Visually Representing History Dependencies in Event Logs

Manuel Wetzel[1], Agnes Koschmider[1], and Thomas Wilke[2]

[1] Group Process Analytics, Kiel University, Germany
manuelwetzel2203@web.de, ak@informatik.uni-kiel.de
https://www.pa.informatik.uni-kiel.de/en/
[2] Group Theory of Computing, Kiel University, Germany
thomas.wilke@email.uni-kiel.de

Abstract. Many process mining tools produce directly-follows graphs (DFG) as visual representations of event logs. While the "directly follows" relation is a good starting point for visualizations, there are simple phenomena it does not capture, for instance, when whether or not an event directly follows another event depends on the event directly preceding it. We call this a *history* dependency. This paper presents an empirical study of preferences for visualizing history dependencies: plain DFGs and two enhanced variants of DFGs (with additional arcs or rectangles) are evaluated. Our empirical study provides strong support for making an effort (to discover and) to explicitly visualize history dependencies. A ProM plug-in generating such explicit visualization is described in this paper.

Keywords: directly-follows graph · process visualization · process discovery · history dependency · empirical study.

1 Introduction

Event log files are used as input to any process mining algorithm aiming to discover an as-is process model, to analyze processes or to identify bottlenecks. To reduce inappropriate conclusions from the discovered process model, it is essential that this model reflects the reality found in an event log as best as it can. Mostly, available commercial process mining tools produce a visualization of a directly-follows graph (DFG) as a representation of event logs. While the "directly follows" relation is a good starting point for a visualization, there are simple phenomena it does not capture, for instance, when whether or not an event directly follows another event depends on the event directly preceding it. Fig. 1 illustrates this phenomenon in terms of traces.

According to Fig. 1 a) the execution of activity D right after C is only allowed when A was executed directly before C. When activity B was executed directly before C, then only E may directly follow, but D must not. We call this event dependency a *history dependency* (more precisely, a history-1 dependency). Fig. 1 b) shows a Petri net modeling the dependencies exactly.

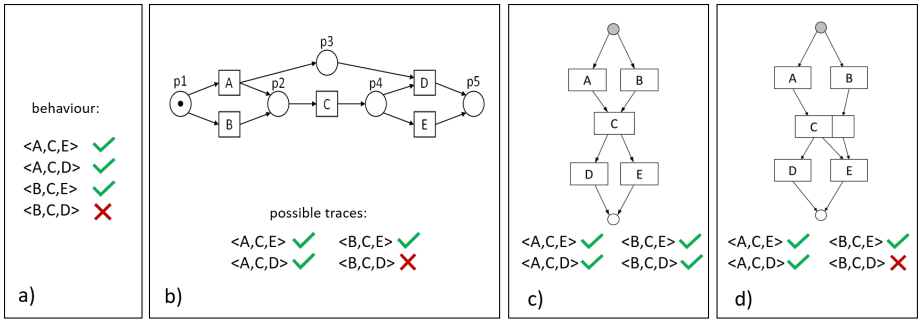


Fig. 1: a) example of a history dependency; b) discovered Petri net; c) discovered DFG; d) proposed visualization

A process mining algorithm based on the directly follows relation would produce the DFG in Fig. 1 c). This DFG allows behavior that is not reflected in reality, namely the trace $\langle \mathsf{B}, \mathsf{C}, \mathsf{D} \rangle$. Although limitations of DFGs have been demonstrated [9], the graphical visualization as DFG is still a common practice for available commercial process mining tools. The motivation behind our research, therefore, is not to resort to a completely different type of visualization, but rather to study visual enhancements of DFGs being suitable to visualize history dependencies. For the example given in Fig. 1 a), we propose the visualization in Fig. 1 d). Another example is given in Fig. 2. Mining an event log as given in Fig. 2 a) leads to the DFG in Fig. 2 c), which has an unintended cycle; we propose the visualization in Fig. 2 d), which reflects reality exactly. A process model exactly modeling the event log is given in Fig. 2 b).

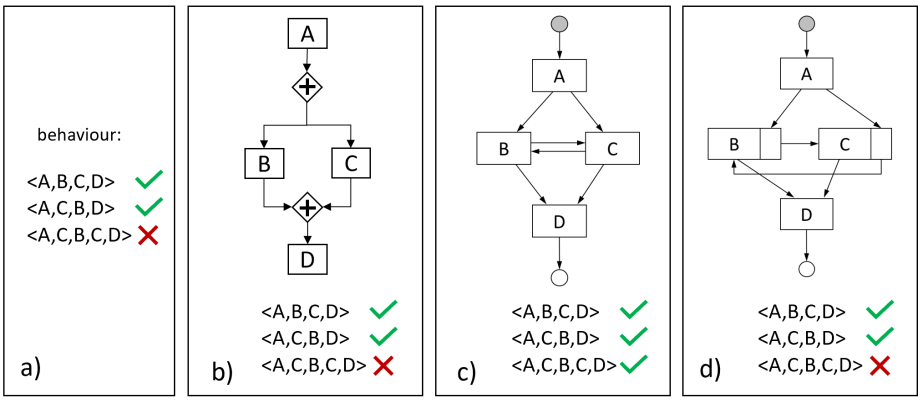


Fig. 2: a) a second example of a history dependency; b) BPMN model; c) discovered DFG; d) proposed visualization

To learn more about history dependencies and enhancements of the DFG, we conducted a user study. We compared plain DFGs to two enhanced visualizations, one with additional arcs (AA) and another one with additional rectangles (AR). Our findings show strong support for enhancing DFGs to visualize history dependencies. Thus, our finding does not correspond to common practical implementations (where history dependencies are not explicitly visualized); this is why we developed a ProM plug-in that produces enhanced DFGs, more precisely, it produces AR visualizations.

This paper is structured as follows. The next section compares our work with related works. Section 3 discusses, beside the plain DFG, visualization variants for history dependencies. Section 4 describes the design of a study to provide evidence about the visualization strategies for history dependency. The results of the study are discussed in Sections 5 and 6. The ProM plug-in implementation is presented in Section 7. The paper concludes with a summary and an outlook.

2 Related Work

Process discovery algorithms generally distinguish between two types of dependencies [10]: explicit and implicit ones. An explicit dependency, which is also called direct or causal dependency [1], exists when an activity is directly followed by another activity in a considerable number of cases. An implicit dependency refers to various types of indirect (causal) relationships between activities, for instance, that an activity is eventually followed by another activity in a considerable number of cases. Dependency measures are used to determine whether or not and which kind of dependency is present. A history dependency in our sense takes into account causal dependencies that are not only concerned with two consecutive activities but a small number of consecutive activities [8]. Process models visualizing history dependency prevent, in some case, unintended cycles as demonstrated in Fig. 2 and thus overcome a limitation of the DFG discussed in [9].

We subsume our approach to techniques explicitly visualizing history- dependent information as in [5, 4]. Compared to these approaches we do not introduce a new visualization technique nor label the process model with additional information but rather enhance the DFG, which is commonly used in commercial process mining tools, aiming to reflect the reality found in an event log better than a plain DFG.

3 Visualization Techniques

To understand the usefulness of the Directly-Follows Graph in terms of visualizing history dependencies we compared it with two other visualization variants as discussed in this section. The visualization techniques lay the foundation for the empirical validation of visualization preferences for history dependencies presented in Section 4.

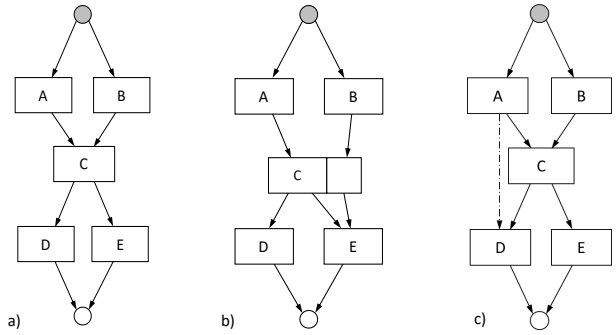


Fig. 3: Examples for a) DFG-based visualization, b) Visualization Additional Rectangle c) Visualization Additional Arc

3.1 Visualization Directly-Follows Graph

The DFG is widely used to visualize behavior between process activities. It gives information on a similar level of abstraction as the process modeling notations BPMN, EPK or Petri nets. Usually, commercial process mining tools also attach time and frequency based performance measures on the arcs for process monitoring reasons. Although the semantics of the DFG is easily understandable, the DFG does not represent a precise process model since it allows more behavior than has actually been recorded in the event log and can reasonably be expected as has shown in the introductory example. In this way, a history dependency is wrongly visualized and is even hardly spotted, which may lead to inappropriate conclusions inferred from the DFG. The following two visualization variants aim to overcome this issue.

3.2 Visualization Additional Rectangle

To visualize history dependency the plain DFG is enhanced with an additional rectangle. We insert to the activity routing the history dependency. We call this visualization technique "Additional Rectangle", see Fig. 3 b) for an example. Additional Rectangle (AR) is inspired by hyperedges, a well-known concept from graph theory, see [2]. Incoming and outgoing arcs of the additional empty rectangle indicate the allowed behavior between two activities. Trace replays on this model would be: $\langle \mathsf{A}, \mathsf{C}, \mathsf{D} \rangle$, $\langle \mathsf{A}, \mathsf{C}, \mathsf{E} \rangle$, $\langle \mathsf{B}, \mathsf{C}, \mathsf{E} \rangle$, which corresponds to the allowed behavior in Fig. 1 a). The trace $\langle \mathsf{B}, \mathsf{C}, \mathsf{D} \rangle$ cannot be replayed by this process model due to a missing empty rectangle at activity C. A more complex visualization of AR for large process models can be seen in 4 b).

3.3 Visualization Additional Arc

A second technique to visualize history dependencies is "Additional Arc" (AA), see Fig. 3 c). Here, the plain DFG is enhanced with dashed arcs. This visualization, like AR, explicitly illustrates a history dependency. The semantics of the

dashed arc is "activity D can only be executed if activity A has been executed previously". Thus, the process model visualized through AA also does not allow to replay the trace $\langle \mathsf{B}, \mathsf{C}, \mathsf{D} \rangle$ on the model. A complex example process model is shown in 4 a) visualizing a model with a total of seven history dependencies.

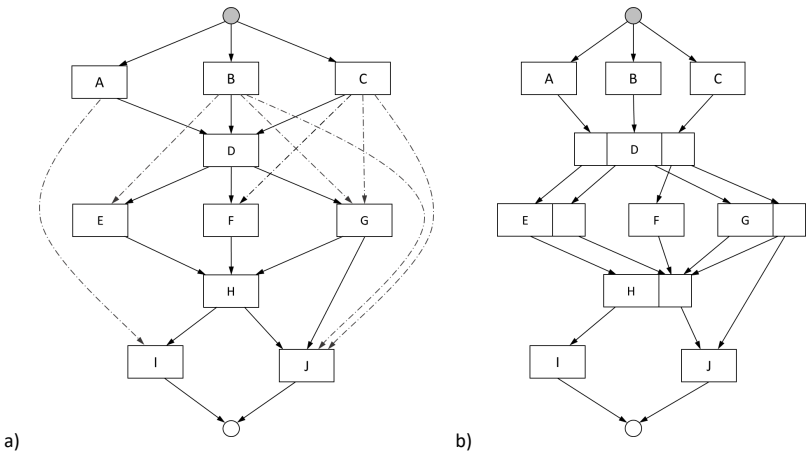


Fig. 4: Examples for visualizations a) Additional Arc b) Additional Rectangle for large process models.

4 Design Setting

To evaluate whether the proposed visualizations are suited to recognize history dependencies, a questionnaire was designed. The main focus of the study was to investigate the visualization preferences of the three visualization techniques presented in the previous section. Therefore, a web-based questionnaire was set up. Participants were free to answer the questions and could withdraw the completion of the questionnaire at any time and collection of data was anonymous. The following section describes the design of the questionnaire.

Objects. The objects evaluated by each participant were a set of traces and a total of eleven process models. Five of the process models were visualized according to the AR visualization, five with the AA visualization and one as a Directly-Follows Graph. Small process models (see Fig. 3) had six activities and one history-1 dependency. Large process models (see Fig. 4) were designed through ten activities having seven history dependencies including history-3 dependencies.

Response variable. The response variable in our study is the level of understanding that the respondents displayed with respect to a visualization technique recognizing history dependencies. Understandability is measured as follows:

- the perceived ease of understanding (PEOU),
- preference between the visualization techniques,
- the number of correct traces spotted for a visualization technique,
- degree of agreement for comprehension questions

Instrumentation. The questionnaire was constructed as follows. After a motivation in history dependency, we asked the participants to complete for each visualization technique a task and to answer comprehension questions. The semantics of each visualization technique was not formally introduced. The task was to evaluate against a set of traces whether traces can be replayed by the process model. We showed 4 traces for each small process model and 21 traces for each large process model. The description of a task was "*Evaluate for each trace if the trace can be replayed by the process model. Please tick the corresponding box.*". After having completed this task, we asked the participants to rank (on a 5 point Likert scale) visualization preferences to represent history dependencies and to rank the visualization techniques based on their ability to represent history dependencies in a process model. In the second part of the questionnaire the participants had to complete two further tasks for the visualization techniques AA and AR. The first task was to choose between true or false statements. We provided for each of the two visualization techniques the following statements "*Please consider the three process models and choose if one of these models describes the following behavior according to the Additional Arc visualization The model is able to replay the traces:* $\langle A, D, E\rangle$, $\langle A, D, F\rangle$, $\langle B, D, F\rangle$, $\langle C, D, F\rangle$. *and is not able to replay* $\langle B, D, E\rangle$ *and* $\langle C, D, E\rangle$ *Please select the correct representation of the wanted behavior"* also providing the option "*none of the shown models describes the desired behaviour.*". Subsequently, the participants had to complete tasks for the visualizations AA and AR for large process models and to rank their individual preferences for both visualization techniques.

Subjects. The survey was conducted in July 2020. The link to the survey was sent to the participants of the "Advanced Process Mining"[3] course at the Kiel University as well as a number of experts across different European universities.

Data collection. Along with the questionnaire, we asked the participants about the amount of experience they have in the fields of process mining and business process management. Furthermore, we asked whether they (practically) worked with process mining and if they understood the motivation behind history dependency that we presented at the beginning of the questionnaire.

5 Evaluation Results

Finally, the questionnaire was answered by 13 persons. 62% of the respondents had more than one year of experience in process mining and 38% had more than three years of experience. All participants used process mining in research. One

[3] Students of the "Advanced Process Mining" course also attended the course "Process Mining" in the winter term and thus had advanced process mining knowledge.

person worked with process mining in industry projects. The average time to complete the questionnaire was 41 minutes.

The results for visualization preferences were analyzed with respect to frequency distribution. Table 1 shows the statistical results for each preference, its answer options, the frequency in numbers per option, the frequency (%), and the cumulative frequency (%) for each question. Cumulative frequency is determined by aggregating agreement (strong agree, agree) and disagreement (disagree, strongly disagree) with the preference. Table 1 summarizes the statistical results. The results investigate the PEOU measure for each visualization technique while Table 2 shows the preference between the visualization techniques and an order between the three visualization techniques with respect to understandability. According to this result DFG is ranked in average 2.85 out of 3 with standard deviation of 0.36 meaning that it is less understandable as representation for history dependency. The Directly-Follows Graph is being perceived as not helpful to understand history dependencies. Visualization preferences for DFG received very low agreements (8% and 15%). In the ranking of individual preferences the DFG received the last position.

For small process models a slight preference exists for AR against AA visualization. 69% agreed that the AR visualization helped them better to recognize history dependencies, while 46% voted for AA as first choice. Related to individual visualization preferences AR was in average ranked (1.54 out of 3) compared to 1.62 for AA. The high standard deviation, however, might implicate that the participants are undecided.

A contrary individual preference is observed for large process models. The statement "*Additional Rectangle helped better than Additional Arc to recognize implicit dependency*" received an 85% approval, while the contrary statement only received a 23% approval. The preference for AR for large process models is also confirmed by the results in Table 2. The AR visualization was ranked 1.15 (out of 3), while the preference for AA declined to 1.85. So there is a significant preference for Additional Rectangle visualization over Additional Arc for larger process models when many implicit dependencies exist.

The results of user tasks (correct traces spotted for trace replay) are shown in Table 3. The tasks were not evaluated with respect to the correctness of an answer since no explanation of the semantics of each visualization technique was introduced to the users. Instead we compared the visualization techniques according to the overall consensus for each selection.

For each task multiple measures were calculated. The first three measures point to the consensus on a decision. This was measured by the relative amount of participants with identical answers. A consensus of 100% means that all participants made the identical decision, while a consensus of 50% means that half of the participants had the opinion that a trace can be replayed by the model while the other half had the opposite opinion. As for the trace replay tasks multiple decisions existed we determined the average [AVG] consensus of all traces within a task, the corresponding standard deviation [STD] and its minimum [MIN] which is the consensus on the most controverse trace selection.

Preference	Options	Freq	Freq. (%)	Cum. Freq. (%)
The visualization Directly-Follows Graph helped me better to recognize the implicit dependency over the visualization Additional Rectangle	s. agree	1	8 %	8 %
	agree	0	0 %	
	undecided	4	31 %	
	disagree	3	23 %	62 %
	s. disagree	5	38 %	
The visualization Directly-Follows Graph helped me better to recognize the implicit dependency over the visualization Additional Arc	s. agree	2	15 %	15 %
	agree	0	0 %	
	undecided	3	23 %	
	disagree	2	15 %	62 %
	s. disagree	6	46 %	
The visualization Additional Rectangle helped me better to recognize the implicit dependency over the visualization Directly-Follows Graph	s. agree	5	38 %	69 %
	agree	4	31 %	
	undecided	2	15 %	
	disagree	1	8 %	15 %
	s. disagree	1	8 %	
The visualization Additional Rectangle helped me better to recognize the implicit dependency over the visualization Additional Arc	s. agree	3	23 %	62 %
	agree	5	38 %	
	undecided	2	15 %	
	disagree	1	8 %	23 %
	s. disagree	2	15 %	
The visualization Additional Arc helped me better to recognize the implicit dependency over the visualization Directly-Follow Graph	s. agree	7	54 %	85 %
	agree	4	31 %	
	undecided	1	8 %	
	disagree	1	8 %	8 %
	s. disagree	0	0 %	
The visualization Additional Arc helped me better to recognize the implicit dependency over the visualization Additional Rectangle	s. agree	1	8 %	46 %
	agree	5	38 %	
	undecided	1	8 %	
	disagree	5	38 %	46 %
	s. disagree	1	8 %	
The visualization Additional Rectangle helped me better to recognize the implicit dependency over the visualization Additional Arc in large process models	s. agree	6	46 %	85 %
	agree	5	38 %	
	undecided	0	0 %	
	disagree	1	8 %	15 %
	s. disagree	1	8 %	
The visualization Additional Arc helped me better to recognize the implicit dependency over the visualization Additional Rectangle in large process models	s. agree	1	8 %	23 %
	agree	2	15 %	
	undecided	1	8 %	
	disagree	5	38 %	69 %
	s. disagree	4	31 %	

Table 1: Results of preferences

	Rank (AVG)	Rank (SD)
Additional Rectangle	1.54	0.63
Additional Arc	1.62	0.62
Directly-Follows Graph	2.85	0.36
Additional Rectangle for large models	1.15	0.36
Additional Arc for large models	1.85	0.36

Table 2: Results of ranking the proposed visualizations

The aggregated results show a consensus of 85% vs. 92% on average. Recall that for small process models users had to evaluate four trace replays. A value of 92% means that almost all participants had the same understanding of the AR visualization. Note, that it was a binary decision so the expected value of a random distribution would have been 50%. But not only the average overall decisions were better for the Rectangle visualization. The worst consensus was between 77% to 69% and also both calculated standard deviation metrics are smaller for AR. When evaluating trace replay for large process models (i.e., "*Please rate your visualization preferences to represent implicit dependencies in large process models*"), the consensus declines for both visualization techniques. But still, Additional Rectangle with a consensus of 90% is superior to Additional Arc. The last task (i.e., "*Please rank the presented visualization based on their ability to represent implicit dependencies in a process model*") shows again a clear preference for AR (ten out of 13 choose it), while AA was ranked the second with 54% of the participants.

Semantic	Task name	Cons.[AVG]	Cons.[STD]	Cons.[MIN]
DFG	Trace verification (small example)	100%	0%	100%
AR	Trace verification (small example)	92%	9%	77%
	Trace verification (large example)	90%	8%	69%
	Choose correct Model	77%		
AA	Trace verification (small example)	85%	15%	69%
	Trace verification (large example)	77%	17%	54%
	Choose correct Model	54%		

Table 3: Aggregated Results of the tasks

6 Discussion

Interpretation: The empirical study provides strong support for another visualization for history dependencies than the Directly-Follows Graph. Additional Rectangle, as well as Additional Arc, are better suited to visualize history dependencies. This finding does not directly correspond to common practical implementations. The DFG-based visualization, which is the common practice for available commercial process discovery tools, does not explicitly visualize history dependencies. Our statistical results show that AR and AA are easily understandable for users for small process models, while a strong support for AR was observed for large process models. Therefore, Additional Rectangle is a suitable alternative to current visualizations.

Implications: the design of a visual notation is a challenging task [7]. It requires a balance between *symbol deficit* (i.e., no constructs representing a graphical symbol), *symbol overload* (i.e., same graphical symbol for different representations), *symbol redundancy* (i.e., alternative graphical symbols for same representation) and *symbol excess* (i.e., showing all constructs on a diagram). The rejection of a DFG-based visualization is in line with the postulation of symbol deficit. When no construct is used to represent a graphical symbol then understandability decreases. Process discovery tools should implement an explicit visualization for history dependencies.

Limitations: The similar preference for Additional Rectangle vs. Additional Arc for small process models might be explained due to a weakness of understanding the semantics of the AA visualization. There are two contrary ways to interpret the dashed arc in 3 c). Either it is understood as "If A has been executed then it must be followed by D" or it can mean "activity D can only be executed if activity A has been executed before". Apparently, four out of twelve participants have intuitively interpreted the arc in the first way.

7 Implementation as ProM Plugin

In response to the evaluation results (and also the limitation of AA visualization discussed previously) we implemented a ProM plug-in, which experiments with visualizing history dependencies by enhancing the nodes in a DFG with additional rectangles. An overview of the components of the plug-in, called Dependent Directly Follows Model Miner (DDFM Miner), is shown in Figure 5. The plug-in carries out three sequential steps.

Step 1. The plain DFG is mined from an event log in XES format using the "DirectlyFollowsModelMiner" plugin [6], which produces a directly-follows matrix.

Step 2. History dependencies are being identified. For this purpose, the algorithm takes as input the directly-follows matrix from Step 1 and compares it with a Petri net manually mined from the same event log using a suited ProM plug-in miner (see "Preparation" in Fig. 5). The DDFM Miner then analyses every transition-place-transition triplet in the Petri net and determines whether

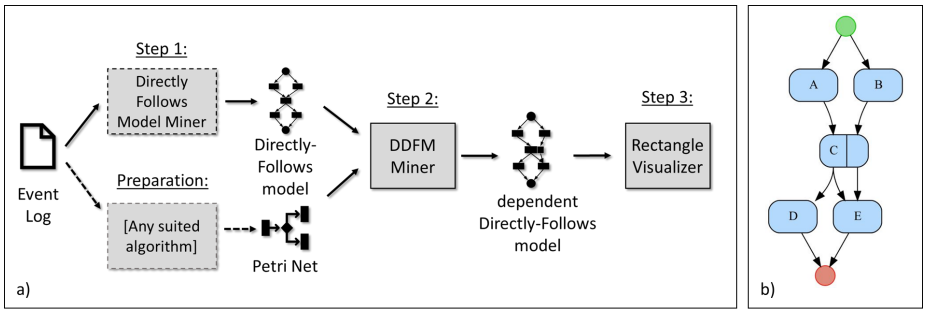


Fig. 5: a) Steps of the DDFM Miner and b) its output

this relation is already in the directly-follows matrix. If this is not the case, a history dependency is added to the history-dependency matrix.

Step 3. A visualizer plug-in relying on the Graphviz environment [3] translates the two matrices into a graphical representation. This includes four steps:

1. Calculate all *dependent paths* that connect two history dependent activities.
2. Pre-process node names to create the additional rectangles with Graphviz.
3. Generate nodes and *directly-follows edges*.
4. Iterate through *dependent paths* to remove *directly-follows edges* and add *dependency edges*.

Fig. 5 b) shows the final result of the DDFM miner.

8 Summary and Outlook

The simple understandablity of the DFG might be one reason for its high popularity. With regard to history dependencies, however, the DFG-based visualization fails. The objective of this work was to study how to visualize history dependencies. For this purpose, we compared the plain DFG with visualization variants in a user study. The results of the study provide strong support to enhance the plain DFG with additional rectangles to visualize history dependencies.

In response to our finding, the DDFM miner has been implemented, but this is only a first step. Future tasks are: (1) to evaluate the DDFM miner on large event logs, (2) to visualize history dependencies for multiple interfering dependent paths, as, for instance, present in Fig. 4: consider $\langle \mathsf{C}, \mathsf{G}, \mathsf{H}, \mathsf{J} \rangle$ and $\langle \mathsf{B}, \mathsf{D}, \mathsf{E}, \mathsf{H}, \mathsf{I} \rangle$, (3) To enhance the semantics of AR visualization (see Appendix) with quantitative aspects such as frequency and time since our definition of history dependency is a "discrete" one.

References

1. van der Aalst, W., Günther, C.: Finding structure in unstructured processes: The case for process mining. pp. 3 – 12 (08 2007)

2. Berge, C.: Graphs and Hypergraphs. North-Holland (1973)
3. Ellson, J., Gansner, E.R., Koutsofios, E., North, S.C., Woodhull, G.: Graphviz and dynagraph – static and dynamic graph drawing tools. In: GRAPH DRAWING SOFTWARE. pp. 127–148. Springer-Verlag (2003)
4. Goedertier, S., Martens, D., Baesens, B., Haesen, R., Vanthienen, J.: Process mining as first-order classification learning on logs with negative events. In: Business Process Management Workshops. pp. 42–53. Springer (2008)
5. van Hee, K.M., Oanea, O., Serebrenik, A., Sidorova, N., Voorhoeve, M.: History-based joins: Semantics, soundness and implementation. In: BPM 2006. Lecture Notes in Computer Science, vol. 4102, pp. 225–240. Springer (2006)
6. Leemans, S.J., Poppe, E., Wynn, M.T.: Directly follows-based process mining: a tool. In: ICPM 2019. pp. 9–12 (2019)
7. Moody, D.: The "physics" of notations: Toward a scientific basis for constructing visual notations in software engineering. Software Engineering, IEEE Transactions on **35**, 756 – 779 (01 2010)
8. Sun, H., Du, Y., Qi, L., He, Z.: A method for mining process models with indirect dependencies via petri nets. IEEE Access **7**, 81211–81226 (2019)
9. van der Aalst, W.M.: A practitioner's guide to process mining: Limitations of the directly-follows graph. Procedia Computer Science **164**, 321 – 328 (2019)
10. Wen, L., Wang, J., Sun, J.: Detecting implicit dependencies between tasks from event logs. In: APWeb 2006. pp. 591–603. Springer (2006)

A Rigorous Definitions

Let $\mathcal{A}$ be a finite set of *activities* without $\triangleright$ and $\square$. A *trace* is a finite sequence $\triangleright A_0 \dots A_n \square$ which satisfies $A_i \in \mathcal{A}$ for every $i \leq n$. An *event log* is a finite non-empty set of traces.

