes, A., Conforti, R., Dumas, M., La Rosa, M.: Measuring fitness and precision of automatically discovered process models: a principled and scalable approach. IEEE Transactions on Knowledge and Data Engineering **34**(4), 1870–1888 (2020)
2. Baier, T., Mendling, J., Weske, M.: Bridging abstraction layers in process mining. Information Systems **46**, 123–139 (2014)
3. Bakullari, B., van der Aalst, W.M.: High-level event mining: A framework. In: 2022 4th International Conference on Process Mining (ICPM). pp. 136–143. IEEE (2022)
4. Brzychczy, E., Trzcionkowska, A.: Process-oriented approach for analysis of sensor data from longwall monitoring system. In: Intelligent Systems in Production Engineering and Maintenance. pp. 611–621. Springer (2019)
5. Burattin, A.: PLG2: multiperspective process randomization with online and offline simulations. In: Proceedings of the BPM Demo Track 2016. CEUR Workshop Proceedings, vol. 1789, pp. 1–6. CEUR-WS.org (2016)
6. Leemans, S.J.J., Fahland, D., van der Aalst, W.M.P.: Discovering block-structured process models from event logs - a constructive approach. In: Proceedings of the 34th International Conference on Application and Theory of Petri Nets and Concurrency. LNCS, vol. 7927, pp. 311–329. Springer (2013)
7. 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)
8. de Leoni, M., Pellattiero, L.: The benefits of sensor-measurement aggregation in discovering iot process models: a smart-house case study. In: International Conference on Business Process Management. pp. 403–415. Springer (2021)
9. Mannhardt, F., Tax, N.: Unsupervised event abstraction using pattern abstraction and local process models. arXiv preprint arXiv:1704.03520 (2017)
10. Mayr, M., Luftensteiner, S., Chasparis, G.C.: Abstracting process mining event logs from process-state data to monitor control-flow of industrial manufacturing processes. Procedia Computer Science **200**, 1442–1450 (2022)
11. Van Der Aalst, W.M.P.: Data science in action. Springer (2016)
12. Van Eck, M.L., Sidorova, N., Van der Aalst, W.M.: Enabling process mining on sensor data from smart products. In: 2016 IEEE Tenth International Conference on Research Challenges in Information Science (RCIS). pp. 1–12. IEEE (2016)
13. Van Houdt, G., Depaire, B., Martin, N.: Unsupervised event abstraction in a process mining context: A benchmark study. In: Process Mining Workshops: ICPM 2020 International Workshops, Padua, Italy, October 5–8, 2020, Revised Selected Papers 2. pp. 82–93. Springer (2021)
14. van Zelst, S.J., Mannhardt, F., de Leoni, M., Koschmider, A.: Event abstraction in process mining: literature review and taxonomy. Granular Computing **6**, 719–736 (2021)