A *directly-follows graph with multiplicities (DFG+)* consists of a finite set V of vertices, a finite set $E \subseteq V \times V$ of edges, and a labeling function $\lambda \colon V \to \mathcal{A} \cup \{\triangleright, \square\}$ such that V together with E is a directed acyclic graph (DAG), there is exactly one vertex labeled $\triangleright$, and this vertex is the only vertex with no incoming edge, there is exactly one vertex labeled $\square$, and this vertex is the only vertex with no outgoing edge. A DFG+ represents the event log which consists of the labelings of all paths through it that are traces.

A DFG+ is a *directly-follows graph (DFG)* when every event is the label of at most one vertex. A DFG+ is *reduced* if every vertex is on some path from the vertex labeled $\triangleright$ to the vertex labeled $\square$ and there are no two distinct vertices with the same set of successor vertices.

Fact For every event log there is, up to isomorphism, exactly one reduced DFG+ that represents it. Such a DFG+ is called *canonical* for the event log.

Definition Let $\mathcal{D}$ be a canonical DFG+ for an event log. 1. The event log is *history-free* if $\mathcal{D}$ is a DFG.

2. Let k be a non-negative integer and $A \in \mathcal{A}$. There is a *k-conflict* for A if there are paths $v_0 v_1 \dots v_k$ and $w_0 w_1 \dots w_k$ in $\mathcal{D}$ such that $v_k \neq w_k$, $\lambda(v_i) = \lambda(w_i)$ for all $i < k$, and $A = \lambda(v_k) = \lambda(w_k)$. There is a *$k$-history dependency* ($k > 0$) for A if there are no k-conflicts for it, but a $(k-1)$-conflict.

Analysis of Business Process Batching using Causal Event Models

Philipp Waibel[0000−0002−5562−4430], Christian Novak[0000−0003−4367−223X], Saimir Bala[0000−0001−7179−1901], Kate Revoredo[0000−0001−8914−9132], and Jan Mendling[0000−0002−7260−524X]

Institute for Information Business, Vienna University of Economics and Business (WU), Vienna, Austria
{philipp.waibel,saimir.bala,kate.revoredo,jan.mendling}@wu.ac.at
christian.novak@s.wu.ac.at

Abstract. Process mining supports business process management with operational insights extracted from event logs. A key challenge for process mining is that operational processes in production and logistics often include batching and unbatching, e.g., to delivery several packages using one truck tour. Such n:m relations blur the notion of a process instance and make the causality between events difficult to trace. In this paper, we address this research problem by introducing causal event models that capture batching behavior accurately. To this end, we construct conflict-free prime event structures for event instances of the event log, and devise various analysis techniques on top of them. We implemented the techniques in a tool and run in real data of a manufacturing company with various 1:n and n:1 relations in their production process showing the potential of our approach.

Keywords: Process mining · Business process modeling · Batching · Causality.

1 Introduction

Business Process Management (BPM) comprises the various management activities that help organizations to discover, analyse, implement and monitor their processes [10]. Recently, BPM has become increasingly evidence-based thanks to advancements of process mining [1]. The availability of event log data from enterprise systems for various business processes is one of the key drivers of these developments as much as the commercial tool support.

Various algorithms have been proposed that support automatic process discovery, conformance checking, enhancement, or analysis of variants [1,10]. One aspect of specific interest to the process analyst is the batching behavior of processes, i.e. the merge of different objects either within the same case or between different cases. This practice contributes to both, more cost-efficient processing as well as delays. In order to grasp the batching behavior precisely, the knowledge about causality between the events is necessary. Without this knowledge spurious batches may occur.

In this paper, we address this research problem by introducing causal event models that capture batching behavior accurately. To this end, we construct conflict-free prime event structures, inspired by [3], for process instances stored in event logs, before devising various analysis techniques on top of them. Our use case demonstrates the benefits of our technique for the case of a manufacturing company with various 1:n and n:1 relations in their production process.

The rest of the paper is structured as follows. Section 2 discusses a motivational scenario for our work. Subsequently, Section 3 presents prior work related to our research problem with a focus on n:m relations and batch in business processes. Section 4 presents the conceptual foundations of our technique. Section 5 describes findings from applying our technique for a production process and discusses the lessons learned. Finally, Section 6 concludes the paper outlining future research.

2 Motivational Scenario

To motivate the presented work, we refer to an order-to-cash process example of one of our industry partners called Pastamaker (a pseudonym). Pastamaker's business is producing and delivering pasta to major supermarket chains in Austria. Their order-to-cash process is triggered by direct orders of a supermarket. These orders contain a list of items, where each of these items needs to be picked separately from the warehouse. In the next step, each item is packaged and sent as one or multiple deliveries. Each order generates an invoice that is settled and closed by a payment. In addition, packaging and delivering steps have sub-steps that are creating packaging notes and group delivery information.

Pastamaker uses batching at various stages of its production process, most importantly for bundling deliveries. For instance, one order can trigger different deliveries, and one delivery can include items from different orders. Such delivery batches are of central importance for keeping the operational costs of the process low. To further optimize the batching of the orders, Pastamaker would like to analyze them. In particular, Pastamaker would like to get information, like how many batching events took place, what events caused batches, or which steps are bottlenecks or caused delays.

3 Related Work

Perspectives on this problem have been discussed in two main streams of research: *i)* n:m relations in business processes; and *ii)* batching, which includes modeling batches and extracting knowledge about batches from event logs.

For what concerns stream *i)*, various works consider the problem of bundlings and unbundlings as well as multiple instance activities for business processes. Gerke et al. construct end-to-end case identifiers for RFID-based logistic processes by propagating order identifiers to subsequent and subordinate events [15,16]. Approaches by Weber et al. [26] and by Conforti et al. [7] present algorithms that discover process models with multiple instance activities. Lu et al. [21] present an approach that maintains complex event relations based on database

schemas to construct artifact-centric models with causal relations. Berti and van der Aalst [6] provide support for exploring event logs stored in databases from multiple viewpoints. Esser and Fahland [11,12] use *labeled property graphs* to capture the concept of one event being part of multiple cases. González López de Murillas et al. [24] identify interesting case notions from databases, while heuristics for finding suitable case identifiers are evaluated by Bala et al. [4]. Li et al. [19] follow another approach and create an object-centric event log format that does not require a case notion as it is required for the XES format. In this publication, Li et al. argue that this object-oriented event log format helps to store relations in the form of 1:n and n:m as it is common in databases. The problem involved in the usage of classic "flattened" event logs is also discussed in [2]. To solve the issue, the author proposes an object-centric process mining approach. The majority of these works do not consider the causality relations between events or do not explicitly represent batching nodes in a tool that can be used by a process analyst for batch analysis.

There are works that take into causality along with n:m relations. Dumas and García-Bañuelos [9] discuss a process mining approach based on prime event structures. This publication transforms the cases in an event log into prime event structures and then use the concept of asymmetric event structures to create a process model. This approach is further used in [3] to diagnose behavioral differences between business process models. A similar approach is chosen by Ponce de León et al. in [18]. The authors use event structures together with the concept of occurrence nets to create process models. In [5], the author also uses prime event structures as an intermediate step to create a process model. However, in comparison to [3] and [18], the author uses a different approach for creating the process model out of the separated prime event structures. Similar to our approach, these approaches utilize the concept of event structures. However, they use the event structures to create process models by combining the event structures. Thus, they are losing the possibility of considering the single process instances separated from each other, which is an essential part of analyzing different batching events.

For what concerns stream *ii)*, Fahland [13] presents the concept of event synchronization, which is related to the concept of batching. He emphasizes that proper semantics for processes with many-to-many interactions require, among others, cardinality constraints. Research on modeling batch behavior in a business process addresses this point at least at the type level. Pufahl et al. [25] extend BPMN with a specific batch activity type that considers an activation rule, a grouping attribute, a maximum batch size, and an execution order along with the definition of corresponding operational semantics. Martin et al. [23] present batching metrics for identifying patterns in event logs that point to batches. These include frequency of batch processing, batch size, instances per batch, duration and waiting times of instances in batches and temporal overlaps of batches. Klijn and Fahland [17] devise an algorithm to mine some of those metrics. Furthermore, Martin et al. define a mining technique for discovering batch activation rules in event logs [22] assuming that any observation of events done by the same

resource doing the same activity for different cases represents a batch. Without knowledge about causality between events, this assumption may lead to spurious batches. Lu et al. [20], Diamantini et al. [8] and Genga et al. [14] consider the causality between events by modeling the traces as a partial order of its events. However, these works are different than ours since they only model batching behavior within a process instance leaving out inter-case batching. Finally, a combination of mining (i) n:m relations and (ii) presenting batching behaviour based on causal event models is a novel approach.

4 Batch Analysis based on Causal Event Models

This section proposes our approach to discover specific batching behavior, which explains characteristics in batching behavior from an event log. Section 4.1 presents the underlying formal concept used to capture causally related data from an ERP system. Section 4.2 discusses how all relevant batching nodes are identified, how batches are visualized and insights into batching behavior is presented. Finally, Section 4.3 discusses the implementation of our approach.

4.1 Determine Causal Event Models for Event Log

The first step of our approach is concerned with identifying the causal relations between event instances of the event log. To this end, we make use of foreign key relationships between entities of the database schema. Based on these relationships, we are able to reconstruct which events have triggered each other in passages of the process that exhibits 1:n or n:1 relationships like in order:delivery (n:1). As a formal structure for representing causal event models of the event log, we build on conflict-free prime event structures.

Definition 1 (Conflict-free prime event structure). *A labeled* conflict-free prime event structure *is defined by the tuple* $cf\_PES = \langle E, \leqslant, \lambda \rangle$*, where* E *is a set of events,* $\leqslant$ *defines the causality relation as a partial order on* E *and* $\lambda : E \rightarrow \Lambda$ *is a labeling function.*

A cf_PES is based on the *prime event structure (PES)* as defined in [3] excluding conflict relations. The latter are excluded from cf_PES, because conflicts and decisions that where made during the process execution are not visible in the event log, and therefore they can also not be represented in our causal event model. A cf_PES is equivalent to the notion of labeled partial order. As an example for a cf_PES, Fig. 1 depicts the order-to-cash processes from our industry partner discussed in Section 2. The example in Fig. 1 shows one order with three separate items that got picked individually from the warehouse and then delivered in one package.

If the whole event log is described by using a cf_PES, we call it a database of conflict-free prime event structure (DB_PES) or a *causal event log*. Such a DB_PES can contain several separated cf_PES, but also cf_PESs that share

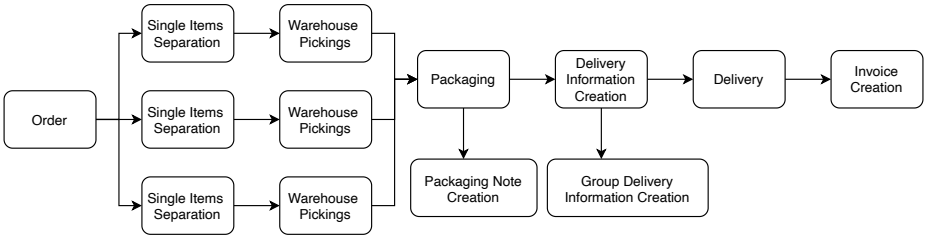


Fig. 1: Example cf_PES for the Motivational Scenario.

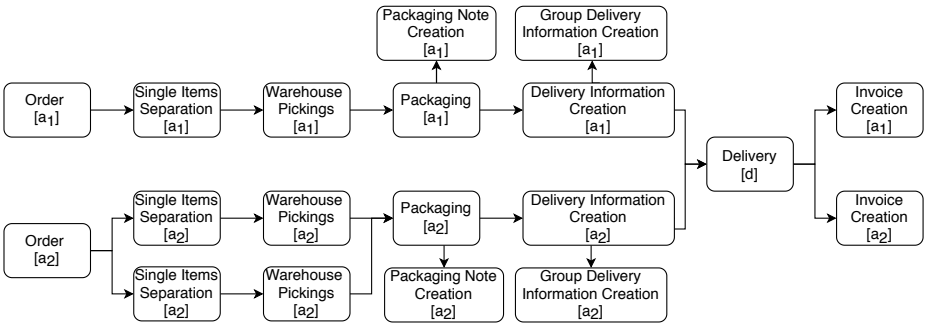


Fig. 2: Example DB_PES for the Motivational Scenario. The Numbers in Square Brackets Depicts the Order Affiliation.

one or several events. We call these shared events *batch nodes*, since they are bundling together several process instances.

For example, in the running order-to-cash process example, the orders a_1, and a_2 might be bundled into one delivery d. A corresponding $cf\_PES = \langle E, \leqslant, \lambda \rangle$ would then be composed of $E = \{a_1, a_2, d\}$, $\leqslant = \{(a_1, d), (a_2, d), (a_3, d)\}$, and $\lambda = \{(a_1, order),\ (a_2, order),\ (d, delivery)\}$, when we omit the other events. Fig. 2 depicts these two order-to-cash cf_PES. As can be seen in Fig. 2 the two individual orders (the text in square brackets represents the event affiliation to E) share a common delivery event.

4.2 Batching Analysis

Once the DB_PES is created, various analysis operations can be performed to analyse the data. In the work at hand, the functionality to analyse the batching behavior is presented. In the following, we will first define some preliminaries, and then the analysis approach.

Definition 2 (Event Type, Case Identifier, Node Identifier, Preset and Postset Nodes). *Given E is a set of events and ET is a set of event types, we define $\tau : E \rightarrow ET$ as the function that returns the type of an event. We define $e^{\tau} = \tau(e)$. Furthermore, we define $id : E \rightarrow I$ as an index function for the node ID, and $c\_ids : E \rightarrow I$ as an index function that defines the case IDs of the cases that are using an event $e \in E$. Furthermore, for a relation $R \subseteq E \times E$, we define for an $e \in E$ the preset of nodes $\bullet e = \{x \mid (x, e) \in R\}$ and the postset of nodes $e\bullet = \{x \mid (e, x) \in R\}$.*

To perform an analysis of the batching operations, we devise an algorithm that first identifies all batching nodes $E_{batches}$ of a DB_PES, i.e., $E_{batches} = \{e | e \in E \wedge |c\_ids(e)| > 1\}$. In a second step the algorithm identifies for each $e \in E_{batches}$ the cf_PES that contains the batching node, defined by $cf\_PES^e_{batch}$. This is done by iterating through the nodes of the DB_PES with $\bullet e$ and $e\bullet$, starting from the batch node $e \in E_{batches}$, until the start and end nodes, i.e., $|\bullet e| = 0$, respectively $|e \bullet| = 0$, are reached. The start and end nodes are stored in E^e_{start} and E^e_{end}.

Depending on the size of the $cf\_PES^e_{batch}$, the visualisation can be too crowded for a clear visualisation. To overcome this problem, the algorithm aggregates all nodes with the same event type, i.e., e^τ, of the $cf\_PES^e_{batch}$ together. This aggregation step also counts the quantity of the aggregated nodes and relationships, and the cardinality in a form of {1:1, 1:N, N:1, N:M} of each relationship. Additionally, the algorithm gets all preceding nodes of the batching events, i.e., $E^e_{prec} = \{\bullet e | e \in E_{batches}\}$. Eventually, the collected information, i.e., $cf\_PES^e_{batch}$, $E^e_{batches}$, E^e_{start}, and E^e_{end}, is returned by the algorithm.

The returned information can then be used to visualize and analyze the batches. To this end, we use a visualization based on the aggregated $cf\_PES^e_{batch}$ in a way that all cf_PESs are shown and the corresponding batch nodes (stored in $E^e_{batches}$) are highlighted. Furthermore, we provide for each $e \in E^e_{batches}$: event type e^τ, the batching factor $|c\_ids(e)|$ (i.e., the size of the batch), the execution start and end time of e, the average duration of the start of the preceding nodes E^e_{prec} to the start of the batch node e, the earliest start time in E^e_{start}, and the latest end time in E^e_{end}.

4.3 Implementation

Our approach has been implemented as a proof-of-concept, called Causal Miner. The prototype is developed in Java and uses the Spring Framework (vers. 2.3.0). We store the cf_PESs built from the event log in the graph database Neo4j (vers. 4.0.4) and use Cypher as query language. At the current state, the event log data can be read from Oracle DB as well as from Microsoft SQL Server. The visualisations are provided in a Web UI. The source code of the proof-of-concept prototype can be accessed on GitHub (https://github.com/piwa/causal-miner).

For the analysis of the batching operations, several Cypher queries are used. Two of them are presented in Listing 1.1 and Listing 1.2. The former returns the $E_{batches}$'s ordered by the batching factor. To limit the result, the query returns only the three $E_{batches}$'s with the highest batching factor, however, this limitation is configurable. Listing 1.2 returns the $cf\_PES^e_{batch}$ for the $e \in E_{batches}$ with the node $ID(e) = 5647$. For a better readability, we replaced some configuration parameters in line 5 with "$\cdots$".

Listing 1.1: Neo4j Cypher Query to Find all Batching Nodes.

```
1  MATCH (n:InstanceActivity) WHERE size(n.instanceIds) > 1
2  RETURN DISTINCT ID(n) AS batchNodeId, size(n.instanceIds) AS batchSize,
       n.instanceIds AS batchInstanceIds
3  ORDER BY batchSize DESC LIMIT 3;
```

Listing 1.2: Neo4j Cypher Query to get all cf_PES that Share a Common Batching Node.

```
MATCH (n:InstanceActivity) WHERE ID(n) = 5647
MATCH p=(:InstanceStartActivity)-[*1..10]->(n)-[*1..10]->(:
    InstanceEndActivity)
UNWIND nodes(p) AS unwindedNodes
WITH collect(distinct unwindedNodes) AS collectedNodes
CALL at.ac.wuwien.extendedGroup(...) YIELD node, relationship
RETURN collect(distinct node) AS modelActivityList, collect(distinct
    relationship) AS modelRelationshipList
```

5 Results and Evaluation

In this section we evaluate our prototypical implementation illustrating how it enables both visual and quantitative analyses of batching.

5.1 Setup and Dataset

The evaluation uses a real-world dataset from our industry partner, the previously mentioned food production company Pastamaker. The dataset is composed of the order-to-cash processes of the company. In total the dataset contains nearly 70,000 orders with more than 8,500,000 events. As discussed in Sect. 2, the process is composed of the following steps: A new instance is triggered by a supermarket order. The order is then broken down into single order items. These items are then picked from the warehouse separately. The items are then packed, and sent as one or multiple deliveries. As a last step, the invoice is created. Some of these steps also contain substeps and can be shared by several orders. For instance, it is often the case that several orders are delivered together, i.e., sharing the same delivery event, as depicted in Fig. 2.

The company uses an ERP system, which is build on an SQL database. For the evaluation we first import the data from this SQL database into the Neo4j graph database according to the approach presented in Section 4. Subsequently, we perform different analysis steps on the Neo4j data.

5.2 Visualization of the Results

The Causal Miner, offers different ways to work with the data. Figure 3 shows a screenshot of the Causal Miner. This screenshot exhibits an aggregated view of the DB_PES, which brings together all event nodes of the same type. It also shows the node and relationship quantity (in parenthesis), and the cardinality.

Along with the aggregated view presented in Fig. 3, the Causal Miner offers several other visualisation methods, such as methods to validate the cf_PESs according to a given process structure, to visualise single cf_PESs, and to represent the activity durations in a GANTT charts. The work at hand focuses on the visualization of batching. A screenshot of this view is presented in Fig. 4. The main UI consists of two tabs. In the Filter tab, several filtering functionalities can be selected. In the Upper Batches tab, the results of the batching queries

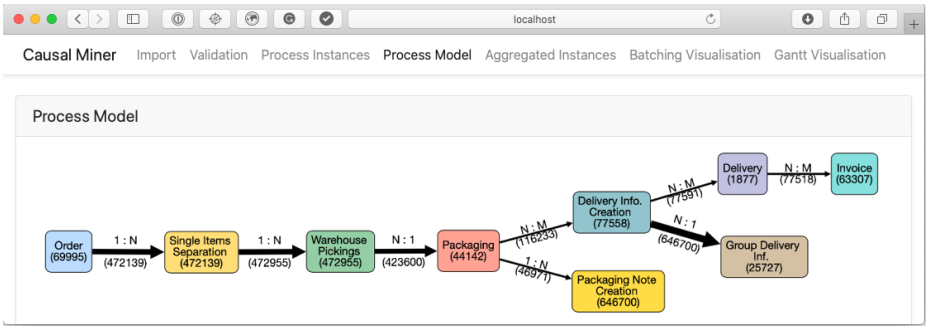


Fig. 3: Aggregated View of the DB_PES.

are visualised. The view depicted in Fig. 4 is showing the batching node with the highest batching factor, together with the corresponding $cf\_PES^{e}_{batch}$. The tables on the right side, show the information about the batches that are gathered by the algorithm discussed in Sect. 4.2. If a deeper analysis of the $cf\_PES^{e}_{batch}$ is required, an analyst can click on the *Show Instances* link, under the table. This link opens the view depicted in Fig. 5 that shows all cf_PESs that are involved in the current batching node, together with information about the single events.

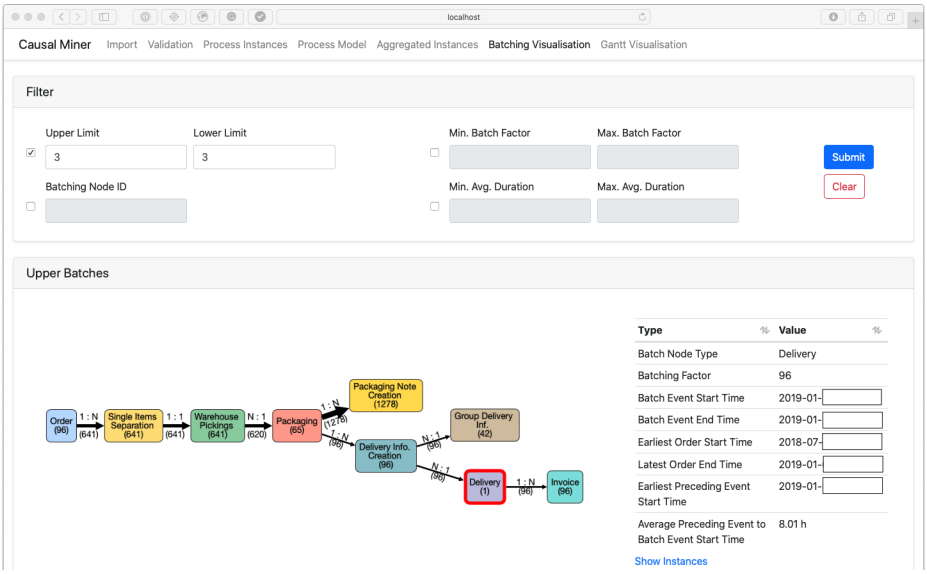


Fig. 4: Filtering Options and the Batching Node, including the Aggregated cf_PES, with the Biggest Batch Factor (exact times obfuscated due to privacy).

5.3 Data Analyses

Besides the visualisation possibilities shown in Section 5.2, the approach presented in Section 4, can be used for more detailed quantitative data analyses.

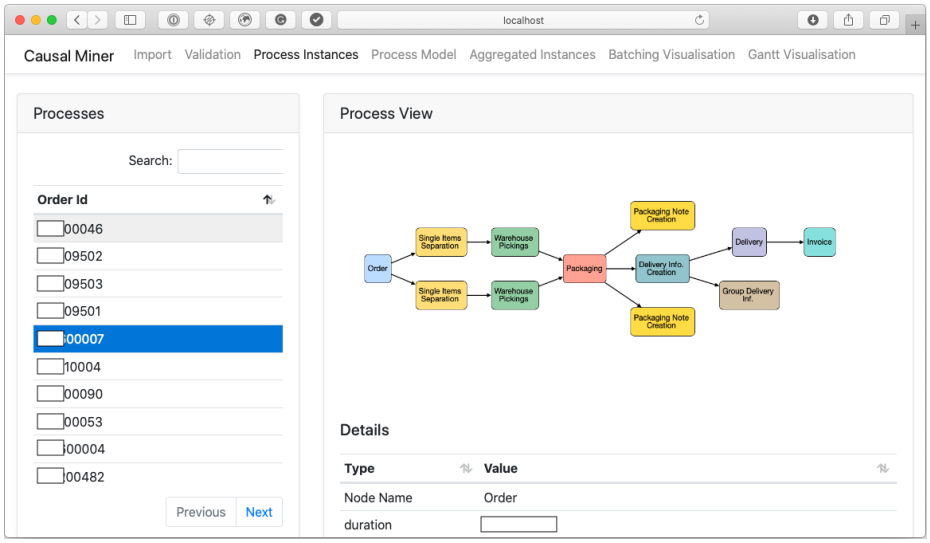
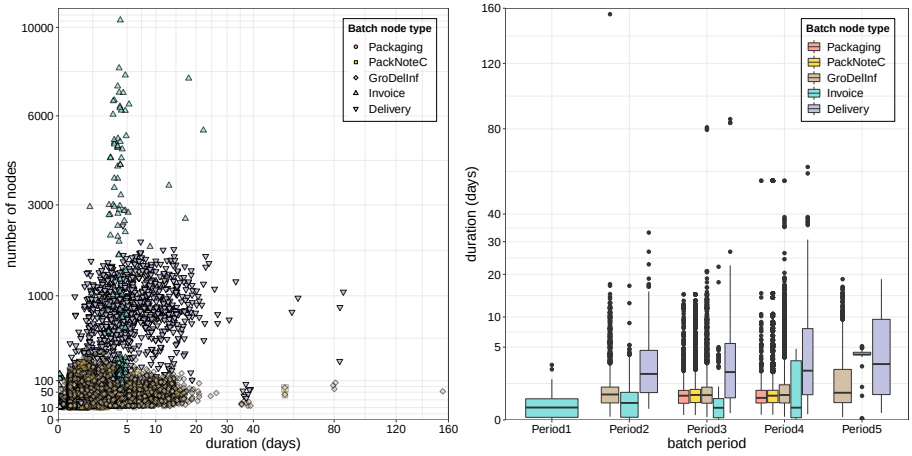


Fig. 5: Depiction of all cf_PES that Share a Particular Batch Node (note: some Information was Anonymized due to Privacy Concerns).



(a) Total Node Count versus Duration. (b) Batching in Different Time Periods.

Fig. 6: Data Analyses of Batching.

Figure 6 provides two types of analyses that allow us to gather interesting insights into the batching. Figure 6a presents a scatter plot that shows the total amount of nodes before the batching node versus the duration between the earliest start time in E^e_{start} and the latest end time in E^e_{end}. Each event type is plotted with a distinct shape. Figure 6a helps visualizing at least three main clusters of batching. Especially, it is possible to observe that *Invoices* are typically batched in five days. Figure 6b depicts a boxplot that shows the duration between the earliest start time in E^e_{start} and the latest end time in E^e_{end} in different time periods for each event type. Moreover, it can be observed that most of the time,

the orders that are batched in the *Delivery* event need more time than the orders that are batched in different events. The shorter durations in the other case are partially due to corrections in the bookings without logistic activities and internal orders without deliveries.

The presented charts enable a process analyst to quickly identify outlier instances of the process. These outliers can then be analysed in a greater detail with the help of the Causal Miner by analysing the $cf\_PES^{e}_{batch}$ and the involved cf_PES. Figure 7 presents an outlier that was identified from the scatter plot.

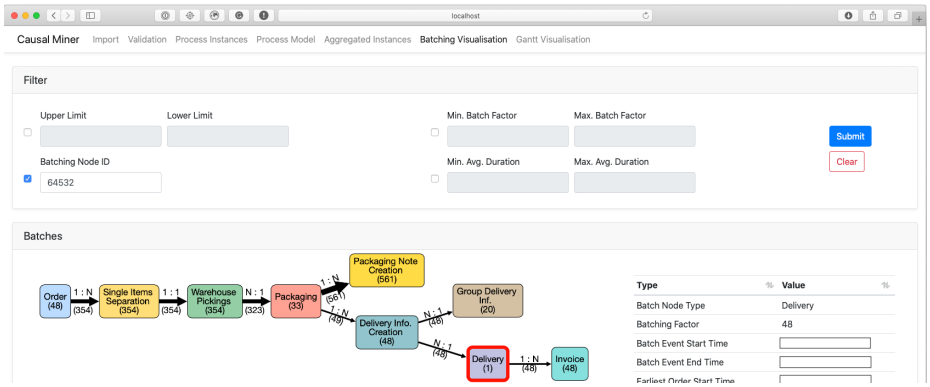


Fig. 7: Identified Outlier (Exact Times Obfuscated due to Privacy).

5.4 Discussion

As shown in the evaluation, our approach provides different ways to analyze the batching behavior: First, the Causal Miner provides a way to visualize the processes as a whole, together with the event quantities and relationship cardinalities, by using the aggregated view of the *DB_PES*. Second, the Causal Miner provides a view to analyze the batching behavior of the processes. This view further provides a filter functionality to analyze the batching behavior regarding different aspects. At the current state, this filtering provides, e.g., the means to filter for the batching factor. Third, a separate view allows the analysis of the batched processes on the instance and single events level. Fourth, the approach provides different ways to analyze the batching by using plots. As shown, these plots can be used to analyze the batching behavior and to detect outliers. These outliers can then be analyzed further by using the process visualizations.

These functionalities provide an analyst a way to start with a high-level analysis of the batching, using the plots and the filters, and then dig deeper into the batching behavior by analyzing the aggregated and the single process instances. Moreover, an analyst can even go to the level of single events. In addition to the presented features, the Causal Miner provides different other views, like representing the process instances as a GANTT chart.

Since our approach considers all events stored in a graph-based database, in our case Neo4j, query performance also plays an important role. In our evaluation with the 8,500,000 events, queries that are searching for specific events or process

instances need less than 200ms. Only queries like the one presented in Listing 1.1 needs around 7.5s since this query searches for all batching nodes, and Listing 1.2 needs around 1.6s. The evaluation was done on a server with eight cores with 2,6 GHz and 47GB RAM.

6 Conclusion

This paper introduces a technique to use causal event models to capture batching behavior. Our approach can be used to identify batches and determine its most important attributes, which helps to retrieve further insights of the batch processing. The algorithm is evaluated on real-world event logs, showing the practicability and usability of the approach. The resulting data can be used to discover batches and understand their context.

The implemented approach shows that there are important factors of batching. The processes are batched by different node types which have different characteristics. Important differences can be seen in complexity of the process and total duration time of batches. Process analysts can use the data to conduct further performance analysis and trigger process improvements.

Future work will aim at empirically comparing alternative batches in processes and further improve batching analysis methods. This includes automatic identification of outliers and automatic evaluation of its causing process instances. Another direction for future research involves the automatic suggestions for process improvement based on the batch analysis.

References

1. van der Aalst, W.M.P.: Process Mining - Data Science in Action, Second Edition. Springer (2016)
2. van der Aalst, W.M.P.: Object-centric process mining: Dealing with divergence and convergence in event data. In: SEFM. Lecture Notes in Computer Science, vol. 11724, pp. 3–25. Springer (2019)
3. Armas-Cervantes, A., Baldan, P., Dumas, M., García-Bañuelos, L.: Diagnosing behavioral differences between business process models: An approach based on event structures. Inf. Syst. **56**, 304–325 (2016)
4. Bala, S., Mendling, J., Schimak, M., Queteschiner, P.: Case and activity identification for mining process models from middleware. In: PoEM. Lecture Notes in Business Information Processing, vol. 335, pp. 86–102. Springer (2018)
5. Bergenthum, R.: Prime miner - process discovery using prime event structures. In: International Conference on Process Mining, ICPM 2019, Aachen, Germany, June 24-26, 2019. pp. 41–48. IEEE (2019)
6. Berti, A., van der Aalst, W.: Extracting multiple viewpoint models from relational databases. In: Ceravolo, P., van Keulen, M., Gómez-López, M.T. (eds.) Data-Driven Process Discovery and Analysis. pp. 24–51. Springer International Publishing, Cham (2020)
7. Conforti, R., Dumas, M., García-Bañuelos, L., Rosa, M.L.: BPMN miner: Automated discovery of BPMN process models with hierarchical structure. Inf. Syst. **56**, 284–303 (2016)

8. Diamantini, C., Genga, L., Potena, D., van der Aalst, W.M.P.: Building instance graphs for highly variable processes. Expert Syst. Appl. **59**, 101–118 (2016)
9. Dumas, M., García-Bañuelos, L.: Process mining reloaded: Event structures as a unified representation of process models and event logs. In: Petri Nets. Lecture Notes in Computer Science, vol. 9115, pp. 33–48. Springer (2015)
10. Dumas, M., Rosa, M.L., Mendling, J., Reijers, H.A.: Fundamentals of Business Process Management, Second Edition. Springer (2018)
11. Esser, S., Fahland, D.: Storing and querying multi-dimensional process event logs using graph databases. In: BPM Workshops. LNBIP, vol. 362, pp. 632–644. Springer (2019)
12. Esser, S., Fahland, D.: Multi-dimensional event data in graph databases. CoRR **abs/2005.14552** (2020)
13. Fahland, D.: Describing behavior of processes with many-to-many interactions. In: Petri Nets. Lecture Notes in Computer Science, vol. 11522, pp. 3–24. Springer (2019)
14. Genga, L., Alizadeh, M., Potena, D., Diamantini, C., Zannone, N.: Discovering anomalous frequent patterns from partially ordered event logs. J. Intell. Inf. Syst. **51**(2), 257–300 (2018)
15. Gerke, K., Claus, A., Mendling, J.: Process mining of rfid-based supply chains. In: CEC. pp. 285–292. IEEE Computer Society (2009)
16. Gerke, K., Mendling, J., Tarmyshov, K.: Case construction for mining supply chain processes. In: BIS. Lecture Notes in Business Information Processing, vol. 21, pp. 181–192. Springer (2009)
17. Klijn, E.L., Fahland, D.: Performance mining for batch processing using the performance spectrum. In: Business Process Management Workshops. Lecture Notes in Business Information Processing, vol. 362, pp. 172–185. Springer (2019)
18. de León, H.P., Rodríguez, C., Carmona, J., Heljanko, K., Haar, S.: Unfolding-based process discovery. CoRR **abs/1507.02744** (2015)
19. Li, G., de Murillas, E.G.L., de Carvalho, R.M., van der Aalst, W.M.P.: Extracting object-centric event logs to support process mining on databases. In: CAiSE Forum. Lecture Notes in Business Information Processing, vol. 317, pp. 182–199. Springer (2018)
20. Lu, X., Fahland, D., Andrews, R., Suriadi, S., Wynn, M.T., ter Hofstede, A.H.M., van der Aalst, W.M.P.: Semi-supervised log pattern detection and exploration using event concurrence and contextual information. In: OTM Conferences (1). Lecture Notes in Computer Science, vol. 10573, pp. 154–174. Springer (2017)
21. Lu, X., Nagelkerke, M., van de Wiel, D., Fahland, D.: Discovering interacting artifacts from ERP systems. IEEE Trans. Serv. Comput. **8**(6), 861–873 (2015)
22. Martin, N., Solti, A., Mendling, J., Depaire, B., Caris, A.: Mining batch activation rules from event logs. IEEE Transactions on Services Computing (2019)
23. Martin, N., Swennen, M., Depaire, B., Jans, M., Caris, A., Vanhoof, K.: Retrieving batch organisation of work insights from event logs. Decis. Support Syst. **100**, 119–128 (2017)
24. de Murillas, E.G.L., Reijers, H.A., van der Aalst, W.M.P.: Case notion discovery and recommendation: automated event log building on databases. Knowl. Inf. Syst. **62**(7), 2539–2575 (2020)
25. Pufahl, L., Weske, M.: Batch activity: enhancing business process modeling and enactment with batch processing. Comput. **101**(12), 1909–1933 (2019)
26. Weber, I., Farshchi, M., Mendling, J., Schneider, J.: Mining processes with multi-instantiation. In: SAC. pp. 1231–1237. ACM (2015)

Process "Prospecting" to Improve Renewable Energy Interconnection Queues: A Case Study

Gerry Murphy[1]

[1] PGMoDE, LLC, Boulder, CO 80302, USA
gerry.murphy@pgmode.om

Abstract. Globally, interconnecting a new solar or wind generation project to the grid involves navigating a queue requiring financial deposits, engineering studies, and fees to upgrade the electric grid. The process can take years, during which time changes to regulatory regimes, tax incentives, financial markets, or competitive pressures can make a project suddenly nonviable for an investor. For grid operators, the increasing saturation of intermittent generation concurrent with retiring fossil fuel generation makes every new project increasingly complex to assess. This paper provides a case study of applying process mining techniques to address the question of whether the options proposed by Duke Energy Carolinas (DEC) to reform its generation interconnection queue process are warranted. Two options for reform have been proposed: creating study clusters based on concurrency or creating them based on locational proximity. Results indicate support for aspects of both options, although some causes may prove uncontrollable due to their origin in external factors such as market competition and power systems engineering decision making.

Keywords: Interconnection queue, process discovery, conformance analysis.

1 Background

This paper provides a case study of applying process mining techniques to "prospect" options for reforming generation interconnection queue procedures operated by Duke Energy Carolinas (DEC), an electric utility in the United States with 2.6M customers[1]. *Interconnection queue* refers to the process for new power plants to get approval for connecting to the electric grid. DEC interconnection workflows consists of the following steps: 1.) application and review, 2.) system impact study, 3.) potential restudy loops as needed including feasibility and facilities studies, 5.) interconnection agreement, 6.) construction, 7.) commissioning, and 8.) commercial operations.

Driven by rapid equipment cost declines, government incentives, and a favorable economic environment, the quantity of solar generation capacity installed in North Carolina (NC) grew from ~1,000 MW of installed capacity in 2015 to 6,435 MW of installed capacity as of Q1 2020[2,3]. This $9B cumulative investment has made NC the second ranked state in the United States for solar generation capacity[3].

Investment has been influenced by periodic expiration of the Federal Production Tax Credit (PTC). Created in 1992, PTC has been renewed 12 times since expiring in

1999[4]. In early 2016, PTC was extended until 12/31/2019. This created unprecedented market stability for solar project developers. In late 2019, PTC was extended only through 12/31/2020[4]. For this study, 12/31/2015 and 12/31/2019 are key dates.

Investors risk developing and operating solar generation plants in exchange for selling their electricity to an electric utility at a fixed price that guarantees a positive return on investment. Because these investments are speculative in nature until final stages, economic and competitive forces can have a large effect on investor behavior.

In 2019, North Carolina Utilities Commission (NCUC), which regulates the electricity market in NC, required DEC to expedite the interconnection queue[5]. DEC proposed two approaches: clustering new solar projects based on a temporal basis versus doing so on a locational basis. The proposed changes would involve the same activity sequence, but activities would be coordinated across multiple projects. Grid upgrade costs would be shared across multiple projects as opposed to having the single project that triggers an upgrade bearing the full upgrade cost[6].

Quarterly DEC regulatory filings were used as source data for an event log of interconnection milestones for new solar projects. Due to their legal nature, the filings were assumed accurate. Filings had quarterly intervals so trace alignment would not have tied to daily workflow activities; a standard approach to identify factors driving activity bottlenecks was not viable. Summarized below, a "prospecting" approach was taken to compare process performance of multiple study groups filtered within a single data set.

Table 1. The process "prospecting" approach taken for this study.

Research Question	Data Limitation	"Prospecting" Adaptations
Is there evidence to support temporally grouped cluster studies? • H_a: there is seasonal variation in project activities • H_o: there is no seasonal variation	Data describe queue performance in quarterly snapshots but do not capture daily activities.	• Define two seasonal study groups of cases based on their queue entry date • Compare Petri net behavior and event log conformance for the seasonal groups
Is there evidence to support locationally grouped cluster studies? • H_a: locational clustering can be seen in a higher count of projects per substation where interconnection occurs • H_o: there is no locational clustering	Data lack project developer activities that describe how they select locations or how they manage projects through the queue.	• Define a third study group of cases located at top quartile substations in terms of project volume. • Compare Petri net behavior and event log conformance for top quartile substation projects vs. seasonal groups
Is there evidence to explain how developers navigated the key date effect on their investments? • H_a: PTC expiration dates influence activities • H_o: PTC expiration dates do not influence activities	The effect of external factors on queue dynamics was not directly measured.	• Define a fourth study group of top quartile installers based on their number of cases • Compare project cancellations or sales in key date years vs. other years

2 Methodology

2.1 Gather DEC regulatory filing data and convert it into MS Excel format

An event log was assembled by collecting documents provided by DEC to NCUC on a quarterly basis from Oct 2015 – Apr 2020[7]. The documents were converted to a spreadsheet, standardized, and prepared for process mining in a relational database.

The original data set contained 18,560 events for 4,868 cases. It included these attributes: Queue Number, Queue Issued Date, Installer Account Name (5,424 null entries), Energy Source Type (1 nulls), Installed Capacity (no nulls), Facility County (919 nulls), Substation Name (752 nulls), and Feeder Number (1,331 nulls). The Installed Capacity attribute was removed due to unit of measure variations. The following activity types were discovered: Additional Field Work Required, Cancelled, Construction - In Progress, Construction – Pending, Engineering Design - In Progress, Engineering Design – Pending, Facility Study - In Progress, Facility Study – Pending, Feasibility Study – Pending, Interconnection Agreement Execution – Pending, IR Review - In Progress, Open, Request Incomplete, and Superseded.

2.2 Assess Process Performance and generate an event log CSV file

Using SQL queries, start and completion times were calculated for project activities. Completion times were not given, so they were imputed by determining the quarterly filing report date in which that activity or project disappeared: for example, if a project was listed as "construction – in progress" in one report but then the project was no longer listed in the subsequent report, construction was assumed to have completed and thus given a completion date of the report when it first disappeared. This reduced total events from 18,560 to 6,659 as events without date information were removed.

Activity types reported by DEC changed over time. Newer activity types that appeared from 2018 were filtered because they had low occurrence and incomplete data. Only these events were analyzed: "Open," "Cancelled," "Superseded," "Construction – In Progress," and "Construction – Pending." Total events reduced from 6,659 to 6,456. A CSV file was created in event log structure and exported.

2.3 Conduct Petri net behavior and event log conformance analysis

The CSV file was uploaded to PROM 6.9, converted to XES format, and filtered into the following groups: a.) projects initiated between July 1 and December 31 ("Ones") b.) projects initiated between January 1 and June 30 ("Zeros"), and c.) projects initiated for interconnection at a substation within the top quartile of interconnection requests ("TopQS"). Filter groups a.) and b.) addressed the temporal clustering question. Filter group .c) addressed the locational clustering question. The sequence of analysis was as follows: 1.) Petri net analysis, 2.) Conformance analysis using Multi-perspective Process Explorer and Replay A Log On Petri Net for Conformance packages.

Using PROM 6.9, Petri nets were created using Mine Petri Net With Inductive Miner package utilizing default settings. Inductive Miner was chosen because it produced a Petri net model with sequential activities most resembling the actual DEC

interconnection process, unlike Alpha Miner and ILP packages. A key assumption was that Inductive Minor can correctly identify the main process behavior in this event log; all deviations identified in conformance analysis are true process deviations.

2.4 Conduct event log visualization and directly follows graph analysis

Log analysis was conducted using Explore Event Log (Track Variants), Log Pattern Explorer, and Dotted Chart visualizations. Lastly, Mine Matrix package was run to generate event causality data for comparison.

2.5 Analyze key date behavior of project developers (*installers*)

In its regulatory filing data, DEC uses the term *installer* to refer to project developers. To assess their key date behavior a fourth study group was created for *installers* within the top quartile of solar project volume – TopQI. The project events of TopQI were compared to the remaining 75% of *installers* (Rest), focusing on events surrounding key legislative dates: 12/31/15 and 12/31/19. A key assumption was that larger developers have more engineers and resources compared to smaller *installers*. Consequently, TopQI were expected to be savvier in their responses to key dates.

3 Results

3.1 Assess Process Performance

Figure 1 below is a forward-looking chart created in MS Excel that counts projects in the queue on a quarterly basis in terms of what their future end state will become.

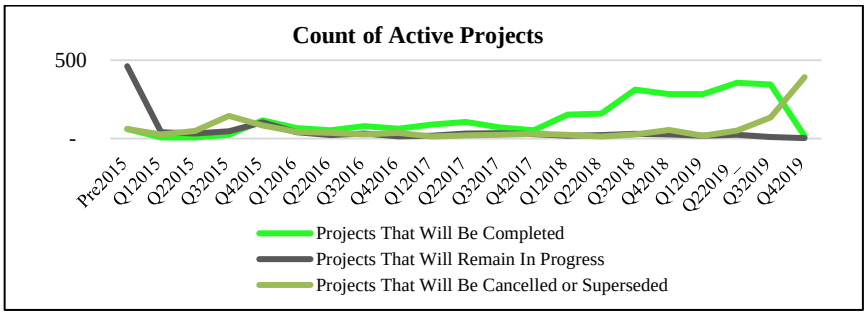


Fig. 1. Quarterly count of active projects by future disposition in the DEC interconnection queue.

DEC performance in processing interconnection requests improved after 2017. If you were a developer entering the queue in August 2018, you would have had 3X the likelihood of completing your project as someone entering just 10 months before. Above, the 12/31/15 key date effect is visible in the precipitous decline of projects that will remain in progress following Q1 2015. Of this cohort of 527 projects, 20% were cancelled in Q3 2015, 64% went on to be cancelled or sold ("superseded") in 2016 -

2019, 11% were in pre-construction state as of April, 2020, and only 6% passed onto the construction stage. The 12/31/19 key date effect is again visible above in the increase in projects that will be canceled or superseded as of Q4 2019.

Calculated in MS Excel, Figures 2 and 3 below show the average and standard deviation of project duration for projects in the DEC interconnection queue.

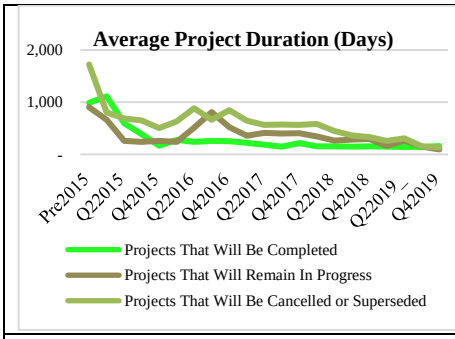


Figure 2. Quarterly average duration (days) of projects in DEC interconnection queue.

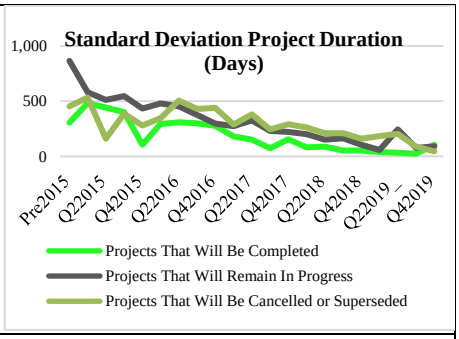


Figure 3. Quarterly standard deviation duration (days) of projects in DEC interconnection queue.

For projects that will be completed, the average project duration for projects declined from 1,112 days in Q1 2015 to 161.93 days in Q4 2019. Standard deviation of for projects that will be completed declined from 482 days in Q1 2015 to 107 days in Q4 2019.

Perhaps the best metric of interconnection queue performance success is whether a project gets constructed. Figure 4 below gives an overview of project success trends:

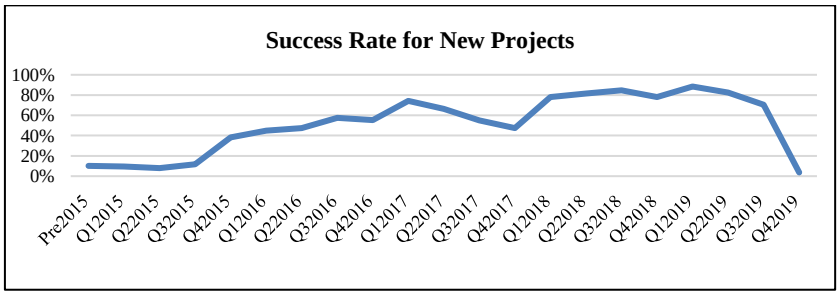


Fig. 4. Success rate for new projects initiated each quarter: the count of projects that eventually start construction / count of all projects initiated.

Figure 4 provides further evidence of improving queue performance between 2017 and 2019. Yet as the key date of 12/31/2019 approached, construction starts fell.

Table 2 below is a summary of queue performance across the three study groups:

Table 2. 2015 – 2020 interconnection queue performance for Zeros, Ones, and TopQS groups.

Filter Group	Total Projects Initiated	Avg Elapsed Time/Case (Days)	Success Rate
Zeros: Q1 - Q2 start	1,671	167.41	89%
Ones: Q3 - Q4 start	1,838	154.51	80%
TopQS	890	161.25	88%

Between July and December, Ones group initiated more projects (1,838) vs. Zeros group from January to June (1,671). Ones group projects had shorter elapsed time compared to Zeros group but had a lower success rate. TopQS located projects had lower elapsed time compared to Zero group while having a similar success rate to Ones group.

3.2 Petri Net Behavioral Analysis

Petri nets for the Zeros, Ones, and TopQS groups were created using the Mine Petri Net Using Inductive Miner package and assessed using Analyze Behavioral Property of Petri Net and Analyze Structural Property of Petri Net packages. Results were compared using the Show Petri Net Metrics package. Table 3 below summarizes results.

Table 3. Petri analysis results for Zeros, Ones, and TopQS groups.

	Ones	Zeros	TopQS	All
Density metric	0.08824	0.08824	0.06875	0.16667
|F|	36	36	44	22
|P X T|	204	204	230	66
Extended Cardoso metric	15	17	20	9
Extended Cyclomatic metric	26	22	35	9
Number of arcs	36	36	44	22
Number of places	12	12	16	6
Number of transitions	17	17	20	11
Structuredness metric	66	100	117	22

Every Petri net was found to be a sound workflow net. All three groups had Extended Cardoso values of 14 - 20, which per Cardoso places them in the "easy to understand" complexity category[8]. Extended Cyclomatic metrics show wider variation than Extended Cardoso metrics: there is a wider difference in the number of possible linear paths across the three groups compared to the number/type of splits. The Structuredness metric results align more with Extended Cardoso results: TopQS has the most complex model, followed by Zeros and then Ones. The three filtered study groups each have greater workflow complexity than the event log as a whole (All).

3.3 Conformance Analysis

Conformance analysis was conducted on each filter group and the results were compared side by side. The results of Multi-perspective Process Explorer and Replay A Log On Petri Net are shown in Table 4 below.

Table 4. Conformance analysis results for Zeros, Ones, and TopQS groups.

	Ones	Zeros	TopQS	All
Avg activity precision	79.2%	86.4%	72.6%	94.7%
# Moves Observed	29,897	27,488	15,089	32,342
# Moves Possible	37,726	31,806	20,776	34,143
Avg Fitness	63.7%	58.6%	58.2%	68.2%
% Violations	33.4%	37.9%	39.3%	41.8%
# Correct Events	3,983	3,946	1,619	6,973
# Wrong Events	1,895	2,160	975	5,011
# Missing Events	98	251	75	-
# Traces	2,408	2,548	1,130	4,671
# Events	5,878	6,106	2,594	11,984
# Event Classes	5	5	5	5

The Ones group had higher average fitness compared to Zeros and TopQS. All other conformance metrics were lower for Zeros versus Ones. Combined with the lower Structuredness and Extended Cardoso metrics for Ones in Table 3 above, projects initiated from July to December (Ones) have better performing models than projects initiated from January to June (Zeros). TopQS group covers the entire year with a locational focus, and it has the lowest overall precision and fitness. Despite its successful balance of low case duration, high success rate, and lowest raw fitness cost, the TopQS model has more violations and fewer correct events than Ones and Zeros.

3.4 Event Log Visualizations

The figures below summarize log visualizations for Zeros, Ones, and TopQS groups:

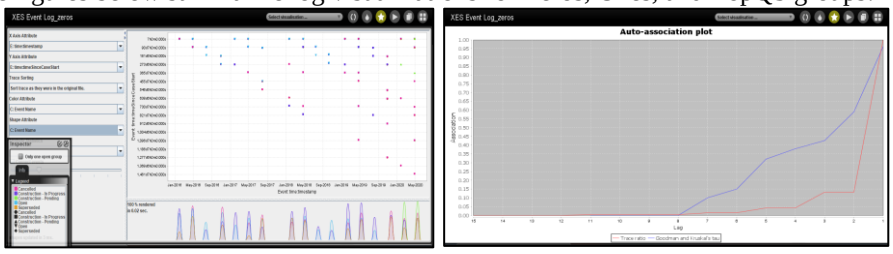


Fig. 5. Zeros group Dotted Chart and Auto-association visualizations.

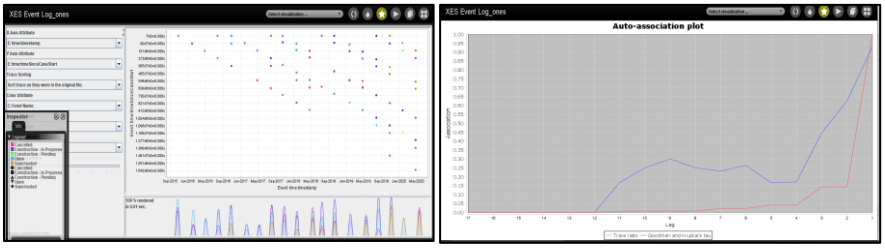


Fig. 6. Ones group Dotted Chart and Auto-association visualizations.

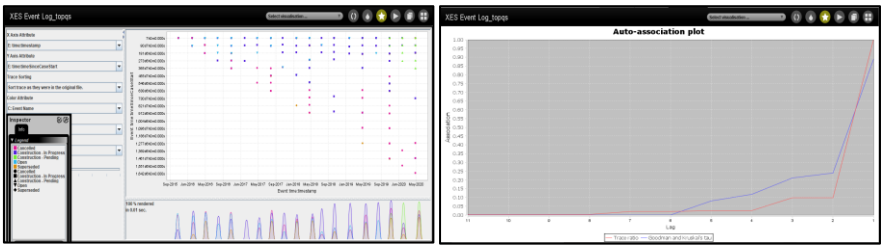


Fig. 7 TopQS group Dotted Chart and Auto-association visualizations.

The Dotted Charts for Zeros and Ones groups are similar. In the upper right corner of the chart, which displays recent, lowest duration events, Ones have better performance than Zeros. For Ones, this corner is denser with events and there are more blue "Construction – In Progress" and green "Construction – Pending" dots that indicate success. Ones also have more tan "Superseded" dots and fewer pink "Cancelled" dots compared to Zeros. TopQS shows strongest overall performance in the dotted chart upper right corner: it is densest and has highest number of blue "Construction – In Progress" and green "Construction – Pending" dots.

Auto-association plots vary most in terms of the Goodman and Kruskal's tau values. For Ones, association values decline overall in the lag range of 9 down to 4 before increasing again. Zeros and TopQS do not have this mid-range decline. All three groups rapidly increase their association values at the lowest lag values.

3.5 Directly Follows Graph

Convert Log to Directly Follows Graph package was next run on each group to compare high-level views of the queue process. Figures 8-10 below show the resulting graphs.

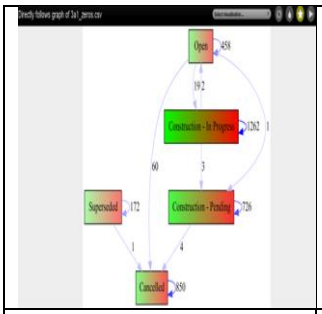


Fig. 8. Directly follows graph of the Zeros group

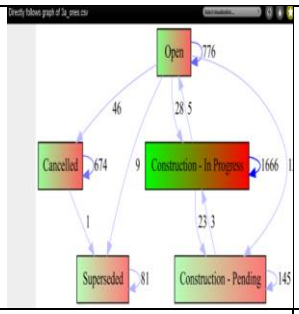


Fig. 9. Directly follows graph of the Ones group

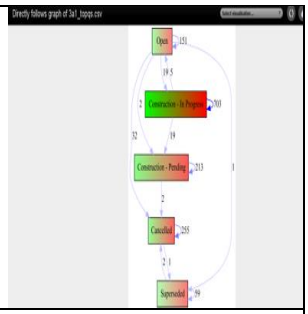


Fig. 10. Directly follows graph of the TopQS group

Comparing Figures 8-10 to the interconnection queue in Figure 1 above, the directly follows graphs differ in the flow of events through the "Construction – Pending" activity. The DEC proposed process shows a series of engineering studies of a new power plant and the milestone payments that lead to a facilities study and then an interconnection agreement prior to construction start. In Figures 8-10, a project can proceed to "Construction – Pending" directly from the initial "Open" activity, but it is most

common for a project to attain "Construction – In Progress" prior to "Construction – Pending." Ones and Zeros differ in the path to "Cancelled" activity, which in the case of Zeros can occur directly following the "Construction – Pending" activity. TopQS shows the most hierarchical flow of events, having both "Cancelled" and "Superseded" both directly following "Construction – Pending" activity.

3.6 Key Date Behavioral Analysis: Solar Project Developers

Key date behavioral analysis focused on comparing *installers* (developers) accounting for the top quartile of solar project volume (TopQI) group to the rest of *installers* (Rest). Comparing the October to January period for key dates of 12/31/15 and 12/31/19, when PTC was set to expire, versus 12/31/2016, 12/31/17, and 12/31/2018, a sharper picture of queue dynamics emerges. Table 5 below shows this comparison.

Table 5. Comparison of key project activities that start between October to January for PTC expiration dates (2015, 2019) versus other years (2016, 2017, 2018).

Time Period	Cancelled	Construction - In Progress	Construction - Pending	Queue Issued	Superseded
PTC Expiration 2015	134	0	0	304	0
PTC Expiration 2019	166	0	0	501	47
2016, 2017, 2018	192	296	49	948	60

PTC expiration years of 2015 and 2019 had a disproportionate number of projects cancelled or superseded (347 = 134+166+0+47) versus the other years (252=192+60). No projects were committed to construction in 2015 or 2019. More projects were initiated in the 2019 end of year period (501) versus 2015 (304), although this was far below 2016 – 2018 levels (948).

The tables below compare top quartile installers (TopQI) to the Rest of installers.

Table 6. Comparison of key project activities of TopQI group during end of year time intervals.

Time Period	Cancelled	Construction - In Progress	Construction - Pending	Queue Issued	Superseded
PTC Expiration 2015	77	0	0	218	0
PTC Expiration 2019	72	0	0	8	23
2016, 2017, 2018	103	58	13	306	21

Table 7. Comparison of key project activities of Rest group during end of year time intervals.

Time Period	Cancelled	Construction - In Progress	Construction - Pending	Queue Issued	Superseded
PTC Expiration 2015	57	0	0	86	0
PTC Expiration 2019	94	0	0	493	24
2016, 2017, 2018	89	238	36	642	39

For *Cancelled* projects, TopQI did not change practices from 2015 to 2019 key dates (77 vs. 72), but the Rest increased cancellations by 65% in 2019 (57 vs. 94). For project starts (*Queue Issued*), TopQI decreased theirs by 96% from 2015 to 2019 key dates (218 vs. 8) and the Rest increased *Queue Issued* by 83% (86 vs. 493). For *Superseded*, TopQI and the Rest both increased this activity in 2019 (0 vs. 23 and 24, respectively).

4 Discussion

4.1 Petri Net Behavioral Analysis

Variation in complexity metrics observed across groups and scenarios based on the same business workflow could point to both anomalies in the data set and opportunities to streamline the workflow and standardize its data model, simply from the perspective of reducing errors. Since all groups derive from the same event log which has been simplified by removing low occurrence events, these complexity differences could reflect real variances in the process. Additionally, better Petri net performance observed in the Ones model could explain its advantage in average case time assuming poor process performance is reflected in project delays. However, the observations rely on Inductive Minor's ability to correctly portray the main process model in Petri net outputs.

4.2 Conformance and Event Log Analysis

Seasonality is a factor in model performance. There is seasonal variance in case duration, in the number of events and traces per case, and in model conformance.

Assuming Inductive Minor did accurately capture the main process model in its generated Petri nets so that variance across study groups reflects real process variation, Zeros had worse model performance than Ones but also had a higher success rate. Projects initiated from July to December (Ones) had more low-occurrence events that were removed during data preparation and this could have advantaged its model performance compared to projects that initiated from January to June (Zeros). Having a lower project workload did not make Zeros group interconnection requests get processed faster than Ones group. It is possible that Zeros having extra time helped resolve issues blocking construction starts, but the key date effect could just as likely have increased cancellations for Ones during the end of the year. The higher share of project cancellations and

supersessions (sales) during the end of year period could have been a factor on Ones having a lower success rate versus Zeros. Despite having the lowest density model, TopQS had the lowest model fitness and precision. The simpler model for TopQS did not advantage its model conformance. From a model conformance perspective, it is not clear what drove TopQS success.

Dotted Chart results of Figures 5 – 7 above show differentiation across groups in shortest duration events occurring in the most recent time intervals. In this upper right quadrant area, Ones have higher success versus Zeros and TopQS have highest success overall. Relative success and short duration of TopQS projects could be a recent trend.

Overall, if Inductive Minor did not accurately portray the main process flows then the validity of conformance analysis results is questionable. However, the study group differentiation observed in Dotted Chart results could support the differentiation observed in conformance analysis.

4.3 Key Date Behavioral Analysis

End of year project activities differed between the TopQI and Rest of installers group. Comparing TopQI to the Rest, there is a relative increase in Q4 cancellations across all years, but not on the specific key dates of 2015 and 2019. TopQI also showed a relative increase in superseded projects in Q4 and on key dates compared to the Rest. For project initiations, TopQI had a greater volume in Q4 across all years including key dates compared to the Rest, but the volume of project initiations was still lower during this time of year. Overall, it appears that TopQI were reducing DEC area investment at the end of 2019 while the Rest were still ramping up DEC area investment. TopQI were aggressive investors in 2015 but cautious or exiting the DEC area in 2019.

4.4 Approach Viability

Converting regulatory filings into an event log for process "prospecting" analysis was a novel approach. Because the main questions in the queue reform debate had already been framed within publicly available documents, process "prospecting" was able to provide valuable context despite limitations of the data set.

Filtering study groups from a common data source met the objective of finding process variations relative across the groups. Going further to benchmark this interconnection queue data set against that for queues in other regions would be a more challenging topic which would require more robust data.

Addressing more detailed questions about root cause would have required alignment analysis at the trace level. Alignment analysis was not viable because the end dates of many activities were imputed on a quarterly basis, which may have created concurrency that did not really exist.

5 Conclusions

Based on results, the null hypotheses for seasonal and locational cluster patterns can be rejected. Neither the process performance nor process "prospecting" results indicate

seasonal or locational uniformity in the DEC interconnection queue. In terms of project developer behavior, the null hypothesis that PTC expiration dates do not influence project events can also be rejected. Addressing the influence of key dates on project events will be an important consideration to finalizing the proposed DEC cluster study process.

Process "prospecting" played a useful role in addressing whether the proposed queue clustering approaches were warranted. Its insights into model performance could be useful to design the optimal cluster study workflow. As a followup, process mining on a more robust DEC interconnection queue data set could determine whether the process variances observed by Inductive Minor are accurate. In addition to supporting the business and regulatory mandate for queue reform at DEC, this could help improve the design of the cluster study workflow.

More broadly, this study confirmed that process mining can be incorporated to benefit process-focused business scenario analysis. PROM 6.9 offered a vast array of analytical options, which was advantageous. Since it is a research tool, the downside of PROM 6.9 is that each plugin has its own documentation and accompanying research papers. This caused some confusion around interpreting process mining results.

From the industry perspective, process "prospecting" is a common scenario; businesses are likely to begin a process transformation initiative with a small pilot and limited data. There is an opportunity in the process mining community to craft a "prospecting" interface that allows practitioners to assess their data and recommend plugin options that meet their study objectives. Long term, meta-research studies across the suite of PROM 6.9 plugins may be useful to cultivate the process mining body of knowledge.

6 References

1. Duke Energy website: https://www.duke-energy.com/our-company/about-us, accessed 07/27/2020
2. US Energy Information Agency website: https://www.eia.gov/state/?sid=NC, accessed 07/27/2020
3. Solar Energy International website: https://seia.org/state-solar-policy/north-carolina-solar, accessed 07/27/2020
4. Congressional Research Service: https://crsreports.congress.gov/product/pdf/R/R43453/17, accessed 07/27/2020
5. NCUC website: https://starw1.ncuc.net/NCUC/PSC/PSCDocumentDetailsPageNCUC.aspx?DocumentId=6d82de14-71de-492f-a6d4-f9a14099642a&Class=Order, accessed 07/27/2020
6. NCUC website, "DEC and DEP Queue Reform Update," October 15, 2019: https://starw1.ncuc.net/NCUC/ViewFile.aspx?Id=0ca6866e-dcaf-4622-b543-486ab37cc34a, accessed 07/27/2020
7. NCUC website, Document Search: https://starw1.ncuc.net/NCUC/page/DocumentsTextSearch/portal.aspx. Search term: "Interconnection Queue", accessed 07/27/2020
8. L. Sanchez-Gonzalez et al.: "Towards Thresholds of Control Flow Complexity Measures for BPMN Models," (2011) https://jorge-cardoso.github.io/publications/Papers/CP-2011-060-SAC-Towards-thresholds-of-control-flow.pdf, accessed 07/27/2020

Automated Discovery of Process Models with True Concurrency and Inclusive Choices

Adriano Augusto[1], Marlon Dumas[2], and Marcello La Rosa[1]

[1] University of Melbourne, Australia
{a.augusto, marcello.larosa}@unimelb.edu.au
[2] University of Tartu, Estonia
marlon.dumas@ut.ee

Abstract. Enterprise information systems allow companies to maintain detailed records of their business process executions. These records can be extracted in the form of event logs, which capture the execution of activities across multiple instances of a business process. Event logs may be used to analyze business processes at a fine level of detail using process mining techniques. Among other things, process mining techniques allow us to discover a process model from an event log – an operation known as automated process discovery. Despite a rich body of research in the field, existing automated process discovery techniques do not fully capture the concurrency inherent in a business process. Specifically, the bulk of these techniques treat two activities A and B as concurrent if sometimes A completes before B and other times B completes before A. Typically though, activities in a business process are executed in a true concurrency setting, meaning that two or more activity executions overlap temporally. This paper addresses this gap by presenting a refined version of an automated process discovery technique, namely Split Miner, that discovers true concurrency relations from event logs containing start and end timestamps for each activity. The proposed technique is also able to differentiate between exclusive and inclusive choices. We evaluate the proposed technique relative to existing baselines using 11 real-life logs drawn from different industries.

1 Introduction

Enterprise information systems, such as Enterprise Resource Planning (ERP) systems, maintain detailed records of each execution of the business processes they support. These records can be extracted in the form of event logs. An event log is a set of event records capturing the execution of activities across a set of instances of a process.

Process mining techniques allow us to exploit event logs in order to analyze business processes at a fine level of detail. Among other things, process mining techniques allow us to discover a process model from an event log – an operation known as *automated process discovery*. Despite a rich body of research in the field, existing automated process discovery techniques do not fully capture the concurrency inherent in business processes. Indeed, the bulk of automated process discovery techniques operate under an interleaved concurrency model – a model of concurrency where two events are concurrent if they occur in either order. Specifically, existing techniques treat two activities A and B as concurrent if sometimes A completes before B and other times B completes

before A. The interleaved concurrency model is suitable in systems where actions are atomic. However, in a business process, activities have a duration and the execution of two or more activities may overlap temporally. In other words, business processes contain *true concurrency*. The failure of existing automated process discovery techniques to take into account this true concurrency leads them to miss certain concurrency relations. For example, when an activity A always completes before activity B (because B takes longer) even though A and B overlap, existing techniques treat A and B as sequential. If A is then followed by C and C usually completes after B (but overlaps with it), they conclude that A, B and C are sequential, thus missing the observed concurrency.

This paper addresses this gap by presenting a refined version of an automated process discovery algorithm, namely Split Miner [6], capable of discovering true concurrency relations from event logs that record both the start and end timestamps of activity executions. The proposed technique, namely Split Miner 2.0, is also able to differentiate between exclusive and inclusive choices. The paper reports on an empirical evaluation that compares Split Miner 2.0 against existing baselines in terms of accuracy and model complexity measures.

The rest of the paper is structured as follows. Section 2 briefly reviews existing automated process discovery techniques. Section 3 introduces the approach to exploit true concurrency for automated process discovery. Section 4 presents the empirical evaluation while Section 5 summarizes the findings and further possible extensions.

2 Background and Related Work

An *event log* records information about a set of executions of a business processes (a.k.a. cases). Concretely, an event log is a chronological sequence of events, each one capturing a state change in the execution of an activity. As a minimum, each event in a log has three attributes: the identifier of the process execution (a.k.a. *case ID*); the label (i.e. the process activity the event refers to); and the timestamp (e.g. 10/07/2020 10.43). Optionally, an event may have other attributes such as the resource who triggered the event, their department, etc. In this paper, we require that at least one fourth attribute is attached to each event, namely the *life-cycle transition*. For a given event, this attribute indicates what state-change the referenced activity has undergone. The life-cycle of an activity captures all the states in an activity execution and their possible transitions. In general, one could observe very complex life-cycles, including states such as created, assigned, started, suspended, etc. In this paper, we adopt a simple life-cycle model wherein an activity execution can be in one of two states: *start* (i.e. the activity execution started); and *end* (i.e the activity execution ended).

Event logs can be exploited for different types of analysis including conformance checking, process performance mining, and automated process discovery[16]. In this paper, we focus on the latter. The goal of automated process discovery is to discover a process model (such as the one in Figure 1) by analysing an event log such as the one in Table 1 (the latter is just an extract and not a full log).

The quality of an automatically discovered process model is traditionally assessed over four dimensions: fitness – the amount of process behaviour recorded in the event log that can be replayed by the process model; precision – the amount of behaviour captured by the process model that can be found in the event log; generalization – the

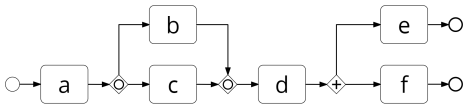


Fig. 1: Process model example.

Case-ID	Activity	Life-cycle	Timestamp
1	a	start	2020-07-08 10.03
2	a	start	2020-07-08 10.42
1	a	end	2020-07-08 10.57
2	a	end	2020-07-08 11.21
1	b	start	2020-07-08 13.29
1	c	start	2020-07-08 14.13
2	b	start	2020-07-08 15.22
2	b	end	2020-07-09 10.24
1	b	end	2020-07-09 10.37
2	d	start	2020-07-09 11.13
2	d	end	2020-07-09 12.28
1	c	end	2020-07-09 12.53

Table 1: Event log example.

amount of behaviour captured by the process model that even not being observed in the event log is likely to belong to the original process; and simplicity – quantifying how difficult is to understand the process model. Furthermore, a process model should be sound. The notion of *soundness* has been defined on Workflow nets [17] as a correctness criterion, and is also applicable to BPMN models. Formulated on BPMN models, soundness encompasses three properties: i) every process instance eventually reaches the end event (no deadlocks); ii) no end event is reached more than once during a process execution (proper completion); iii) each process activity is triggered in at least one process execution (no dead activities).

A recent literature review of automated process discovery algorithms [5] showed that only few algorithms stand out for accuracy and performance among those outputting procedural process models. Specifically, Inductive Miner (IM) [10], Evolutionary Tree Miner (ETM) [7], and Split Miner (SM) [6]. IM and ETM are known to discover process models that are either highly fitting or precise, discovering simple, block-structured and sound process models, while SM focuses on maximizing both fitness and precision at the cost of simplicity, structuredness, and in rare cases compromising the soundness of the process models [5, 6]. However, of these three automated process discovery algorithms, only IM provides a variant that takes into account the activities' life-cycle when discovering a process model. IM life-cycle variant [11] analyses the activities' life-cycles to distinguish between concurrency and interleaving relations.

Past studies that investigated the problem of discovering control-flow relations between activities by leveraging life-cycle information or execution times include: (1) a simple algorithm [13] for discovering block-structured process models from complete and noise-free event logs; (2) an extension of the α-algorithm, i.e. the β algorithm [19]; (3) an extension of Heuristics Miner [8]; and (4) the work of Senderovich et al. which explores process performance modelling via temporal network representation [14]. The first one is limited to noise-free log. The second and third are based on underlying algorithms that produce unsound and inaccurate models when applied to real-life event logs, as shown in [5]. The fourth approach is not geared to discovering process models but rather targets the problem of performance mining.

In this paper, we extend the SM algorithm, which has been shown to produce accurate and (generally) sound process models over real-life logs. Figure 2 shows an overview of how SM discovers a process model from an event log. Given an input event log, SM operates over five steps: i) discover the directly-follows graph (DFG) and loops from the event log; ii) analyse the DFG for discovering concurrency rela-

tions; iii) filter the DFG by removing the infrequent behaviour; iv) discover the split gateways; v) discover the join gateways. Each step is a standalone operation based on tailored algorithms [6], such a modular approach allows the replacement of any step with alternative methods. In this paper, we show how we updated the first, second, and fifth steps to discover true concurrency and inclusive choices, and reduce the chances of producing unsound process models via heuristics.

Fig. 2: Overview of the Split Miner approach [6].

3 Approach

In this section, we describe how we redesigned the first two steps of the Split Miner original approach [6] and integrated in the last step two heuristics to repair models that are unsound due to improper completion and identify inclusive relations between activities, enabling the discovery of OR-splits.

3.1 Refined Directly-follows Graph Discovery

Given an event log, the first step performed by Split Miner is to sequentially read the events and build the directly-follows graph (DFG). Although this operation is straightforward, its output strictly depends on how the event log and the DFG are defined. Definitions 1, 2, and 3 capture the notion of DFG used in the original Split Miner.

Definition 1. *[Event Log as in [6]] Given a set of process activity labels $\mathscr{A}$, an* event log $\mathscr{L}$ *is a multiset of traces, where a trace $t \in \mathscr{L}$ is a sequence of activity labels $t = \langle a_1, a_2, \ldots, a_k \rangle$, with $a_i \in \mathscr{A}, 1 \leq i \leq k$. In addition, we use the notation $a \in \mathscr{A}$ to refer an activity a that belongs to a generic trace $t \in \mathscr{L}$.* [1]

Definition 2. *[Directly-Follows Relation as in [6]] Given an event log $\mathscr{L}$ and two process activities $a_x, a_y \in \mathscr{A}$, we say that activity a_y directly-follows activity a_x, with notation $a_x \rightsquigarrow a_y$, if and only if (iff)* $\exists \langle a_1, a_2, \ldots, a_k \rangle \in \mathscr{L} \mid a_i = a_x \wedge a_j = a_y \wedge j = i+1 \wedge 0 < i < n$.

Definition 3. *[Directly-Follows Graph as in [6]] Given an event log $\mathscr{L}$, its* Directly-Follows Graph (DFG) *is a directed graph $\mathscr{G} = (N, E)$, where N is the non-empty set of nodes, where each node represents a unique activity $a \in \mathscr{L}$ and there exists a bijective function $\lambda : N \mapsto \mathscr{A}$ such that $\lambda(n)$ retrieves the activity n refers to; and E is the set of edges capturing the directly-follows relations of the activities observed in $\mathscr{L}$, $E = \{(n,m) \in N \times N \mid \lambda(n) \rightsquigarrow \lambda(m)\}$.*

To capture the activities' lifecycle information, we refine the concept of event log.

Definition 4. *[Refined Event Log] Given a set of events $\mathscr{E}$, a* refined event log $\mathscr{L}_\rho$ *is a multiset of traces, where a trace $t \in \mathscr{L}_\rho$ is a sequence of events $t = \langle e_1, e_2, \ldots, e_k \rangle$, with $e_i \in \mathscr{E}, 1 \leq i \leq k$. Each event $e \in \mathscr{E}_\rho$ is a tuple $e = (l, p, t)$, where $l \in \mathscr{A}$ is the process activity the event refers to, retrieved with the notation e^l; $p \in \{start, end\}$ is the state of the life-cycle of activity l, retrieved with the notation e^p; and t is the timestamp of the event, retrieved with the notation e^t.*

[1] For simplicity, we use the term activity to refer to its label.

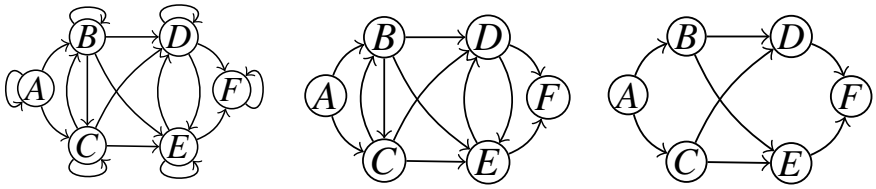
While redefining the event log to capture the activities' life-cycle information is intuitive and follows from its original definition [16], the same does not apply for the DFG. Indeed, more than one approach could be used to generate a DFG from a refined event log. The simplest approach would be to disregard all the events of a specific state of an activity life-cycle, for example, we could remove from $\mathscr{L}_\rho$ all the events $e \in \mathscr{L}_\rho \mid e^p = start$ or all the events $e \in \mathscr{L}_\rho \mid e^p = end$. Then, the refined event log would turn into an event log (Definition 1) and the DFG would be constructed according to Definition 3, but this would be equivalent to discarding the activities' lifecycle information.

An alternative approach was proposed by Leemans et al. [11] and incorporated into a variant of the Inductive Miner that takes into account lifecycle transitions, herein called Inductive Miner Lifecycle (IM-lc). According to [11], an activity a_y directly-follows an activity a_x if any of the life-cycle states of activity a_y is observed after any of the life-cycle states of activity a_x in the same trace and between the two observations no activity completes the execution of its full life-cycle (see Definition 5).

Definition 5. *[Directly-Follows Relation as in [11]] Given a refined event log $\mathscr{L}_\rho$ and two process activities $a_x, a_y \in \mathscr{A}$, the relation $a_x \rightsquigarrow a_y$ holds iff $\exists\ \langle e_1, e_2, \ldots, e_k \rangle \in \mathscr{L}_\rho \mid {e^l}_i = a_x \wedge {e^l}_j = a_y \wedge i < j \wedge \nexists n, m \in\ ]\, i, j\, [\ \mid\ n < m\ \wedge {e^p}_n = start \wedge {e^p}_m = end \wedge {e^l}_n = {e^l}_m$.*

According to Definition 5, a directly-follows relation would hold between two activities whose life-cycles overlap (i.e. the start-state of an activity is observed between the start-state and the end-state of another activity). While this is important and useful for IM-lc to discover concurrency relations [10], it would not be beneficial for Split Miner, since Split Miner requires to remove the directly-follows relations between activities that are considered concurrent [6]. Consequently, we are interested in discarding directly-follows relations of activities whose life-cycles overlap. We redefine the directly-follows relation of activities observed in a refined event log as follows. An activity a_y directly-follows an activity a_x if the start-state of the life-cycle of activity a_y is observed after the end-state of the life-cycle of activity a_x and no end-state of other activities are observed in between (see Definition 6).

Definition 6. *[Directly-Follows Relation] Given a refined event log $\mathscr{L}_\rho$ and two process activities $a_x, a_y \in \mathscr{A}$, the relation $a_x \rightsquigarrow_r a_y$ holds iff $\exists\ \langle e_1, e_2, \ldots, e_k \rangle \in \mathscr{L}_\rho \mid {e^l}_i = a_x \wedge {e^l}_j = a_y \wedge {e^p}_i = end \wedge {e^l}_j = start \wedge 1 \leq i < j \leq k \wedge \nexists n \in\ ]\, i, j\, [\ \mid {e^p}_n = end$.*



(a) Split Miner [6]. (b) Inductive Miner LC [11]. (c) Our approach.

Fig. 3: Examples of discovered DFGs by applying Definition 2, 5, and 6 (left to right).

The new version of Split Miner we propose in this paper relies on Definition 6. Depending on the definition of directly-follows relation that one adopts when generating the DFG, one may discover very different DFGs. As an example, let us consider the following refined event log (captured as a collection of traces, where each event is represented as the activity it refers to – including its life-cycle state as subscript, *s*

standing for *start* and *e* standing for *end*): $\mathscr{L}_{\rho_x} =$
$\{\langle A_s,A_e,B_s,C_s,C_e,B_e,E_s,D_s,D_e,E_e,F_s,F_e\rangle, \langle A_s,A_e,B_s,C_s,B_e,C_e,E_s,D_s,E_e,D_e,F_s,F_e\rangle,$
$\langle A_s,A_e,C_s,B_s,B_e,C_e,D_s,E_s,D_e,E_e,F_s,F_e\rangle, \langle A_s,A_e,C_s,B_s,C_e,B_e,D_s,E_s,E_e,D_e,F_s,F_e\rangle\}$;
Figure 3 shows the DFGs discovered from the $\mathscr{L}_{\rho_x}$ by applying Definition 2 (original Split Miner approach), Definition 5 (Inductive Miner life-cycle approach), and Definition 6 (this paper approach).

3.2 Refined Concurrency Discovery

The second step of the original Split Miner that we redesigned is the concurrency discovery. Split Miner relies on a simple heuristic to discover concurrency, precisely, given a DFG and two activities $A,B \in \mathscr{A}$ such that neither A nor B is a self-loop, A and B are assumed concurrent iff three conditions are true: A directly-follows B and B directly-follows A (Relation 1); A and B do not form a short-loop (Relations 2 and 3); the frequency of the two directly-follows relations $A \rightsquigarrow B$ and $B \rightsquigarrow A$ is similar (Relation 2).[2]

$$A \rightsquigarrow B \wedge B \rightsquigarrow A \tag{1}$$

$$\nexists \langle a_1,a_2,\ldots,a_k\rangle \in \mathscr{L} \mid a_i = A \wedge a_{i+1} = B \wedge a_{i+2} = A \ \wedge \ i \in [1,k-2] \tag{2}$$

$$\nexists \langle a_1,a_2,\ldots,a_k\rangle \in \mathscr{L} \mid a_i = B \wedge a_{i+1} = A \wedge a_{i+2} = B \ \wedge \ i \in [1,k-2] \tag{3}$$

$$\frac{||A \rightsquigarrow B| - |B \rightsquigarrow A||}{|A \rightsquigarrow B| + |B \rightsquigarrow A|} < \varepsilon \quad (\varepsilon \in [0,1]) \tag{4}$$

The simplicity of the concurrency oracle of Split Miner derives from the simplicity of the input event log (see Definition 1). However, when receiving as input a refined event log (Definition 4), it is possible to identify true concurrency by focusing on activities whose life-cycles overlap and are hence truly executed concurrently (e.g. by different process resources). Consequently, we redefine the concurrency discovery oracle as follows. Given two activities $A,B \in \mathscr{A}$ and a refined event log $\mathscr{L}_\rho$, we say A and B are concurrent if the following relation holds:

$$2 \cdot \frac{|A \asymp B|}{|A| + |B|} \geq \varepsilon \quad (\varepsilon \in [0,1]) \tag{5}$$

where $|A \asymp B|$ is the total number of observations of overlapping life-cycles of A and B in $\mathscr{L}_\rho$; $|A|$ and $|B|$ are respectively the total number of complete life-cycle[3] observations of activity A and activity B in $\mathscr{L}_\rho$; and ε is an arbitrary variable (given as input parameter) defining the minimum percentage of times that the two activities' life-cycles are required to overlap to assume the two activities concurrent. In particular, when $\varepsilon = 1$ our notion of concurrency is equivalent to the notion of *strong simultaneousness* defined by Van der Werf et al. [18] as well as Allen's interval relations [3] of *overlaps*, *contains*, *starts*, and *is finished by*. While for any other value of $\varepsilon > 0$ it is equivalent to a parametrized notion of *weak simultaneousness* [18]. Given that real-life event logs often contain noise and infrequent process behaviour, requiring $\varepsilon = 1$ would be very restrictive and may lead to the discovery of no concurrent activities.

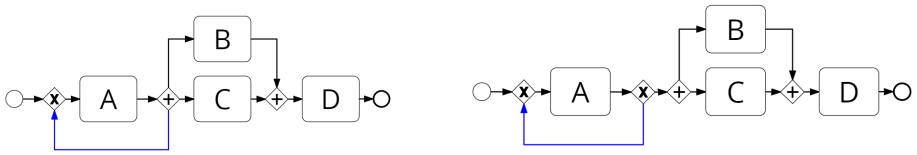
Although both our approach and IM-lc infer concurrency relations between activities from the observation of overlapping life-cycles, we rely on an heuristic before

[2] The frequency of a directly-follows relation is the number of times the relation is observed.
[3] E.g. including start and end states.

validating the concurrency relations (i.e. Equation 5) – in-line with the original Split Miner; while IM-lc assumes the information contained in the log to be valid a priori (this is mitigated by another extension of IM-lc that embeds a filtering technique [11]).

3.3 Heuristic Improvement

Although Split Miner guarantees to discover sound acyclic process models and *deadlock-free* cyclic process models with *no dead activities*, for cyclic process models it does not guarantee *proper completion*. However, it is possible to reduce the chances to discover process models exhibiting improper completion by applying the following heuristic: for each AND-split gateway in a process model with an outgoing edge that is a loop-edge (leading to a topologically deeper node of the process model), we create a preceding XOR-split gateway and set this latter as source of the loop-edge. Figure 4 intuitively show how the heuristic operates, the loop-edge is highlighted in blue and, in general, activities could be present in the loop-edge.



(a) Model with improper completion. (b) Model after applying heuristic.

Fig. 4: Heuristic removal of improper completion generated by loops.

Lastly, we integrated an heuristic to discern between concurrency and inclusive relations, in other words identifying when an AND-split gateway is a candidate OR-split gateway. This second heuristic operates as follows. For each AND-split gateway in a process model, we consider all the successor activities and we check pairwise whether there exist traces where the pair of activities are mutually exclusive (i.e. one of the two activities is executed but not the other). Then, if the majority of the pairs of activities are both mutually exclusive and concurrent in different traces,[4] we turn the AND-split gateway into an OR-split gateway and we update accordingly the OR-join gateway. As an example, let us consider the model in Figure 5a and the event log $\mathscr{L}_{\rho_y} = \{\langle A_s,A_e,B_s,C_s,D_s,B_e,D_e,C_e,E_s,E_e\rangle^3, \langle A_s,A_e,C_s,D_s,C_e,D_e,E_s,E_e\rangle^2,$ $\langle A_s,A_e,B_s,D_s,D_e,B_e,E_s,E_e\rangle, \}$; B and C are observed three times concurrently and three times are mutually exclusive, B and D are observed four times concurrently and two times mutually exclusive, C and D are observed five times concurrently and one mutually exclusive. Given that two pairs of activities out of three (B,D and B,C) are eligible for inclusiveness, we turn the AND gateways into OR gateways (Figure 5b).

4 Evaluation

In this section, we present an empirical evaluation that compares Split Miner 2.0 ($SM_{2.0}$) with three state-of-the-art automated process discovery algorithms: the original Split Miner [6] (SM), the Inductive Miner Lifecycle (IM-lc) [10] including its infrequent behaviour filter [11], and the most recent version of IM, namely IMfa [9].

[4] With at least one observation of mutual exclusiveness every two observations of concurrency or vice-versa.

(a) Before. (b) After.

Fig. 5: Heuristic identification of OR-split gateways.

4.1 Dataset and Setup

As testing dataset, we selected eleven real-life event logs (L1-L11) containing activity lifecycle information. The logs were sourced from companies operating in different fields (e.g. insurance, manufacturing, banking) and geographic areas (i.e. Europe and Australia). Given that these logs are not publicly available, we added a publicly available simulated event log known as the "Repair example" (R-Log),[5] which also contains activity lifecycle information. We did not include the BPIC12 and BPIC17 logs simply because the former does not have any overlapping lifecycle, and for the latter both SM and $SM_{2.0}$ produced the same model, which was analysed in previous studies [4, 6, 5].

Table 2 displays the characteristics of the event logs, highlighting their variety, with logs containing short to long traces (length 2 to 1,230), a wide range of distinct traces (0.28% to 96.55%) and distinct events (6 to 80), as well as notable differences in the total number of traces (37 to 70,512) and events (1,156 to 830,522). The lifecycle information for each activity recorded in these event logs was complete, i.e. the start and end events were recorded for each activity.

From each log, we discovered a process model with $SM_{2.0}$, SM, and IM-lc, and compared the quality of the discovered models over three quality measures: fitness, precision, and simplicity. Several methods have been proposed for measuring fitness and precision of an automatically discovered process model [15]. In this paper we use two methods, the one proposed by Adriansyah et al. [1, 2] (alignment-based accuracy) and the one proposed by Augusto et al. [4] (Markovian accuracy). As proxy for simplicity we use the following three metrics [12]: *Size* – the total number of nodes of a process model; *Control-flow complexity* (CFC) – the amount of branching induced by the split gateways in the process model; *Structuredness* – the percentage of nodes located inside a single-entry single-exit fragment of the process model.

We implemented $SM_{2.0}$ as a Java command-line application,[6] and we ran the experiments on an Intel Core i7-8565U@1.80GHz with 32GB RAM running Windows 10 Pro (64-bit) and JVM 8 with 14GB of allocated RAM (10GB Stack and 4GB Heap). All the discovery algorithms (SM, $SM_{2.0}$, and IM-lc) were executed using their default input parameters, and we set a timeout of 30 minutes for each algorithm execution and for each measurement.

[5] `http://www.promtools.org/prom6/downloads/example-logs.zip`
[6] Available as "Split Miner 2.0" at `http://apromore.org/platform/tools`

Event Log	Total Traces	Distinct Traces	Total Events	Distinct Events	Trace Length		
					MIN	AVG	MAX
L1	28,504	2.64%	443,862	23	4	15	1230
L2	3,885	9.11%	15,096	6	2	3	60
L3	954	10.80%	13,740	18	6	14	46
L4	37	86.49%	1,156	18	22	31	36
L5	146	78.08%	3,764	18	2	25	84
L6	551	96.55%	19,174	80	2	34	126
L7	70,512	0.28%	830,522	8	4	11	40
L8	9,906	2.19%	9,906	26	6	44	354
L9	1,182	92.81%	46,282	9	12	39	276
L10	608	11.51%	18,238	21	4	2	88
L11	1,214	20.18%	11,226	12	4	9	58
R-Log	1,104	5.53%	15,468	8	6	14	30

Table 2: Descriptive statistics of the logs.

4.2 Results

Table 3 reports the fitness, precision, and simplicity measurements. Due to space limits, the table does not show the measurements for IMfa because they were either equal or worse than those for IM-lc, with the exception of those obtained on L9 (which reported a slight improvement).

We can observe that $SM_{2.0}$ is less prone to discovering unsound models than SM, with the latter discovering an unsound model every three and the former only discovering sound models. This achievement reflects the effectiveness of our heuristics for removing improper completion.

In terms of accuracy, the results obtained with the alignment-based accuracy and the Markovian accuracy are partially consistent in line with previous findings [4]. In fact, the two measuring methods agree on the best models in terms of fitness, precision, and F-score, respectively 100%, 66%, and 75% of the times.

As for fitness, IM-lc outperforms both SM and $SM_{2.0}$ as expected [5]. In terms of precision and F-score $SM_{2.0}$ and SM achieve the highest scores, with $SM_{2.0}$ performing better than SM, most of the times discovering more precise and fitting process models ultimately achieving a higher F-score. In fact, $SM_{2.0}$ accuracy scores are either higher than or equal to those of SM, the latter outperforming the former in fitness or precision only two times according to the alignment-based accuracy, and only three times according to the Markovian accuracy. Compared to IM-lc, $SM_{2.0}$ discovers eleven times more precise process models, indipendently of the measurement method.

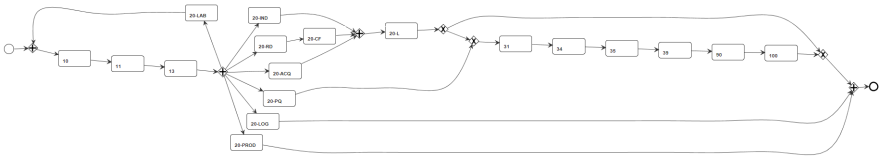
As for simplicity, $SM_{2.0}$ stands out by producing smaller models than those discovered by both SM and IM-lc (9 times out of 12) and with a lower CFC (10 times out of 12). However, $SM_{2.0}$ and SM cannot systematically produce fully-structured process models as opposed to IM-lc which achieves this by design. Lastly, the execution times of IM-lc, SM, and $SM_{2.0}$ are negligible: all the process models were discovered within a minute (except for log L7, where IM-lc timed out).

Figure 6 shows two qualitative examples of the improvements yielded by $SM_{2.0}$. Considering the models from the L6 log (Figures 6a and 6b), $SM_{2.0}$ discovered the inclusive-OR relations between several activities of the process and removed the improper completion, while SM produced an unsound model. In the specific case of the L6 log, we also had the chance to validate the discovered model with the process analysts of the organization this log was extracted from, who confirmed that the activities were indeed in an inclusive-OR relation. Considering the models from the R-log (Fig-

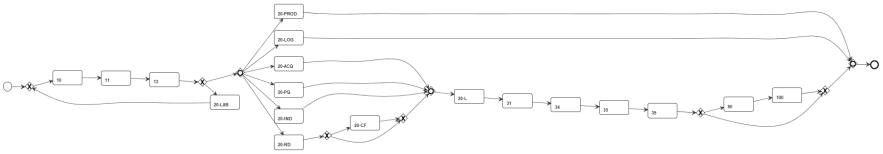
Event Log	Discovery Approach	Alignment Accuracy [1, 2]			Markovian Accuracy [4]			Simplicity		
		Fitness	Precision	F-score	Fitness	Precision	F-score	Size	CFC	Struct.
L1	IM-lc	0.88	0.78	0.83	0.82	0.15	0.25	**40**	26	**1.00**
	SM	**0.98**	0.94	**0.96**	**0.96**	**0.44**	**0.60**	47	32	0.47
	$SM_{2.0}$	0.83	**0.97**	0.90	0.44	0.34	0.38	45	**25**	0.56
L2	IM-lc	0.87	0.44	0.59	0.53	0.14	0.22	20	11	**1.00**
	SM	**0.92**	**1.00**	**0.96**	**0.69**	**0.88**	**0.77**	**14**	**6**	**1.00**
	$SM_{2.0}$	**0.92**	**1.00**	**0.96**	**0.69**	**0.88**	**0.77**	**14**	**6**	**1.00**
L3	IM-lc	**0.98**	0.71	0.82	**0.88**	0.08	0.14	49	27	**1.00**
	SM	0.96	0.97	**0.96**	0.72	**0.41**	**0.52**	36	16	0.58
	$SM_{2.0}$	0.93	**0.99**	**0.96**	0.40	0.07	0.12	**31**	**10**	0.77
L4	IM-lc	**0.98**	0.41	0.57	**1.00**	0.06	0.12	35	12	**1.00**
	SM	0.84	**1.00**	**0.91**	0.45	**0.79**	**0.57**	26	6	0.46
	$SM_{2.0}$	0.94	0.66	0.78	0.93	0.08	0.15	**25**	**3**	**1.00**
L5	IM-lc	**0.83**	0.53	0.65	**0.90**	0.17	0.29	33	12	**1.00**
	SM	*unsound*			*unsound*			31	11	0.45
	$SM_{2.0}$	0.76	**0.79**	**0.78**	0.86	**0.19**	**0.31**	**27**	**3**	0.59
L6	IM-lc	*measurements timeout*			0.10	0.00	0.01	**126**	**78**	**1.00**
	SM	*unsound*			*unsound*			161	98	0.14
	$SM_{2.0}$	**0.70**	**0.66**	**0.68**	**0.31**	**0.23**	**0.26**	138	80	0.50
L7	IM-lc	*discovery timeout*			*discovery timeout*			*discovery timeout*		
	SM	**0.88**	**1.00**	**0.94**	**0.73**	**0.90**	**0.81**	**12**	**2**	**1.00**
	$SM_{2.0}$	**0.88**	**1.00**	**0.94**	**0.73**	**0.90**	**0.81**	**12**	**2**	**1.00**
L8	IM-lc	**0.85**	0.40	0.55	**0.87**	0.03	0.06	61	39	**1.00**
	SM	*unsound*			*unsound*			160	118	0.02
	$SM_{2.0}$	0.77	**0.76**	**0.77**	0.38	**0.33**	**0.35**	**46**	**26**	0.70
L9	IM-lc	**0.94**	0.26	0.41	**0.92**	0.43	**0.58**	23	11	**1.00**
	SM	*unsound*			*unsound*			**17**	**5**	0.53
	$SM_{2.0}$	0.57	**0.91**	**0.70**	0.28	**0.45**	0.35	**17**	**5**	0.47
L10	IM-lc	**0.95**	0.75	0.84	**0.98**	0.15	0.26	31	8	**1.00**
	SM	0.77	**1.00**	**0.87**	0.95	**0.93**	**0.94**	**29**	**6**	**1.00**
	$SM_{2.0}$	0.77	**1.00**	**0.87**	0.95	**0.93**	**0.94**	**29**	**6**	**1.00**
L11	IM-lc	**0.91**	0.75	0.82	**0.45**	0.14	0.22	36	21	**1.00**
	SM	0.83	**0.90**	**0.87**	0.29	0.26	**0.28**	44	28	0.16
	$SM_{2.0}$	0.84	**0.90**	**0.87**	0.06	**0.33**	0.10	**22**	**11**	0.59
R-Log	IM-lc	**0.99**	0.98	**0.99**	**1.00**	0.96	**0.98**	16	5	**1.00**
	SM	0.91	**0.99**	0.95	0.45	0.83	0.59	**14**	**4**	0.36
	$SM_{2.0}$	0.98	0.97	0.98	0.94	**0.98**	0.96	16	5	0.50

Table 3: Experiment results.

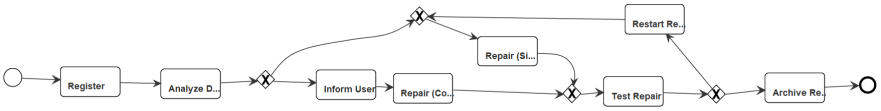
ures 6c and 6d), only $SM_{2.0}$ discovers the concurrency relations between its activities, while SM mixes us the concurrency relations with loops.



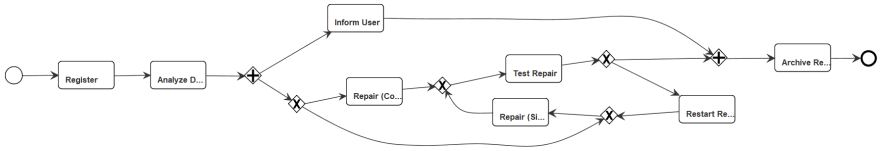
(a) SM model discovered from L6.



(b) $SM_{2.0}$ model discovered from L6.



(c) SM model discovered from R-Log.



(d) $SM_{2.0}$ model discovered from R-Log.

Fig. 6: Models discovered by SM and $SM_{2.0}$ from the L6 and R-Log.

5 Conclusion

In this paper, we presented Split Miner 2.0 ($SM_{2.0}$), an extension of Split Miner (SM) [6] that exploits the activities' start and end timestamps recorded in an event log to discover true concurrency and inclusive choice relations between activities. This is achieved by redesigning the discovery of a directly-follows graph from an event log, adapting the concurrency notion of Van der Werf et al. [18], and introducing an intuitive heuristic to identify inclusive relations. Furthermore, given that SM cannot guarantee sound process models, we designed an heuristic that reduces the chances of discovering process models exhibiting improper completion. The empirical evaluation shows that $SM_{2.0}$ can discover more concurrent relations than SM, remove improper completion, and identify OR-splits, while preserving SM's model accuracy and reducing the complexity.

Although several studies have investigated the problem of automated process discovery from event logs [5], most of them operate on simple event logs with only three attributes: case id, timestamp, and activity label. Future research work in this area may

focus on designing more sophisticated automated process discovery algorithms that can discover more complex BPMN process models by leveraging additional information that may be available in real-life event logs. Another direction for future work is to design accuracy measures such as fitness and precision that go beyond simple control-flow relations and include support for inclusive gateways, including the OR-join.

Acknowledgments. Research funded by the Australian Research Council (grant DP180102839) and the Estonian Research Council (grant PRG887).

References

1. A. Adriansyah, J. Munoz-Gama, J. Carmona, B. van Dongen, and W.M.P. van der Aalst. Alignment based precision checking. In *BPM*. Springer, 2012.
2. A. Adriansyah, J. Munoz-Gama, J. Carmona, B. van Dongen, and W.M.P. van der Aalst. Measuring precision of modeled behavior. *ISeB*, 13(1), 2015.
3. J. F. Allen. Maintaining knowledge about temporal intervals. *Communications of the ACM*, 26(11):832–843, 1983.
4. A. Augusto, A. Armas Cervantes, R. Conforti, M. Dumas, and La Rosa. Measuring fitness and precision of automatically discovered process models: A principled and scalable approach. *IEEE TKDE (to appear)*, 2020.
5. A. Augusto, R. Conforti, M. Dumas, M. La Rosa, F.M. Maggi, A. Marrella, M. Mecella, and A. Soo. Automated discovery of process models from event logs: Review and benchmark. *IEEE TKDE*, 31(4), 2019.
6. A. Augusto, R. Conforti, M. Dumas, M. La Rosa, and A. Polyvyanyy. Split miner: automated discovery of accurate and simple business process models from event logs. *KAIS*, 2018.
7. J. Buijs, B. van Dongen, and W. van der Aalst. On the role of fitness, precision, generalization and simplicity in process discovery. In *CoopIS*. Springer, 2012.
8. A. Burattin and A. Sperduti. Heuristics miner for time intervals. In *ESANN*, 2010.
9. S.J.J. Leemans and D. Fahland. Information-preserving abstractions of event data in process mining. *Knowledge and Information Systems*, 62(3):1143–1197, 2020.
10. S.J.J. Leemans, D. Fahland, and W.M.P. van der Aalst. Discovering block-structured process models from event logs containing infrequent behaviour. In *BPM Workshops*. Springer, 2014.
11. S.J.J. Leemans, D. Fahland, and W.M.P van der Aalst. Using life cycle information in process discovery. In *BPM*, pages 204–217. Springer, 2016.
12. J. Mendling. *Metrics for process models: empirical foundations of verification, error prediction, and guidelines for correctness*, volume 6. Springer Science & Business Media, 2008.
13. G. Schimm. Mining exact models of concurrent workflows. *Comput Ind*, 53(3):265–281, 2004.
14. A. Senderovich, M. Weidlich, and A. Gal. Temporal network representation of event logs for improved performance modelling in business processes. In *International Conference on Business Process Management*, pages 3–21. Springer, 2017.
15. A.F. Syring, N. Tax, and W.M.P. van der Aalst. Evaluating conformance measures in process mining using conformance propositions. In *Transactions on Petri Nets and Other Models of Concurrency XIV*, pages 192–221. Springer, 2019.
16. W.M.P. van der Aalst. *Process Mining - Data Science in Action*. Springer, 2016.
17. W.M.P. van der Aalst, K. van Hee, A. ter Hofstede, N. Sidorova, E. Verbeek, M. Voorhoeve, and M. Wynn. Soundness of workflow nets: classification, decidability, and analysis. *Formal Asp. Comput.*, 23(3), 2011.
18. J.M.E.M. van der Werf, R. Mans, and W.M.P. van der Aalst. Mining declarative models using time intervals. In *PNSE+ ModPE*, pages 313–331. Citeseer, 2013.
19. L. Wen, J. Wang, W.M.P. van der Aalst, B. Huang, and J. Sun. A novel approach for process mining based on event types. *J Intell Inf Syst*, 32(2):163–190, 2009.

A Novel Approach to Discover Switch Behaviours in Process Mining

Yang Lu[0000-0002-9002-8650], Qifan Chen[0000-0003-1068-6408] and Simon Poon[0000-0003-2726-9109]

School of Computer Science, The University of Sydney, Sydney, NSW 2006, Australia
{yalu8986, qche8411}@uni.sydney.edu.au, simon.poon@sydney.edu.au

Abstract. Process mining is a relatively new subject which builds a bridge between process modelling and data mining. An exclusive choice in a process model usually splits the process into different branches. However, in some processes, it is possible to switch from one branch to another. The inductive miner guarantees to return sound process models, but fails to return a precise model when there are switch behaviours between different exclusive choice branches due to the limitation of process trees. In this paper, we present a novel extension to the process tree model to support switch behaviours between different branches of the exclusive choice operator and propose a novel extension to the inductive miner to discover sound process models with switch behaviours. The proposed discovery technique utilizes the theory of a previous study to detect possible switch behaviours. We apply both artificial and publicly-available datasets to evaluate our approach. Our results show that our approach can improve the precision of discovered models by 36% while maintaining high fitness values compared to the original inductive miner.

Keywords: Process Discovery, Complex Behaviours Detection, Switch Behaviours, Inductive Miner, Process Trees.

1 Introduction

Process mining is useful for analyzing business processes along with improving and predicting which contains three parts – process discovery, conformance checking and process enhancement [1]. The most critical part of process mining is process discovery, which aims at extracting insight of the system workflow from real data. The resulting process model should not only have a high fitness value, but also be an accurate representation of the real process [2]. The inductive miner is one of the leading process discovery algorithms which can guarantee to produce sound process models within finite time [1, 3]. Given the direct outcome of the inductive miner is a process tree [3], the behaviours being represented are limited. When giving complex event logs as input, the inductive miner often returns so-called "flower models" which preserve high fitness but have very low precision values [3, 4]. Although we can still replay the majority of traces on the process model, "flower models" fail to represent real processes accurately and precisely [1, 2].

When dealing with an exclusive decision choice in a process model, the decision point is split into multiple branches [5]. However, in many real-life processes, it can be possible to switch between branches after a decision has been made. Although the inductive miner is known to be useful in generating sound models from data, it fails to discover an accurate model when switch behaviours exist.

In this paper, we propose a novel extension to the process tree model to handle switch behaviours between different exclusive choice branches. We then develop a novel extension to the inductive miner to discover sound process models with switch behaviours based on the theory in [6]. From a broader perspective, our proposed method not only guarantees to produce sound process models but also not being limited to produce block-structured process models. We apply both artificial and publicly-available datasets to evaluate our approach. Fitness, precision and F-score [4, 7] are used to measure the accuracy of resulting models, size (the number of nodes) and CFC (the number of branching caused by split gateways) [8] are adopted to measure the model complexity.

The rest of the paper is structured as follows: Section 2 is a literature review of related work. Section 3 introduces formal definitions of some terms. Section 4 introduces the extension to the process tree model and how to translate it into a workflow net. In Section 5, we describe our process discovery technique. The approach is evaluated in Section 6. We finally conclude our paper in Section 7.

2 Background

When modelling switch behaviours between different exclusive choice branches using Petri-nets, a hidden transition is needed since we cannot connect two places directly [6]. The classical alpha algorithm [9] cannot discover any hidden transitions. [6, 10] improve the classical alpha algorithm to allow the detection of invisible tasks. Although the alpha algorithms are not robust to noises and cannot guarantee to produce sound models. [6] proposes a heuristic for detecting invisible transitions between activities directly from event logs. If there is a hidden transition between two activities on different exclusive choice branches, a switch behaviour is detected.

In reviewing other process discovery algorithms which can discover switch behaviours between different exclusive choice branches including the alpha algorithms with invisible tasks [6, 10], heuristics miners [11], genetic miners [12] and the ILP algorithm [13], none of them can guarantee to produce a sound process model. In addition, some of them cannot handle noises, thus, not suitable to be applied to real data. Although the split miner [7] can discover switch behaviours and guarantee to produce deadlock-free models. It still cannot guarantee to produce sound models as defined in [9], which defines soundness as (a) safeness, (b) proper completion, (c) option to complete, (d) absence of dead tasks.

The inductive miners are a family of process discovery algorithms which utilize the divide-and-conquer approach in the field of process discovery [3, 14-18]. The inductive miners recursively divide the activities into different partitions and split event logs until base cases are touched. The direct outcomes of the inductive miners are process trees, which can be easily translated into equivalent block-structured workflow nets [3]. An

important feature of the inductive miner family is that the resulting model is always sound regardless of the input log. However, process trees also limit the behaviours which can be represented. For example, they fail to represent switch behaviours between exclusive choice branches.

When the given event log is complex, the inductive miner [3] can easily return a "flower model" with high fitness but low precision. [14] removes infrequent relations between activities before partitioning the activities. However, according to the benchmark in both [4] and [7]. The inductive miner infrequent (IMf) in [14] still returns models with low precision values compared with other algorithms. [19] tries to solve the problem by giving duplicate labels to the same activity when a local "flower model" is returned. The algorithm successfully improves the precision of the outcome models but leads to longer execution time. Besides, if we apply the algorithm in [19] with the inductive miners, the outcome models are still block-structured workflow nets.

The process mining framework in the original inductive miner [3] allows researchers to define their ways to partition activities and customized process tree semantics. For example, [17] puts lifecycle information into the process discovery to distinguish "interleaving" behaviours from "parallel" behaviours. [18] defines new operators on the process tree and uses the inductive miner to discover cancellation behaviours.

3 Preliminaries

In this section, we present some formal definitions which will be used in this paper. For process trees and block-structured workflow nets, we refer to [3], for soundness of Petri-nets, we refer to [9]. Besides, for IWF-net (workflow nets with invisible tasks), DIWF-net (a subset of IWF-nets) and log completeness, we refer to [6]. For clarification, in this paper, we use "X" to represent the exclusive choice operator, "→" to represent the sequence operator, "⋀" to represent the parallel operator and "↺" to represent the loop operator in the process tree [3].

Definition 1 (Relations between activities). Let L be an event log of a workflow net N, let a, b be two activities in L. Then:

1. $a >_L b$ if there is a trace $t \in L$ where t = <……, a, b, …… >,
2. $a \sim_L b$ if there is a trace $t \in L$ where t = <……, a, b, a, …… >, and there is a trace $t \in L$ where t = <……, b, a, b, …… >,
3. $a \rightarrow_L b$ if $a >_L b \wedge (b \not>_L a \vee a \sim_L b)$,
4. $a \,||_L b$ if $a >_L b \wedge b >_L a \wedge a \nsim_L b$.

Definition 2 (Mendacious dependency) [6]. Let N = (P, $T_v \cup T_{iv}$, F) be a potential sound IWF-net, T_v is the set of visible tasks, T_{iv} is the set of invisible tasks. There is a mendacious dependency between activities a, b in event log L, denoted as $a \twoheadrightarrow_L b$, iff $a \rightarrow_L b \wedge \exists x, y \in T_v: a \rightarrow_L x \wedge y \rightarrow_L b \wedge y \not>_L x \wedge x \nparallel_L b \wedge a \nparallel_L y$.

4 The Switch Process Tree

In this section, we formally define the switch behaviour and its corresponding representation on the process tree. The switch process tree is a novel extension based on the process tree model described in [3].

Definition 3 (First, Path function). Let n be a leaf in a process tree, tp be an arbitrary operator type. First (n, tp) refers to the first ancestor node of n with operator type tp. For example, in the process tree shown in Fig. 1, First (Node 3, X) refers to the root node. First $(n, tp) = \emptyset$ if none of the ancestor nodes of n has type tp. Let n_1, n_2 be two arbitrary nodes of a process tree, Path (n_1, n_2) refers to the path from n_1 to n_2 (excluding n_1 and n_2). Path $(n_1, n_2) = \emptyset$ if n_2 is not reachable from n_1 or $n_1 = n_2$. Referring back to Fig. 1, Path (Node 0, Node 8) = <Node 2>, Path (Node 1, Node 8) = $\emptyset$.

Definition 4 (Switch process tree and switch behaviour). Assume a finite alphabet A of activities. A switch process tree is a normal process tree with switch leaf operators $a \Rightarrow B$ where $a \in A$, $B \subset A$. Combined with an exclusive choice operator X, the novel leaf node denotes the place we execute activity a, and have an option to switch to one of the activities in set B on another branch of an exclusive choice operator. $a \Rightarrow b$ is a switch behavior if there exists $a \Rightarrow B$ such that $b \in B$, we call a the source of the switch behavior, b the destination of the switch behaviour. To ensure the process model is still sound, we define the constraints below:

1. The activities on different sides of a switch leaf node must be put on different branches of an exclusive choice operator X, i.e. we can only switch execution rights from one exclusive choice branch to another.
2. If there exists a leaf operator $a \Rightarrow B$ in the process tree, then $\forall\, b \in B$, First $(a \Rightarrow B, \bigwedge)$ = First $(b, \bigwedge)$, and if First $(a \Rightarrow B, \bigwedge)$ = First $(b, \bigwedge) \neq \emptyset$, then Path (First $(a \Rightarrow B, \bigwedge)$, $a \Rightarrow B) \cap$ Path (First $(b, \bigwedge)$, $b) \neq \emptyset$. i.e. we cannot switch out of a parallel branch.

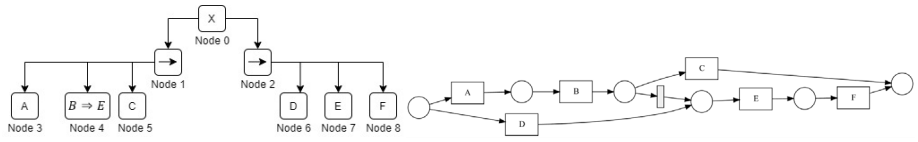


Fig. 1. An example switch process tree and its corresponding workflow net

Fig. 1 shows an example switch process tree and its corresponding workflow net. There are three possible traces in the model, which are <A, B, C>, <D, E, F> and <A, B, E, F>.

Definition 5 (Translating switch process trees into workflow nets). Translating a switch process tree into a workflow net is straightforward. We first ignore the switch leaf nodes and translate the process tree into a block-structured workflow net. Then we connect the activities on different sides of the switch operators using hidden transitions.

Suppose Tr is a switch process tree, S is the set of all the switch leaf nodes in Tr, Tr^* is an equivalent process tree of Tr but all the switch behaviours are removed, i.e., for all the $a \Rightarrow B \in S$ in Tr, we convert them into a in Tr^*. $N = (P, T_v \cup T_{iv}, F)$ is a block-structured workflow net corresponding to Tr^*. For each $a \Rightarrow B \in S$ and $b \in B$, we create a new invisible task t_{switch} into set T_{switch}, then:

1. If $|a \cdot| = 1$ in N, $p_{a-out} = a \cdot$, $|\cdot p_{a-out} \backslash T_{switch}| = 1$, then we add a new arc f into N, $f = (p_{a-out}, t_{switch})$.
2. If $|\cdot b| = 1$, $p_{b-in} = \cdot b$, $|p_{b-in} \cdot \backslash T_{switch}| = 1$, then we add a new arc f into N, $f = (t_{switch}, p_{b-in})$.
3. If $|a \cdot| = 1$ in N, $p_{a-out} = a \cdot$, $|\cdot p_{a-out} \backslash T_{switch}| > 1$, we first delete the arc $f_1 = (a, p_{a-out})$ from N, then we create another new invisible task t_{bridge} and place p_{bridge} into N. We finally add arcs $f_2 = (a, p_{bridge})$, $f_3 = (p_{bridge}, t_{bridge})$, $f_4 = (t_{bridge}, p_{a-out})$ and $f_5 = (p_{bridge}, t_{switch})$.
4. If $|\cdot b| = 1$, $p_{b-in} = \cdot b$, $|p_{b-in} \cdot \backslash T_{switch}| > 1$, we first delete the arc $f_1 = (p_{b-in}, b)$ from N, then we create another new invisible task t_{bridge} and place p_{bridge} into N. We finally add arcs $f_2 = (p_{bridge}, b)$, $f_3 = (t_{bridge}, p_{bridge})$, $f_4 = (p_{b-in}, t_{bridge})$ and $f_5 = (t_{switch}, p_{bridge})$.
5. If $|a \cdot| > 1$, there is a "and split" after a. We add a new invisible task t_{bridge} after a as the split point and then go back to step 1, i.e., $|a \cdot| = 1$, $|t_{bridge} \cdot| > 1$, $a \cdot \cap \cdot t_{bridge} \neq \emptyset$.
6. If $|\cdot b| > 1$, there is a "and join" before b. We add a new invisible task t_{bridge} before b as the joining point and then go back to step 1, i.e., $|\cdot b| = 1$, $|\cdot t_{bridge}| > 1$, $\cdot b \cap t_{bridge} \cdot \neq \emptyset$.

To illustrate the translation process, we use three examples translated from the above different scenarios in Fig. 2 - Fig. 4:

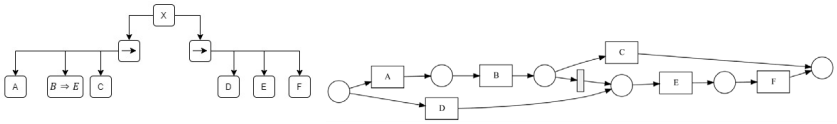


Fig. 2. An example translation from a switch process tree to a workflow net (Definition 5, case 1, 2).

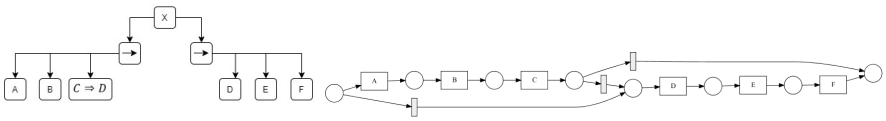


Fig. 3. An example translation from a switch process tree to a workflow net (Definition 5, case 3, 4).

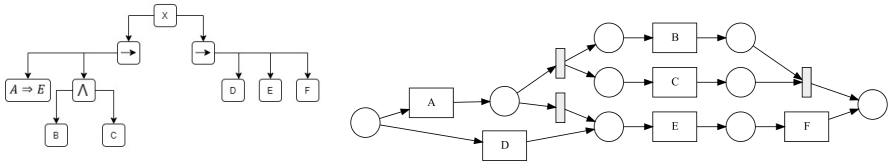


Fig. 4. An example translation from a switch process tree to a workflow net (Definition 5, case 5).

Theorem 1. If we translate a switch process tree into a workflow net, the resulting workflow net is always sound if the constraints in Definition 4 are all satisfied.

Proof. Assume we ignore all the switch behaviours in the process tree during the translation, according to [3], we can get an equivalent sound block-structured workflow net. According to Definition 5, each switch invisible transition is always connected to one single input place and one single output place. The translation process does not increase the number of input/output places of any transitions. As a result, it is free for a token in the model to choose whether firing a switch invisible transition or not. Thus, the resulting process model will not contain dead tasks and is always safe. In addition, since the original block-structured workflow net is sound, if we move a token from one exclusive choice branch to another one, the process can still be completed properly as long as we don't move the token out of a parallel branch. Thus, if the constraints in Definition 4 are all satisfied, the resulting workflow net is always sound.

5 Discovering Switch Process Trees

In [6], researchers define the prime invisible tasks into SKIP, REDO, SWITCH, INITIALIZE and FINALIZE where SWITCH refers to switching execution rights between alternative branches. Thus, the SWITCH invisible tasks can be used to represent the switch behaviours we define in Section 4. Researchers in [6] prove that given L is a complete event log of a sound DIWF-net N = (P, $T_v \cup T_{iv}$, F), if $a, b \in T_v$ are two visible tasks, then there is a prime invisible task $t \in T_{iv}$ between a and b, i.e., $a \cdot\cap\cdot t \neq \emptyset$ and $t \cdot\cap\cdot b \neq \emptyset$ iff $a \twoheadrightarrow_L b$. Although the scope of the proof is limited, the evaluation of [6] shows that the power of the theory is not limited to complete logs of DIWF-nets. More importantly, [6] provides us with a heuristic to predetermine possible invisible tasks between activities from event logs directly. Suppose we know two activities are on two different exclusive choice branches and there is an invisible task between them, then we know there is a switch behaviour between the two activities.

To discover switch process trees using the inductive miner, we extend the normal exclusive choice cut of the inductive miner framework to a switch exclusive choice cut. In this section, we show the switch exclusive choice cut step by step. To illustrate the process, we use a complete log of the example model presented in Fig. 2 $L_1 = < A, B, C >, < D, E, F >, < A, B, E, F >$ as a running example. To make sure we detect all the switch behaviours, we put the switch exclusive choice cut before the other three cuts in each iteration. The extended IM framework is shown in Algorithm 1.

Algorithm 1. The extended IM framework augumented with switch behaviours

Input: An event log L
Output: A Switch Process Tree Tr
$Discover\ (L)$
1 **If** $BaseCase(L)\ !=\ \phi$ **Then Return** $BaseCase(L)$
2 **Else**
3 $(cut, (\Sigma_1, \dots, \Sigma_k), switches) = SwitchExclusiveChoiceCut(G(L), L)$
4 **If** $k \leq 1$ **Then** $\left(cut, (\Sigma_1, \dots, \Sigma_k)\right) = SequenceCut(G(L))$ *//cut in the original IM*
5 **If** $k \leq 1$ **Then** $\left(cut, (\Sigma_1, \dots, \Sigma_k)\right) = ConcurrentCut(G(L))$ *//cut in the original IM*
6 **If** $k \leq 1$ **Then** $\left(cut, (\Sigma_1, \dots, \Sigma_k)\right) = LoopCut(G(L))$ *//cut in the original IM*
7 **If** $cut = \phi$ **Then Return** $Fallthrough(L)$ *//function in the original IM*
8 $NumOfActivitiesBefore = CountActivities(L)$
9 $(L_1, \dots, L_k) = SplitLog(L, (cut, (\Sigma_1, \dots, \Sigma_k)))$ *//function in the original IM*
10 **If** $cut = SwitchExclusiveChoiceCut$ **Then**
11 $NumOfActivitiesAfter = CountActivities(L_1, \dots, L_k)$
12 **If** $NumOfActivitiesBefore\ !=\ NumOfActivitiesAfter$ **Then**
13 $(cut, (\Sigma_1, \dots, \Sigma_k)) = XORCut(G(L), L)$ *//Exclusive choice cut of the original IM*
14 **If** $k \leq 1$ **Then** $\left(cut, (\Sigma_1, \dots, \Sigma_k)\right) = SequenceCut(G(L))$ *//cut in the original IM*
15 **If** $k \leq 1$ **Then** $\left(cut, (\Sigma_1, \dots, \Sigma_k)\right) = ConcurrentCut(G(L))$ *//cut in the original IM*
16 **If** $k \leq 1$ **Then** $\left(cut, (\Sigma_1, \dots, \Sigma_k)\right) = LoopCut(G(L))$ *//cut in the original IM*
17 **If** $cut = \phi$ **Then Return** $Fallthrough(L)$ *//function in the original IM*
18 $(L_1, \dots, L_k) = SplitLog(L, (cut, (\Sigma_1, \dots, \Sigma_k)))$ *//function in the original IM*
19 **Return** $cut(Discover(L_1), \dots, Discover(L_k))$

5.1 The Switch Exclusive Choice Cut (Line 3)

Step 1: Adding Artificial Start and End Activities
According to Definition 5, if the source activity of a switch behaviour is at the end of an exclusive choice branch or if the destination activity of a switch behaviour is at the beginning of an exclusive choice branch, we need to add an extra invisible task before the destination activity or after the source activity to represent the process precisely. However, the process model is no longer a DIWF-net after adding the extra invisible task according to [6], so we may fail to detect an invisible task between the two activities using the mendacious dependency.

To solve the problem, before a switch cut, we first identify all the start and end activities in the event log and add a unique start and end activity to each of them. For example, after adding artificial activities into L_1, we get $L_1^* = < Start_A, A, B, C, End_C >, < Start_D, D, E, F, End_F >, < Start_A, A, B, E, F, End_F >$.

Step 2: Calculating All the Mendacious Dependencies Between Activities
We then go through the event log and identify all the mendacious dependencies. Besides, we ignore all the mendacious dependencies containing artificial start and end activities. After the mendacious dependencies are identified, we delete all the artificial start and end activities.

In our example, we get one mendacious dependency, which is $B \rightsquigarrow_L E$.

Step 3: Finding Switch Exclusive Choice Cut and Switch Leaf Operators
Firstly, if there is a mendacious dependency between two activities in the directly-follows graph, we replace the edge between them with an invisible edge.

Definition 6 (Invisible edge). Given $G(L)$ is a directly-follows graph of event logs L, A is the set of activities in L. $a, b \in A$, there is an invisible edge from a to b in $G(L)$ iff $a \nrightarrow_L b$.

Definition 7 (Switch exclusive choice cut). Suppose E is the set of all edges in $G(L)$, E^* is the set of all invisible edges. A switch exclusive choice cut is a cut $\Sigma_1, \Sigma_2, \dots, \Sigma_n$ of a directly-follows graph $G(L)$ such that:

1. There are only invisible edges between $\Sigma_{1 \le i \le n}$ and $\Sigma_{1 \le j \le n}$.

$$\forall i \neq j \wedge a_i \in \Sigma_i \wedge a_j \in \Sigma_j : (a_i, a_j) \notin E \backslash E^*$$

Fig. 5. Switch exclusive choice cut for L_1

If an invisible edge is cut during the switch exclusive choice cut, i.e., $\exists a_i \in \Sigma_i, a_j \in \Sigma_j : (a_i, a_j) \in E^*$, then a new switch behaviour $a_i \Rightarrow a_j$ is discovered. By merging all the switch behaviours with the same source together in the end, we can get switch leaf operators.

Step 4: Removing Traces with Switch Behaviours
We use the same exclusive choice cut split function as the inductive miner infrequent [14] to split the event logs after an exclusive choice switch cut. A problem here is splitting the event log could cause extra "skip" behaviours. In our running example, since we partitioned the activities into two groups which are $\{A, B, C\}$ and $\{D, E, F\}$, the trace <A, B, E, F> will be projected into either <A, B> or <E, F>. The options will either produce an extra end activity B or an extra start activity E in the local sub-process. To resolve the issue, we consider deleting the traces with switch behaviours before splitting the log. For example, we delete <A, B, E, F> from L_1 before splitting the log. However, deleting traces increases the requirement of log completeness, which may cause the loss of activities or behaviours when dealing with real-life data. We decide to make the option adjustable.

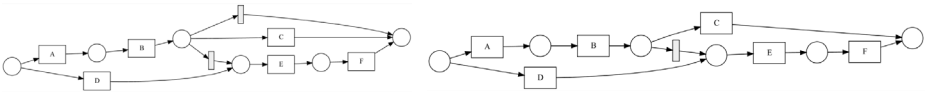


Fig. 6. The resulting model of L_1 with the deleting traces option off (left) and on (right), there is one more extra trace <A, B> on the left graph.

5.2 Verifying the Exclusive Choice Switch Cut (Line 10 - 18)

Performing the switch exclusive choice cut and splitting the event log may cause unnecessary loss of activities. Every time after we perform the switch exclusive choice cut and split the event log, we check if the total number of activities changes. If there is a change in the number of activities (line 12), we abort the whole cut, redo the log split and disable the exclusive choice cut in the next iteration (line 13). We enable the exclusive choice cut again after the current log has been split into sub logs.

5.3 Removing Incorrect Switch Behaviours

Although we can identify switch behaviours during the exclusive choice cut, we are unable to determine if the constraints in Definition 4 are met before the whole process tree has been constructed. To ensure a sound model is returned, we iterate through the whole process tree at the end and delete any switch behaviours which violate the constraints defined in Definition 4.

6 Evaluation

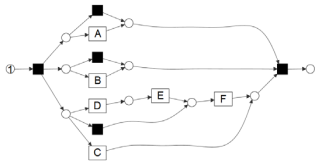
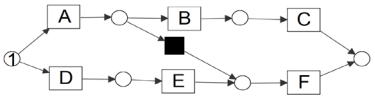
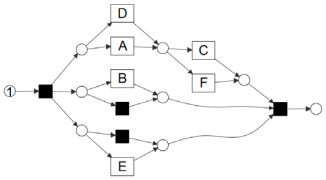
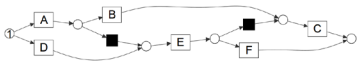
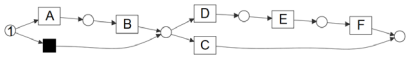
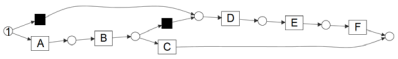
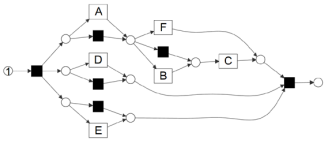
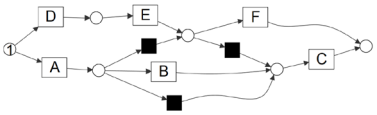
We implement our approach on the inductive miner directly in the ProM framework [20]**.** Our code and evaluation results are available at https://github.com/bearlu1996/switch. We applied both artificial and publicly-available event data to evaluate our algorithm. Fitness and precision are used to evaluate the accuracy of our process models. Besides, we use the formula in [4] and [7] to calculate F-score, i.e., $F - Score = 2 * \frac{fitness * precision}{fitness + precision}$. CFC (the number of branching caused by split gateways) [8] and size (the number of nodes) are also used to evaluate the complexity of our process models. For replicable purposes, we use "Replay a log on Petri net for conformance analysis" in ProM to calculate fitness, "Check Precision based on Align-ETConformance" to calculate the precision. We use "Calculate BPMN Metrics" to calculate model complexity. In addition, the tools in the "BPMN Miner" are used to covert between Petri-nets and BPMNs. We use default settings for all the parameters.

6.1 Evaluation Using Artificial Data

We first use several artificial logs with switch behaviours to demonstrate the performance of our approach. When applying the original inductive miner on these logs, it fails to discover precise models. Instead, "flower" models with low precision are returned. We show that after using our extension, we can get precise models.

Table 1. Evaluation using artificial data

Log: <A, B, C>, <D, E, F>, <A, F>

IM	**IM augmented with switch behaviours**
	
Fitness: 1.0, Precision: 0.38	Fitness: 1.0, Precision: 1.0
Log: <A, B, C>, <D, E, F>, <A, E, F>, <D, E, C>, <A, E, C>	
IM	**IM augmented with switch behaviours**
	
Fitness: 1.0, Precision: 0.42	Fitness: 1.0, Precision: 1.0
Log: <A, B, C>, <D, E, F>, <A, B, D, E, F>	
IM	**IM augmented with switch behaviours**
	
Fitness: 1.0, Precision: 0.94	Fitness: 1.0, Precision: 1.0
Log: <A, B, C>, <D, E, F>, <D, E, C>, <A, F>, <A, C>	
IM	**IM augmented with switch behaviours**
	
Fitness: 1.0, Precision: 0.42	Fitness: 1.0, Precision: 0.97

6.2 Evaluation Using Publically-Available Data

We use a publicly-available dataset called “BPIC13-incident” from the “4TU Center for Research Data” to evaluate our algorithm. We use “Event name + lifecycle” as the activity classifier, the dataset contains 7554 traces, 2278 distinct traces, 65533 events and 13 distinct events. The average length of traces is 9 while the shortest length is 1 and the longest length is 123. We combine our approach with the inductive miner infrequent (IMf) [14] and switch off the “delete trace” option, we also compare our results with the split miner (SM) [7]. In addition, we use default settings for all the parameters.

Table 2. Evaluation results with the publicly-available dataset (IMs refers to our approach)

	Accuracy			Complexity	
	Fitness	**Precision**	**F-Score**	**Size**	**CFC**
IMf	0.95	0.59	0.73	35	**33**
SM	**0.98**	0.71	0.82	39	48
IMs	0.97	**0.80**	**0.88**	**33**	46

Evaluation results are presented in Table 2. All three methods can produce a model with high fitness. However, the IMf returns a model with low precision. Our approach rises the precision of IMf by 36%. In addition, our approach returns a model with both higher precision and F-score than the split miner. For the model complexity, our approach also achieves both smaller size and CFC than the split miner.

7 Discussion and Conclusion

In this paper, we present an extension to both the inductive miner and the process tree model. We allow the inductive miner to discover sound process models but not being limited to block-structured workflow nets. The evaluation results show that our approach can reduce the chance for the inductive miner to return flower models. Besides, in our evaluation, our approach can also discover models that are comparable in terms of both model accuracy and complexity to these produced by the split miner.

One limitation is that when performing the switch exclusive choice cut, we do not know if the switch behaviour is valid or not, thus we need to check the validity of the switch behaviours to make sure the model is still sound in the end. It has to be noted that the fitness of resulting models might be reduced if too many switch behaviours are removed. We aim to develop better algorithms to repair the models in the future. Besides, as shown in the artificial data evaluation, when the same place is both the input and output of two switch invisible transitions, there might be redundant hidden transitions in the model, future work is required to remove these redundant hidden transitions.

Finally, we also aim to conduct more experiments to evaluate the performance of our approach in the future, including the impacts of different orders of cuts.

References

1. van der Aalst, W.M.P.: Process Mining - Data Science in Action. Springer, Heidelberg (2016)
2. Buijs, J., 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)
3. 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)
4. Augusto, A., Conforti, R., Dumas, M., La Rosa, M., Maggi, F.M., Marrella, A., Mecella, M., Soo, A.: Automated discovery of process models from event logs: review and benchmark. IEEE Trans. Knowl. Data Eng. 31, 686–705 (2018)

5. Dumas, M., La Rosa, M., Mendling, J., Reijers, H.: Fundamentals of Business Process Management. Springer (2013)
6. Wen, L., Wang, J., van der Aalst, W.M.P., Huang, B., Sun, J.: Mining process models with prime invisible tasks. Data & Knowledge Engineering 69(10), 999–1021 (2010)
7. 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. 50, 1–34 (2019)
8. Cardoso, J.S.: Business process control-flow complexity: metric, evaluation, and validation. Int. J. Web Serv. Res. 5(2), 49–76 (2008)
9. van der Aalst, W.M.P., Weijters, A.J.M.M., Maruster, L.: Workflow mining: discovering process models from event logs. IEEE Trans. Knowl. Data Eng. 16(9), 1128–1142 (2004)
10. Guo, Q., Wen, L., Wang, J., Yan, Z., Yu, P.S.: Mining invisible tasks in non-free-choice constructs. In: Motahari-Nezhad H., Recker J., Weidlich M. (eds.) BPM 2016. LNCS, vol 9253, pp. 109-125. Springer, Cham (2015)
11. Weijters, A., Ribeiro, J.: Flexible heuristics miner (FHM). In: CIDM, pp. 310–317. IEEE (2011)
12. de Medeiros, A., Weijters, A., van der Aalst, W.: Genetic process mining: an experimental evaluation. Data Min. Knowl. Discov. 14(2), 245–304 (2007)
13. van Zelst, S., van Dongen, B., van der Aalst, W.M.P., Verbeek, H.M.W.: Discovering workflow nets using integer linear programming. Computing 100, 529 (2018)
14. Leemans, S.J.J., Fahland, D., van der Aalst, W.M.P.: Discovering block-structured process models from event logs containing infrequent behaviour. In: Lohmann, N., Song, M.,Wohed, P. (eds.) BPM 2013. LNBIP, vol. 171, pp. 66–78. Springer, Cham (2014).
15. Leemans, S.J.J., Fahland, D., van der Aalst, W.M.P.: Discovering block-structured process models from incomplete event logs. In: Ciardo, G., Kindler, E. (eds.) PETRI NETS 2014. LNCS, vol. 8489, pp. 91–110. Springer, Heidelberg (2014)
16. Leemans, S.J., Fahland, D., van der Aalst, W.M.: Scalable process discovery and conformance checking. Softw. Syst. Model. 16, 1–33 (2016)
17. Leemans, S.J.J., Fahland, D., van der Aalst, W.M.P.: Using life cycle information in process discovery. In: Reichert, M., Reijers, H. (eds.) BPM Workshops 2015. LNBIP, vol. 256, pp. 204–217. Springer, Heidelberg (2016)
18. Leemans, M., van der Aalst, W.M.P.: Modeling and discovering cancelation behavior. In: Panetto H. et al. (eds.) OTM 2017. LNCS, vol 10573, pp. 93-113. Springer, Cham (2017)
19. Lu, X., Fahland, D., Biggelaar, F.J.H.M., 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)
20. van Dongen, B.F., de Medeiros, A.K.A., Verbeek, H.M.W., Weijters, A.J.M.M., van der Aalst, W.M.P.: The ProM framework: a new era in process mining tool support. In: Ciardo, G., Darondeau, P. (eds.) ICATPN 2005. LNCS, vol. 3536, pp. 444–454. Springer, Heidelberg (2005)

Process Model Discovery from Sensor Event Data

Dominik Janssen[1], Felix Mannhardt[2,3], Agnes Koschmider[1], and Sebastiaan J. van Zelst[4,5]

[1] Group Process Analytics, Kiel University, Germany
dominik.janssen|ak@informatik.uni-kiel.de
[2] Dept. of Technology Management, SINTEF Digital, Norway
f.mannhardt@tue.nl
[3] Dept. of Computer Science, NTNU, Norway
[4] Fraunhofer Institute for Applied Information Technology, Fraunhofer Gesellschaft, Germany
sebastiaan.van.zelst@fit.fraunhofer.de
[5] Chair of Process and Data Science, RWTH Aachen University, Germany

Abstract. Virtually all techniques, developed in the area of process mining, assume the input event data to be discrete, and, at a relatively high level (i.e., close to the business-level). However, in many cases, the event data generated during the execution of a process is at a much lower level of abstraction, e.g., sensor data. Hence, in this paper, we present a novel technique that allows us to translate sensor data into higher-level, discrete event data, thus enabling existing process mining techniques to work on data tracked at a sensory level. Our technique discretises the observed sensor data into activities by applying unsupervised learning in the form of clustering. Furthermore, we refine the observed sequences by deducing imperative sub-models for the observed discretised data, i.e., allowing us to identify concurrency and interleaving within the data. We evaluated the approach by comparing the obtained model quality for several clustering techniques on a publicly available data-set in a smart home scenario. Our results show that applying our framework combined with a clustering technique yields results on data that otherwise would not be suitable for process discovery.

Keywords: Process mining · sensor data · event correlation · IoT.

1 Introduction

The rise of the Internet-of-Things (IoT), i.e., interconnected devices, mechanical and digital machines, gradually digitalises the day-to-day operations of modern-day enterprises. More-and-more devices are interconnected and store valuable traces of behavioural data, generated during their interaction with humans, as well as other interconnected devices. For example, consider the concepts of *autonomous production* and the adoption of *robotics in healthcare*, in which operational processes are gradually digitised and automated, utilising interconnecting and communicating devices and machines.

Whereas the design of a single device, connected to a larger network of devices, remains manageable (though it is complex in its own right), deficiencies in inter-device communication or handover of work-packages, easily lead to global process under-performance. Hence, a clear understanding of the general flow of work, as well as an understanding of bottlenecks and synchronisation points is of utmost importance to further improve the efficiency of the executed processes. *Process mining* techniques aim to exploit behavioural data, stored in the information systems to support the execution of processes and to distil process models [1]. In particular, they can derive-and-construct process models based on tracked event data, i.e., in a *completely automated* fashion.

In general, process mining relies on discrete event data, typically assumed to be tracked at *the business level*, i.e., the event data directly relates to high-level business process concepts. However, often, the level at which the event data is tracked within information systems is at a much lower level.

Possible application scenarios are settings where the movement of objects or people (entities) is tracked by motion sensors, light barriers or similar types of sensors that only detect absence and presence of a person or object and cannot distinguish between different observed entities. Those sensors can be found in smart home settings, smart factories and healthcare-related applications. If in these possible settings, it is of interest to discover frequent behaviour patterns or abnormal behaviour, our proposed method provides a novel approach that translates sensory data, into a process model. In particular, unlabelled raw sensor events are aggregated and clustered by an unsupervised learning technique to identify activities through clusters of related event sequences. To identify the activities, we discover a process model for each identified cluster. The activities, labelled by a domain expert, serve as input for process mining-based model discovery, which allows to identify concurrent and interleaving behaviour in sensor event data. We evaluate our approach on the publicly available CASAS dataset [2] and compare two clustering methods. The obtained results show promising results, hinting towards a better result by using clustering based on a self-organising map (SOM) in comparison to basic k-means in this context based on our methodology.

To the best of our knowledge, this paper suggests the first activity and process discovery technique for unlabelled sensor event data using SOM as model and addressing the challenges of concurrent behaviour between activities and multiple residents.

The remainder of this paper is structured as follows. The next section presents related work. Subsequently, section Section 3 presents our approach, which has been evaluated using a real-life data-set. The evaluation is summarised in Section 4. The paper concludes with an outlook on future work in Section 5.

2 Related Work

A large body of research exists that partially addresses the discovery of events and activities at different levels (see Fig. 1). In the following we consider related

approaches that use sensor data aiming to translate it into higher-level, discrete event data or applying process mining on raw sensor data. Our focus of related approaches also lays in smart homes as we used position data of smart home sensors for evaluation.

Activity recognition in smart home has been widely addressed relying the recognition on different sensor types like motion or video [3–5] or analysing data from wearables [6] or reference sensors [7]. Recently, Deep Learning (DL) methods for detecting and predicting activities in IoT environments have been increasingly explored [8]. Unlike classical machine learning techniques, DL networks automatically derive features from the data and produce promising results in different domains. Particularly in the field of smart homes or ambient assisted living, there are first approaches that recognise activities based on sensor event data [9–12]. Activity recognition is predominantly used for a situational prognosis [13]. Also these kinds of approaches identify simple activities [14, 15]. Complex activities like people's daily activities can only be identified using extra sensors [14, 16]. Although our method for process model discovery from raw location sensor data also requires a manual labelling of clusters of high-level events, we believe that the process model view on raw sensor data advances existing approaches and is beneficial in terms of evaluating the quality of data aggregations, which DL-based approaches are not capable of.

Mapping low-level events to activities for process mining is still a challenge [17]. Leotta et al. [18] envision to use similar techniques as we employ: however, only discuss challenges. The current status-quo is that approaches indicate only likelihoods of mappings, since there is often more than one possible solution [19]. Our approach for event aggregation in combination with unsupervised learning aims to bridge this gap. Related literature for activity discovery for process mining either use supervised techniques [6, 20] or visualise human habits [21] in order to accurately identify activities. Some works exist that detect activities from high-level events through unsupervised techniques [20,22,23], which have been compared in this paper. These related works [20,22,23] use patterns or local process models to aggregate event data towards higher abstraction levels. But they did not allow to discover meaningful activities for our data set. For unlabelled training sets, related approaches suggest to use a time-based label refinement [24] or locations [25] as characteristics in order to segment the event log and to abstract activities out of it. However, the methods already expects particular representations of traces. Given our scenario, the application of local process models did not allow to identify useful process fragments.

3 Translating Sensor Data to High-Level Traces

Our method for process model discovery from raw location sensor data assumes a location sensor event log E_L as input derived from a set of sensors S e.g., networks of WiFi-access points, or motion sensors in smart homes. We expect events $e \in E_L$ to satisfy some minimal requirements: For each event we can retrieve a timestamp $time(e)$ inducing a partial order on the events, a sensor label

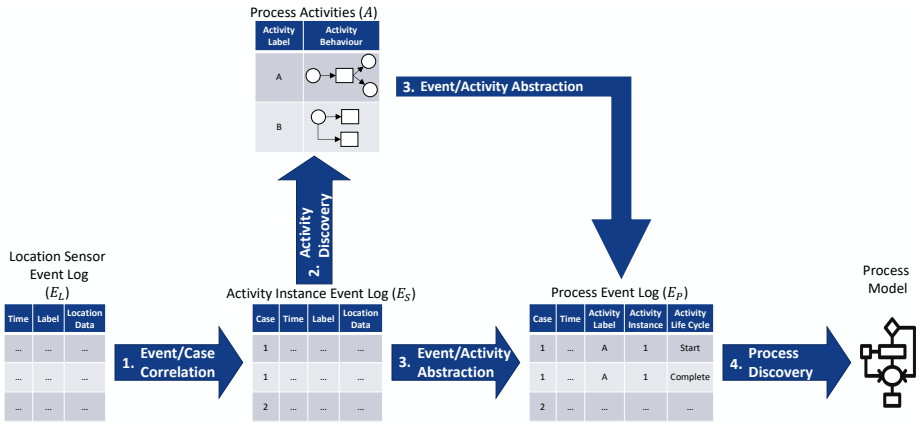


Fig. 1: Process discovery approach for location sensor event data.

$sensor(e) \in S$ indicating which sensor was activated and some form of information that either implicitly or explicitly refers to a location (i.e., $location(e) \in L$). The location information can be explicit in the form of coordinates (e.g., latitude, longitude) or implicit by providing labelled locations together with a distance function providing pairwise distances between them. Throughout the paper we assume that E_L was generated by one or more *entities* $n \in N$. An entity may be a person or an object in the observed area.

Events in a location sensor log do not necessarily have a unique identifier attached to identify by which entity they were triggered. Often data contains overlapping and concurrent activities by multiple entities. In smart homes or factories, multiple entities can be present at the same time. It has to be ensured that the analysed activities are all associated with the correct entity, to obtain a meaningful process model on a by-entity-level. Our method targets such scenarios where a sensor cannot identify entities utilising a unique identifier as it is the case in WiFi networks, for example.

Figure 1 gives an overview of the proposed approach, which consists of the following four steps that are explained in the following sections:

1. *Event Correlation*: Correlation of events from a location sensor event log E_L to (unlabelled) activity instances yielding an instance log E_I.
2. *Activity Discovery*: Discovery of process activities A together with their labels and sensor-level process models describing the expected behaviour on a sensor level.
3. *Event Abstraction*: Abstraction of the instance log E_I to a process event log E_P where events are directly related to the *start* or *completion* of process activities A.
4. *Process Discovery*: Process discovery based on the process event log E_P resulting in an activity-level process model defined over activities A.

3.1 Event Correlation

The first step towards process mining on raw sensor events is to group the input data according to a set of numbered activity instances by correlating each individual location sensor event $e \in E_L$ to an activity instance $i \in \mathbb{N}$. This results in an *instance log* E_I in which, beyond the requirements for E_L, each event $e_i \in E_I$ is additionally assigned an activity instance that can be retrieved with $instance(e) \in \mathbb{N}$. The main goal of this step is to produce traces such that each trace can be associated with an activity instance. We assume every recorded event in the raw data is caused by an entity. In order to determine which entity $n \in N$ caused which event, the raw event location data is assigned to the respective entities. Eventually the trace of each entity is divided into smaller sub-traces (cases) that contain only one single activity: we denote this as *sensor case slicing*. Here, also an approach for *entity detection* is required.

Entity detection. In a setting with sensors providing only information whether an object is present or absent, a distinction between entities is not possible. However, if we know the relative location of the sensors to each other, our weighted average distance approach can be implemented and distinguish between multiple entities. The very first time any of the sensors detects the presence of an entity is the beginning of the first entity's trace. For every subsequent sensor activation, we have to decide which entity caused the activation of a sensor. Each time a sensor is activated, we calculate which already registered entity is closest to the current sensor activation, based on the entities' last known position. If no entity is close enough, the algorithm assumes that a new entity has entered the observed area and creates a new trace for this new entity. Both the proximity threshold and the maximum number of entities are parameters that can be manually adjusted based on the scenario. This straightforward implementation works well if entities always keep a certain distance to others. But as soon as various entities cross paths in a spot that is only covered by a single sensor, this method will not be able to correctly assign the sensor activations after the entities moved on, since the newly activated sensor has the same distance to every entity in that single spot. This limitation can be overcome, by assuming, entities will preserve their direction of motion and predict where entities are headed by also considering the entities' previous locations combined with a decay function in the distance function.

Sensor case slicing. During its presence in the observed area, the entity executes most likely more than a single activity. To identify meaningful activities from the continuous recording (what is called a "long trace"), an appropriate separation into smaller sub-traces, called cases, is required. We have to divide the traces here, because we are identifying and clustering activities by their sensor-activation-signature, therefore the sub-traces can only contain one single activity.

In concrete terms, in our approach, a long trace is cut into sub-traces of a predefined fixed length. Depending on the application, the optimal fixed length

might be different. Our implementation incorporates a grid search, comparing the results for different sub-trace lengths, to maintain flexibility. The challenge is to avoid sub-traces that are too short and contain too little sensor-data to extract meaningful activities. But at the same time, the sub-traces cannot be too long, as a too-long sub-trace may consist of multiple activities.

3.2 Activity Discovery

Having obtained the instance log E_S, we aim to infer a set of process activities A that are likely to have generated the raw sensor events. The outputs of this step in our approach are a set of activities A. Each activity $a \in A$ has both an activity label $label(a)$ as well as a process model describing the low-level behaviour of that activity a in terms of events on the sensor-level. The main challenge in this part of the approach is to determine a good division of activity instances into clusters, i.e., an *activity clustering* where each of the clusters should represent a distinct activity on the process level. This refers not only to the clustering itself but also to finding a good number of clusters. Furthermore, a suitable *activity labelling* needs to be found.

Activity clustering. Independent of the implemented clustering technique, the objective remains the same: Find similar sub-traces and group them. For this, we used a *Self-Organising Map* (SOM) clustering and k-means. The challenge with the discovery of similarities is to find a criterion to define the similarity between sub-traces. Usually, in SOM this is achieved by calculating the euclidean distance between vectors. However, this is challenging if sensors have arbitrary label names. We experimented with three alternative representations of the traces: First, we counted how often each sensor is activated in a trace. Second, we counted for how long each sensor is activated for in a trace. And third, we combined both the quantity method and time method in one vector. The third representation retains the most information of the original trace and is, therefore, the preferred choice.

Activity labelling and Validation. Having discovered clusters of similar traces corresponding to distinct activities, we still lack insights into the kind of activity that may be represented by each cluster. Also, it may be challenging to judge the quality of the obtained clustering. We assume that activity labelling generally requires a human-in-the-loop with appropriate domain knowledge. Thus, we propose to discover a process model based on the events of each cluster by using, e.g., Inductive Miner. Then, the quality of the process model is evaluated based on the F1-score combination of the common *fitness* and *precision* measure. The core idea is that these interpretable process models make our method suitable for complex processes and the quality measure can be used to validate the clustering result. Having access to the process models and their quality evaluation a domain expert can interpret, validate and label each cluster with an appropriate activity label $a \in A$.

3.3 Event Abstraction

The third step of our approach combines the sub-traces yielded by the *event correlation* step (Section 3.1) and the activity clusters detected in the *Activity Discovery* step (Section 3.2). This results in a process event log E_P that groups together events from the original location sensor event log E_L to process events $e_p \in E_P$. For each process event e_p we can obtain the following attributes: $time(e_P) \in \mathbb{N}$, $activity(e_P) \in A$, $entity(e_P) \in T$, and $transition(e_P) \in \{start, completed\}$. Thus, each process event refers to a specific high-level process activity and indicates a transition in the transactional life-cycle, i.e., whether the activity instance has been started or completed.

3.4 Process Discovery

Having promoted the raw location sensor events E_L to the level of activity instances, our process event log E_P fulfils almost all requirements for high-level process discovery. Anyway, still missing are process cases that are meaningful to our analysis goal. Identifying process cases is highly dependent on the particular circumstance. In our application scenario, we propose to focus on re-occurring behaviour of an entity starting with a specific activity (e.g., entering the smart home). Based on our event correlation step (Section 3.1), we build a separate trace for each entity. Then, the potentially very long trace referring to a single entity is subdivided into multiple traces by dividing it into separate traces each time the activity of interest occurs. To discover a meaningful process model, we have to assume that regular and routine behaviour is observable. As a starting point, an activity has to be selected that most likely will be the origin of the routine behaviour such as *entering the observed area*. Finally, an overall process model is discovered using a standard technique, e.g., Inductive Miner [26]. The final output is a process model reflecting the observed behaviour of the entities aggregated only from raw sensor data.

4 Evaluation

4.1 Set-up

We evaluated our approach on the publicly available CASAS data-set, which contains raw sensor data from a smart home environment [2]. The CASAS data fulfils the two requirements of our approach: it contains the timestamps and location information of sensor events. The data was recorded in a smart home test-bed with two residents and a house equipped with 51 motion sensors. Figure 2 shows the house plan and the positions of the motion sensors. Each motion sensor generates low-level events, where each sensor entry is tagged with a timestamp, the sensor ID and the binary sensor value (active / not active). We extracted sensor data from 7 consecutive days (02/05–09/02/2010) from the `20-Kyoto-2-Daily life, 2010-2012` data set.

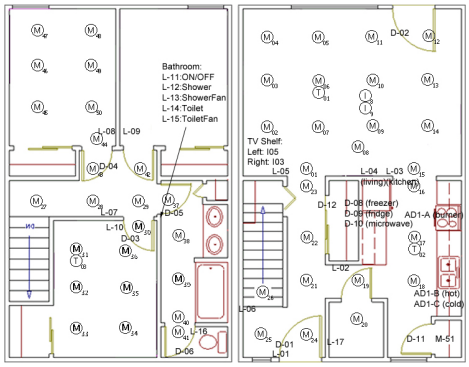


Fig. 2: Sensor layout of an apartment in the CASAS project [2].

We applied our method for different values of parameters such as sub-trace length, number of clusters and the similarity measure used. We used grid search to identify best parameter values for the clustering based on the average combined fitness [27] and precision [28] (F1-score) obtained for the process models discovered for each cluster of high-level events. We employ standard filtering techniques (most frequent traces and activities) used in process mining to focus on the dominant behaviour in each cluster. We compared the proposed SOM clustering with k-means clustering based on the same similarity measures. The implementation of step 1 and 2 is openly accessible [6] We used PM4Py 1.1.1 and heuristicmineR for the process discovery and evaluation.

Having discovered activities and obtained traces based on the idea to discover re-occurring behaviour starting with the same activity (Section 3.4), we applied Heuristics Miner to discover a process model of the behaviour. Based on the spatial layout of the smart home (Figure 2), we choose to create traces that start with the activity *Walk entrance/stairs/storage* as the entry point into the house. Heuristics Miner was selected as we expect the inhabitants of the smart home environment to show a lot of infrequent behaviour, for which Heuristics Miner has shown to be appropriate [29].

4.2 Results & Discussion

Figure 3 shows the results of our grid search. We experimented with trace lengths ranging from four to twelve. Shorter trace lengths generally lead to a better F1-score. However, we need to impose a minimal trace length since traces consisting only of a single event would trivially lead to the discovery of process models with perfect fitness and precision. In our case, less than four events did not allow to infer a set of meaningful activities.

Evaluating sample data has shown that considering both the frequency of activation as well as the duration of the activations as a similarity measure (the

[6] `https://github.com/d-o-m-i-n-i-k/Process-Model-Discovery-public`.

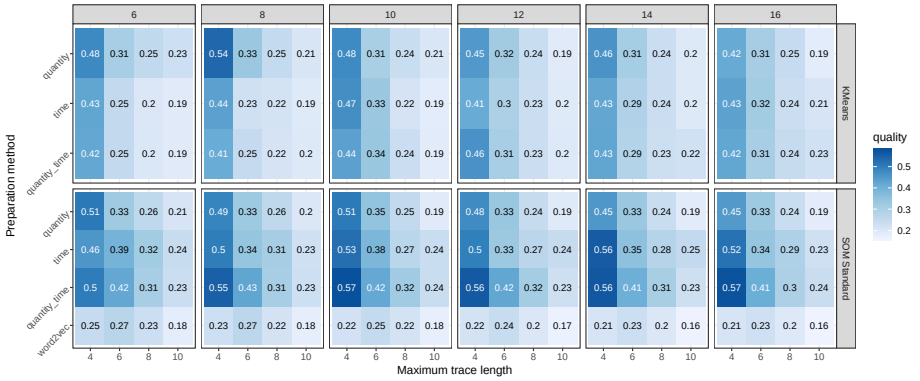


Fig. 3: Average F1-score for process models discovered for the clusters based on six different cluster sizes (6-16), five different maximum trace lengths (4-12), four vector preparation methods and two clustering algorithms.

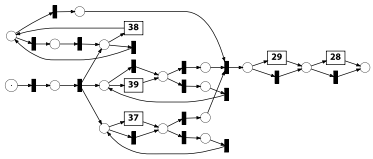


Fig. 4: Example of a Petri net discovered using Inductive Miner for a cluster in the *Activity Discovery* step.

method *quantity_time*) yields superior results, compared to only regarding one aspect. When choosing too few clusters or too many, the quality score decreases. In turn, choosing too many clusters may lead to several clusters representing the same activities, which should have been grouped. We also qualitatively evaluated the clustering by manually inspecting and labelling some of the results.

For example, the Petri net discovered by Inductive Miner on a cluster shown in Figure 4 is a reasonable candidate. The three sensors that can be activated simultaneously are all located in the bathroom. The subsequent sensors `M29` and `M28` are located in the hall with `M28`, which is furthest from the bathroom. From this example process model, it is reasonable to infer that this cluster refers to activities where the entity spends some time in the bathroom and then left the room. Overall, the similar results are obtained for 10 and 16 clusters with a trace length of 4 and using our proposed *quantity&time* vectorisation approach.

We grouped the activity instances of the best clustering results (16 clusters) into traces at the level of process instances. Afterwards we filtered the resulting event log to only retain traces of a length in the range of 5 to 25 events. This yields a log with 5898 events grouped in 273 traces with an average length of 21.6. The application of Heuristics Miner with a dependency threshold of 0.8 and a frequency threshold of 10 returns the Causal net dependencies shown in

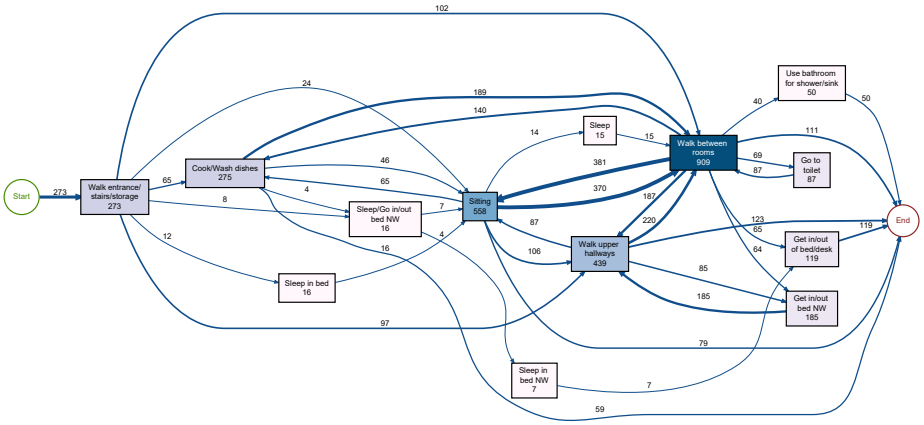


Fig. 5: Causal net discovered with the Heuristics Miner on the obtained process event log.

Figure 5. The activity in entrance area of the house marks the starting point of our Causal net. The activities that can mostly be observed after the entry activity are *walking between the rooms*, *walking in the upper hallways* and *going to the kitchen to cook or wash the dishes*. After cooking the dishes it often occurs that the resident would walk between the rooms to sit down, presumably to eat in the living room.

4.3 Limitations

A drawback of our method is the assumption of continuous movement in the event correlation step (Section 3.1). As soon as the motion at the rendezvous location is more than just a mere passing by, our approach might not return the desired results. Additionally, the entity recognition could be improved by using more sophisticated methods, e.g. hidden Markov models that have already shown promising results in differentiating people from one another [30]. Moreover, the sensor case slicing mechanism could take variable sub-trace length into consideration, i.e., depending on the activity, the number of involved events, and therefore the sub-trace length may vary. For example, the activities *sleeping*, *cooking* and *washing hands* are activities with a distinctive difference in the number of involved events.

5 Conclusion

IoT environments generate a large amount of data, predestined for further analysis. Process mining can give valuable insights into how real-life activities perform when extracting meaningful activities instances from raw sensor events. This paper combined unsupervised learning in the form of clustering and process mining,

to discover activities and process models from motion sensors. We evaluated our approach by comparing the obtained model quality for several clustering techniques on a publicly available data-set in a smart home scenario and found it to be superior. To fully relieve domain experts from process modelling and to automate the process of model discovery, we believe that an accurate approach for entity centricity is imperative. For this, future tasks are to fuse heterogeneous sensor events as input for high-level aggregation, to take into account other vectorisation methods such the shortest path distance between sensors (i.e., relational or pair-wise distances only) to better disambiguate between residents and to apply non-end-to-end process discovery methods such as Local Process Model discovery [22]. In further research, we plan to include spatial information, like room layouts in smart homes, into our approach as well as implement variable trace lengths and experiment with other machine-based learning techniques to further improve the discovered process models.

References

1. van der Aalst, W.M.P.: Process Mining - Data Science in Action, Second Edition. Springer (2016)
2. Rashidi, P., Youngblood, G.M., Cook, D.J., Das, S.K.: Inhabitant guidance of smart environments. In: Lecture Notes in Computer Science. Volume 4551 of LNCS. Springer Berlin Heidelberg (2007) 910–919
3. Nentwig, M., Stamminger, M.: A method for the reproduction of vehicle test drives for the simulation based evaluation of image processing algorithms. In: 13th International IEEE Conference on Intelligent Transportation Systems, IEEE (September 2010) 1307–1312
4. Zhang, M., Sawchuk, A.A.: Motion primitive-based human activity recognition using a bag-of-features approach. In: Proceedings of the 2nd ACM SIGHIT symposium on IHI '12. IHI '12, ACM Press (2012) 631–640
5. Diete, A., Sztyler, T., Weiland, L., Stuckenschmidt, H.: Improving motion-based activity recognition with ego-centric vision. In: PerCom Workshops 2018, IEEE Computer Society (2018) 488–491
6. Sztyler, T., Carmona, J., Völker, J., Stuckenschmidt, H.: Self-tracking reloaded: Applying process mining to personalized health care from labeled sensor data (2016)
7. Larue, G.S., Rakotonirainy, A., Pettitt, A.N.: Predicting reduced driver alertness on monotonous highways. IEEE Pervasive Computing **14**(2) (April 2015) 78–85
8. Weerdt, J.D., vanden Broucke, S., Vanthienen, J., Baesens, B.: Active trace clustering for improved process discovery. IEEE Transactions on Knowledge and Data Engineering **25**(12) (December 2013) 2708–2720
9. Choi, S., Kim, E., Oh, S.: Human behavior prediction for smart homes using deep learning, IEEE (August 2013) 173–179
10. Gallicchio, C., Micheli, A.: Experimental analysis of deep echo state networks for ambient assisted living. In Bandini, S., Cortellessa, G., Palumbo, F., eds.: Proceedings of the Third Italian Workshop on AI for Ambient Assisted Living. Volume 2061 of CEUR Workshop Proceedings., CEUR-WS.org (2017) 44–57
11. Wang, A., Chen, G., Shang, C., Zhang, M., Liu, L.: Human activity recognition in a smart home environment with stacked denoising autoencoders. Volume 9998. (June 2016) 29–40

12. Wang, J., Chen, Y., Hao, S., Peng, X., Hu, L.: Deep Learning for Sensor-based Activity Recognition: A Survey. arXiv e-prints (2017)
13. Lee, S., Lin, F.J.: Situation awareness in a smart home environment. In: 2016 IEEE 3rd World Forum on Internet of Things (WF-IoT). (December 2016) 678–683
14. Dernbach, S., Das, B., Krishnan, N.C., Thomas, B.L., Cook, D.J.: Simple and complex activity recognition through smart phones. In: Intelligent Environments, IEEE (2012) 214–221
15. Sztyler, T.: Sensor-based human activity recognition: Overcoming issues in a real world setting. PhD thesis, University of Mannheim, Germany (2019)
16. Nguyen, H.D., Tran, K.P., Zeng, X., Koehl, L., Tartare, G.: Wearable sensor data based human activity recognition using machine learning: A new approach. ArXiv **abs/1905.03809** (2019)
17. Koschmider, A., Mannhardt, F., Heuser, T.: On the contextualization of event-activity mappings. In: BPM 2018 Workshops. LNBIP, Springer (September 2018)
18. Leotta, F., Mecella, M., Mendling, J.: Applying process mining to smart spaces: Perspectives and research challenges. In: International conference on advanced information systems engineering, Springer (2015) 298–304
19. van der Aa, H., Leopold, H., Reijers, H.A.: Checking process compliance on the basis of uncertain event-to-activity mappings. In: Advanced Information Systems Engineering. Springer International Publishing (2017) 79–93
20. Tax, N., Sidorova, N., Haakma, R., van der Aalst, W.: Mining process model descriptions of daily life through event abstraction. In: Intelligent Systems and Applications. Springer International Publishing, Cham (December 2017) 83–104
21. Leotta, F., Mecella, M., Sora, D.: Visual process maps: a visualization tool for discovering habits in smart homes. Journal of Ambient Intelligence and Humanized Computing **11**(5) (January 2019) 1997–2025
22. Mannhardt, F., Tax, N.: Unsupervised event abstraction using pattern abstraction and local process models. In: RADAR+EMISA@CAiSE. Volume 1859 of CEUR Workshop Proceedings., CEUR-WS.org (2017) 55–63
23. Alharbi, A., Bulpitt, A., Johnson, O.: Towards unsupervised detection of process models in healthcare. Studies in health technology and informatics **247** (January 2018) 381–385
24. 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. Journal of Ambient Intelligence and Smart Environments **11**(2) (2019) 165–182
25. Brzychczy, E., Trzcionkowska, A.: Process-oriented approach for analysis of sensor data from longwall monitoring system. In: Advances in Intelligent Systems and Computing. Springer International Publishing (August 2018) 611–621
26. Leemans, S.J.: Robust Process Mining With Guarantees. PhD thesis (2017)
27. van der Aalst, W.M.P., Adriansyah, A., van Dongen, B.F.: Replaying history on process models for conformance checking and performance analysis. WIREs Data Min Knowl Discovery **2**(2) (2012) 182–192
28. Munoz-Gama, J., Carmona, J.: A general framework for precision checking. International Journal of Innovative Computing, Information and Control (IJICIC) **8**(7) (2012) 5317–5339
29. Augusto, A., Conforti, R., Dumas, M., Rosa, M.L., Maggi, F.M., Marrella, A., Mecella, M., Soo, A.: Automated discovery of process models from event logs: Review and benchmark. IEEE TKDE **31**(4) (April 2018) 686–705
30. Guo, P., Miao, Z.: Multi-person activity recognition through hierarchical and observation decomposed HMM. In: 2010 IEEE International Conference on Multimedia and Expo, IEEE, IEEE (July 2010) 143–148

Unsupervised Event Abstraction in a Process Mining Context: A Benchmark Study

Greg Van Houdt (✉)[1], Benoît Depaire[1], and Niels Martin[1,2,3]

[1] Research group Business Informatics, UHasselt - Hasselt University, Belgium
greg.vanhoudt@uhasselt.be
[2] Research Foundation Flanders (FWO), Belgium
[3] Data Analytics Laboratory, Vrije Universiteit Brussel, Belgium

Abstract. Due to the rise of IoT, event data becomes increasingly fine-grained. Faced with such data, process discovery often produces incomprehensible spaghetti-models expressed at a granularity level that doesn't match the mental model of a business user. One approach is to use event abstraction patterns to transform the event log towards a more coarse grained level and to discover process models from this transformed log. Recent literature has produced various (partial) implementations of this approach, but insights how these techniques compare against each other is still limited.
This paper focuses on the use of Local Process Miner and Combination based Behavioural Pattern Mining to discover event abstraction patterns in combination with the approach of Mannhardt et al. [15] to transform the event log. Experiments are conducted to gain insights into the performance of these techniques. Results show that the results are very limited, with a general decrease in fitness and precision and only a minimal improvement of complexity. Results also show that the combination of the process discovery algorithm and the event abstraction pattern miner matters. In particular, the combination of Local Process Miner with Split Miner seems to improve precision.

Keywords: Process Mining · Unsupervised Learning · Event Log · Abstraction.

1 Introduction

Process Discovery focuses on the discovery of the process flow from an event log [2], in order to gain insights in the real execution of a business process [1]. However, when event logs are recorded at a fine-grained level, the activities in the discovered process model become increasingly less recognisable to the business users. Furthermore, fine-grained event data, often result in incomprehensible spaghetti-models [22].

Against this background, [14] introduced a pattern-based approach to augment low-level events to higher-level activities, resulting in more insightful process models. Their original approach requires domain experts to provide event

abstraction patterns which map low-level events to higher-level activities, which are subsequently used to transform the event log.

More recently, Mannhardt and Tax [16] studied the use of Local Process Models to learn these event abstraction patterns from data. Even more recently, [3] introduced a new unsupervised technique to discover event abstraction patterns that are compact and maximal, increasing the options to apply the technique proposed in [16] in an unsupervised manner. This raises the question how well these two options perform with the end goal in mind, i.e. to transform the log such that a process model is discovered which is more comprehensible and remains properly fitting and precise.

This paper describes a benchmark study of LPM and COBPAM in combination with the approach in [16], focused on their capabilities to obtain models of lower complexity without sacrificing fitness and precision. Furthermore, performance differences between these two approaches were explored in order to identify underlying mechanisms at work. This resulted in following contributions:

- In contrast to previous studies, this study also considers the impact of event pattern abstraction on the understandibility of the final process models, approximated by a broad set of complexity measures.
- This study provides an empirical comparison between LPM and COBPAM in combination with the method presented in [16], providing initial suggestions which of both event abstraction pattern miners performs best.
- This work provides empirical insights into the interaction between the process discovery algorithm and the event abstraction pattern miner with respect to their conjoint impact on fitness, precision and comprehensibility.

The remainder of this paper is structured as follows. Sect. 2 gives an outline of related work in the domain. Next, Sect. 3 defines the methodology for the benchmark study, as well as elaborates on the experimental design. Sect. 4 then gives an overview of the experiment's results before Sect. 5 concludes the paper.

2 Related work

This paper builds further on the work in [16], which studied the use of LPMs to discover event abstraction abstraction patterns in combination with the approach in [14]. However, their work was slightly different then ours, as they only focused on fitness and precision, whereas we also take process model complexity into account.

LPM [23] is can be used to discover event abstraction patterns in an unsupervised manner. It extends frequent pattern mining techniques to more complex patterns and aims to describe frequent behaviour in an event log in local patterns. In [21], LPM is extended with utility functions and constraints to mine more meaningful patterns, while [9] shows how high quality sets of a limited number of LPMs can be constructed. However, both latter approaches require

some kind of domain expert interaction, which puts them outside the scope of our study.

Inspired by LPM, Acheli et al. [3] designed the Combination based Behavioural Pattern Mining (COBPAM) approach. It exploits a partial order on potential patterns to discover only those that are compact and maximal, i.e. least redundant.

Other unsupervised techniques mentioned in [25] are: global trace segmentation [11], HLPM-Mine [10], Bose et al. [8], Alharbi et al. [6], RefMod-Miner [18] and the work of Sánchez-Charles et al. [19].

This study focuses on LPM and COBPAM as the setup under consideration employs the approach in [16], which requires a defined process pattern between the low-level events in the event abstraction pattern. Another approach producing compatible patterns for this setup would be the RefMod [18] miner. Unfortunately, as no public implementation was available for this technique, it was not considered in this study.

It is worth noting that the combination of event abstraction pattern miners and the technique in [16] is not the only possibility to discover higher-level process models from low-level event data. For example, [20] presents a framework designed to transform location sensor data to an event log via interaction mining that business users can understand.

3 Methodology and Experimental Setup

This study takes the following algorithmic problem class as a starting point:

> A process model discovered from an event log is too complex to understand because the event log is too fine-grained. What is needed, is a technique which augments the event log to a higher abstraction level such that a process model discovered from this transformed event log results in less complex process models which are still correctly representing the underlying process.

This paper considers an algorithmic design to tackle this problem based on the approach in [14] in combination with two unsupervised abstract pattern discovery techniques, i.e. LPM [23] and COPBAM [3].

The quality of the algorithm design is defined on three criteria. Firstly, we want the process model discovered from the transformed log to be more comprehensible. The second and third criteria state that the model discovered from the transformed log should remain fitting and precise with respect to the original data.

3.1 Evaluation Method

To evaluate the comprehensibility of the model discovered from the transformed log, we use complexity as a proxy, which has been shown to be inversely related

to the model understandability [17]. In total, ten complexity metrics were used, covering the four complexity dimensions identified in [13]. [4] These complexity metrics are computed for the process models discovered from the transformed event logs.

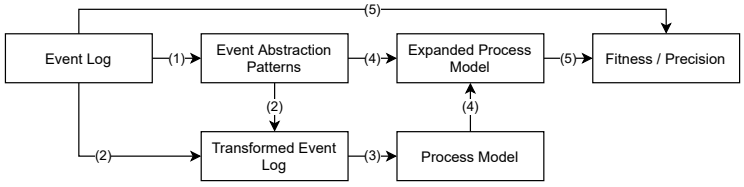


Fig. 1: Method to evaluate fitness and precision: (1) event abstraction patterns are discovered from the original event log, (2) the event log is transformed to higher abstraction level, (3) a process model is discovered, (4) the process model is expanded with the event abstraction patterns to the original granularity level, (5) precision and fitness are computed.

To evaluate the fitness and precision of the model discovered from the transformed event log with respect to the original event log, the same approach as in [16] was used. First, event abstraction patterns are discovered from the original event log. These event abstraction patterns map a local process pattern, defined at the original granularity level, to a higher level activity. Second, these patterns are used to transform the event log. Third, a process model is discovered from this transformed event log. Fourth, the event abstraction patterns are used to expand the process model into an expanded process model which is at the abstraction level of the original event log. This is done by replacing the higher-level activities by its corresponding local pattern. Fifth, the original event log is compared against the expanded process model to calculate fitness and precision values. Fitness is measured by the alignment-based fitness measure [4] and precision is measured with the alignment-based ETC precision measure [5].

3.2 Data

Six publicly available real-life event logs[5] with different characteristics are used in this study. Table 1 illustrates the variation among the logs in terms of number of events, number of activities, number of cases and number of distinct traces.

Regarding the BPI challenge 2019 log, similar to [7], a sample of the event log was taken for performance reasons. We preserved case variants containing at least 50 cases, leaving us with 71% of the events.

[4] This is done via the R package `understandBPMN` [13]
[5] The event logs were extracted from the 4TU Centre for Research Data in May 2020.

Table 1: Event log characteristics.

	# events	# activities	# cases	# distinct traces
Road Traffic Fine Management	561.470	11	150.370	231
Hospital Billing	451.359	18	100.000	1.020
Sepsis Case	15.214	16	1.050	846
BPI 2019	1.135.258	27	224.768	192
BPI 2020 - Request For Payment	36.796	19	6.886	89
BPI 2020 - Domestic Declarations	56.437	17	10.500	99

3.3 Experimental Design

Every experiment consists of controlled variables that are of interest to the study. In our setup, the controlled variables are the event abstraction pattern miners and the process discovery algorithms used.

Two event abstraction pattern miners were considered in this study, i.e. LPM and COBPAM. For LPM the ProM implementation [24] was used with default parameter settings, except for the maximum number of transitions (5) and the number of patterns to discover (10). From the 10 patterns discovered, the top 3 according to the model ranking were selected, ignoring patterns subsumed by other patterns. For COBPAM, the ProM implementation [24] is used with support threshold, language fit threshold and maximum dept set to respectively 0.7, 0.7 and 2 in accordance to the original work [3]. Patterns are sorted by support value and the top 3 patterns which are not subsumed by other patterns are selected.

Furthermore, two discovery algorithms were used, i.e. the split miner [7] and inductive miner infrequent [12]. Both are configured with their default values. This means there is a conversion step from BPMN to Petri net in the case of split miner[6]. The split miner is implemented as stand-alone Java application, while the inductive miner is accessed via the ProM framework [24].

For all four combinations of the 2 event abstraction pattern miners and 2 discovery algorithms, the following experiment was performed:
For each event log, an initial process model was discovered and corresponding complexity, fitness and precision values were computed. These values serve as the baseline. Next, event abstraction patterns were mined from the event log and the top three were used to transform the event log to a higher abstraction level using the approach in [14]. The ProM implementation was used [24] and low-level events that were not mapped to higher-level events were kept in the transformed log. All other parameters were set at their default values. Finally, a new process model was discovered from the transformed event log and complexity, precision and fitness values were computed as described in Sect. 3.1. These measures can be compared against the baseline values to evaluate the impact of a specific event abstraction pattern miner for a given event log and discovery technique.

[6] Done via the *convert BPMN diagram to Petri Net (Control Flow) plug-in in ProM.*

4 Empirical Results

This section will explain the results of our experiment per quality dimension. In total, the six event logs, three abstraction levels, two miners and 12 metrics for each model, resulted in 432 metrics. The raw result set is available online[7].

4.1 The Effect of Abstraction on Model Complexity

Complexity is measured by ten metrics in total. These are cognitive weight, token split, connector heterogeneity, control flow complexity, sequentiality, cyclicity, diameter, depth, density and coefficient of network connectivity [13]. From these ten, token split, connector heterogeneity, control flow complexity, sequentiality, cyclicity and the coefficient of network connectivity did not improve on average due to event abstraction for any activity pattern miner, regardless of the process model miner. The results concerning the remaining four are not uniform, however, as is shown in Table 2. The table describes, for each combination of miner and complexity metric, the number of event logs for which an improvement was observed and the average change. Note that a negative delta is considered an improvement, i.e. a reduction in complexity.

Table 2: Abstraction impact on cognitive weight, depth, density and diameter.

		Split miner		Inductive miner	
		# Improvements	Delta	# Improvements	Delta
Cognitive weight	LPM	3/6	-3.37%	2/6	2.54%
	COBPAM	4/6	-2.88%	2/6	3.45%
Depth	LPM	2/6	-7.14%	0/6	33.33%
	COBPAM	4/6	-35.71%	1/6	22.22%
Density	LPM	1/6	12.18%	3/6	-1.52%
	COBPAM	1/6	16.35%	3/6	-0.42%
Diameter	LPM	1/6	4.92%	3/6	-5.41%
	COBPAM	2/6	-2.19%	2/6	2.70%

Cognitive weight, which is the weighted sum of gateways and activities, seems to improve for both LPM and COBPAM when paired with the split miner. Depth, the amount of split minus join gateways, behaves in a similar fashion. The inductive miner has a tendency to generate more (parallel) gateways than the split miner, and this effect is still present after abstraction. However, it is important to note that the baseline values for the inductive miner are already lower than the split miner's.

Density represents the percentage of sequence flows which are present compared with the theoretical maximum number of sequence flows. It shows the

[7] https://github.com/gregvanhoudt/UnsupervisedEventAbstraction

inverse behaviour of cognitive weight and depth, improving when paired with the inductive miner. However, the improvement here is much smaller than the deterioration with the split miner. A small density value indicates that the process is more sequential. The models substantiate this, as the split miner has a tendency to loop back to previous gateways to allow for repetitive behaviour, creating more sequence flows. In that regard, it is possible for depth to improve while density worsens.

The diameter metric only seems to improve, on average, for the combinations COBPAM-split miner and LPM-inductive miner. Also, even if diameter improves on average, the value decreases for the majority of the logs. Given that the results do not seem to correlate with a discovery miner or activity pattern miner, specific conclusions cannot be drawn for this metric. Further experimentation is required to obtain more conclusive results.

When considering all complexity measures simultaneously, we count 15 and 19 improvements for LPM and COBPAM, respectively. Although the difference is small, this seems to indicate COBPAM is slightly better in reducing complexity than LPM, independent of the process discovery algorithm. In general, we can conclude that, in our experimental setting, pattern-based event abstraction does not reduce the complexity of newly learnt models. However, caution is advised as we limited ourselves to only three patterns for each abstraction. Inserting additional patterns might have a positive impact on complexity. Keep in mind this will probably also impact fitness, and potentially precision.

4.2 The Effect of Abstraction on Model Fitness and Precision

The second facet of the study, the accuracy of process models, is measured by fitness and precision. Table 3 summarises the outcomes. A positive delta is now considered an improvement. Note that for precision - LPM - Inductive miner, we were unable to compute two values.

Table 3: Abstraction impact on fitness and precision.

		Split miner		Inductive miner	
		# Improvements	Delta	# Improvements	Delta
Fitness	LPM	0/6	-23.86%	1/6	-13.00%
	COBPAM	0/6	-32.04%	0/6	-22.93%
Precision	LPM	4/6	0.67%	1/4	-34.25%
	COBPAM	1/6	-4.78%	1/6	-26.00%

Regarding fitness, the data shows there is a severe negative effect: only one improvement is observed. Performing a t-test at the 5% significance level, the only insignificant difference was for LPM in combination with the inductive miner. Of course, the numbers have to be nuanced as the split miner generated a baseline of nearly perfect fitness, so an increase will be very difficult

to accomplish. However, the magnitude of the decrease makes clear automated pattern-based abstraction negatively affects the fitness of new high-level process models.

One possible explanation is the overlap between activity patterns. Recall that fitness can only be calculated after the high-level model is expanded to again include the low-level event classes. This means one event class can now be present at multiple locations in the model. This can result in the obligation of an event class to be executed multiple times according to the high-level process models, which is not the case on the lower level. Also, the overlaps make it unclear which low-level event belongs to which high-level activity [16], generating potential confusion during the abstraction of the event log.

The precision metric also shows clear evolutions, although not as uniform as fitness. In fact, the average precision metric for LPM in combination with the split miner increased. However, the differences were not significant at the 5% significance level. A potential reason for decreases of precision is that we assume parallel relations between activity patterns. Should patterns overlap, it could be more reasonable to state that two patterns cannot co-exist. If two overlapping high-level patterns are present in the model, this is an introduction of additional behaviour. On the other hand, the goal of event log abstraction is hiding / grouping low-level behaviour, which should have a positive influence on precision.

In general, fitness only increased once for LPM without observing any improvements for COBPAM. The precision metric improved 5 and 2 times for LPM and COBPAM respectively, with the majority of improvements located at LPM-split miner in particular. Results suggest that LPM performs better when interested in fitness and precision of abstracted models.

4.3 Discussion

Overall, the study shows that the use of event abstraction pattern miners to transform an event log with the purpose of discovering less complex process model with good fitness and precision, has limited success. On average, it seems that this approach, using either LPM or COPBAM, results in a decrease of fitness and precision, with only limited effect on complexity.

Fitness typically takes a hit when abstracting the event log, which is not completely unexpected as abstraction patterns hide complex behaviour which can no longer be accounted for by the miner. It is also remarkable that the impact on fitness appears to be correlated to the process discovery algorithm. Before abstraction, the split miner produces the best-fitting models. After abstraction, however, this fitness drops heavily, to the extent that the inductive miner produces better-fitting models at that stage.

Precision also has a tendency to deteriorate, with the exception for the combination of LPM with split miner. For this combination, in the majority of the cases we saw an improvement of precision and the average effect was also positive. It is remarkable that this result is not observed for the combination with COBPAM and that LPM cannot reproduce these effects with inductive miner.

This again confirms the pattern that there is some kind of interaction between the process discovery algorithm and the event abstraction pattern miner.

As for complexity, for most of the measures no clear improvement was observed. The only pattern that could be distinguished, which supports the goal of this approach, is the slight improvement of cognitive weight and depth for split miner and the improvement of density for inductive miner. Again, these results hint at an interaction between the discovery miner and abstraction pattern miner.

Overall, we can conclude that this approach combined with LPM and COBPAM has limited results. Future research will be needed to improve these results in order to make them impactfull enough for practical use. Based on our empirical analysis, a potential direction for future research is to delve into the interaction between the discovery algorithm and the abstraction pattern miner. It is clear that there are mechanisms at work and understanding these could open up avenues for improved algorithms. Another path worth investigating is the automatic discovery of how activity patterns interrelate. The approach in [16] has parameters which define which patterns can or cannot co-exist and in what type of interrelation. The current event abstraction pattern miners do not provide this type of information.

Finally, based on these mixed empirical results, one must be careful to draw strong conclusions with respect to the performance of LPM versus COBPAM. One might suggest that both approaches are competitive to each other, with the exception of the combination of split miner with LPM, which actually appears to improve the precision of the models on average. As with respect to reducing complexity, COBPAM seems to have a small edge over LPM, albeit too small to make conclusive statements.

4.4 Limitations

This experiment can be extended in several ways. First of all, a new approach to evaluate LPMs was recently proposed [9], which is not implemented in our work yet. This new evaluation acknowledges the excessive amount of overlapping patterns and disregards confidence and determinism as quality measures. For COBPAM, we only have access to support and language fit scores. A more advanced scoring and selection technique of activity patterns could have improved the experiment, obtaining less overlapping patterns as with the meaningful LPMs [9].

On the other hand, the current study fixes the number of activity patterns that are taken into consideration to three. Mannhardt and Tax [16] concluded that the optimal number of patterns varied per event log. No doubt the same applies to this study.

Next, this experimental setup uses six event logs. Each of them returns six process models: a low-level, a LPM-abstracted and a COBPAM-abstracted model for both the split miner and the inductive miner. Therefore, this study compares 36 process models. To obtain a larger number of observations to draw conclusions from, this number of event logs can easily be increased.

Finally, as discussed in Sect. 2, the RefMod-Miner also satisfies the requirements to take part of this experiment, yet no public implementation is available.

5 Conclusion

In this paper, local process models and the combination based behavioural pattern mining approach are put against each other in unsupervised event log abstraction. The goal was to produce process models at a higher abstraction level with better comprehensibility, while still being well-fitting and properly precise.

However, the experiments show only limited results. While some aspects of complexity show possibilities for improvements, they seem tied to the process model miner. Fitness gets a significant hit overall and precision only tends to improve for the combination of LPM and Split Miner.

Future research is required with respect to the interactions between activity pattern miners and process discovery algorithms. This could allow for more accurate abstraction techniques. Also, discovering meaningful and more precise activity patterns is an interesting research track. But perhaps more importantly, the possibility to discard our assumption about only parallel inter-pattern relations must be explored. Being able to learn this from data could greatly improve the abstraction quality.

References

1. van der Aalst, W.M.P.: Process Mining: Data Science in Action. Springer, Berlin, Heidelberg (2016)
2. van der Aalst, W.M.P., Weijters, T., Maruster, L.: Workflow mining: discovering process models from event logs. IEEE Transactions on Knowledge and Data Engineering **16**(9), 1128–1142 (2004)
3. Acheli, M., Grigori, D., Weidlich, M.: Efficient Discovery of Compact Maximal Behavioral Patterns from Event Logs. In: Giorgini, P., Weber, B. (eds.) Advanced Information Systems Engineering, vol. 11483, pp. 579–594. Springer International Publishing, Cham (2019), series Title: Lecture Notes in Computer Science
4. Adriansyah, A., van Dongen, B.F., van der Aalst, W.M.: Conformance checking using cost-based fitness analysis. In: 2011 ieee 15th international enterprise distributed object computing conference. pp. 55–64. IEEE (2011)
5. Adriansyah, A., Munoz-Gama, J., Carmona, J., van Dongen, B.F., van der Aalst, W.M.: Measuring precision of modeled behavior. Information systems and e-Business Management **13**(1), 37–67 (2015)
6. Amirah, A., Andy, B., A, J.O.: Towards Unsupervised Detection of Process Models in Healthcare. Studies in Health Technology and Informatics pp. 381–385 (2018), place: Netherlands Publisher: IOS Press
7. Augusto, A., Conforti, R., Dumas, M., La Rosa, M., Polyvyanyy, A.: Split miner: automated discovery of accurate and simple business process models from event logs. Knowledge and Information Systems **59**(2), 251–284 (May 2019)
8. Bose, R.J.C., Van der Aalst, W.M.: Abstractions in process mining: A taxonomy of patterns. In: International Conference on Business Process Management. pp. 159–175. Springer (2009)

9. Brunings, M., Fahland, D., van Dongen, B.: Defining meaningful local process models. In: Proceedings of the International Workshop on Algorithms & Theories for the Analysis of Event Data 2020. CEUR-WS.org (2020), http://ceur-ws.org/Vol-2625/paper-01.pdf
10. Folino, F., Guarascio, M., Pontieri, L.: Mining Multi-variant Process Models from Low-Level Logs. In: Abramowicz, W. (ed.) Business Information Systems. pp. 165–177. Lecture Notes in Business Information Processing, Springer International Publishing, Cham (2015)
11. Günther, C.W., Rozinat, A., Van Der Aalst, W.M.: Activity mining by global trace segmentation. In: International Conference on Business Process Management. pp. 128–139. Springer (2009)
12. Leemans, S.J.J., Fahland, D., van der Aalst, W.M.P.: Process and Deviation Exploration with Inductive Visual Miner. In: Limonad, L., Weber, B. (eds.) Business Process Management Demo Sessions (BPMD 2014). CEUR Workshop Proceedings, vol. 1295, pp. 46–50. CEUR-WS.org (2014), event-place: Eindhoven, The Netherlands
13. Lieben, J., Jouck, T., Depaire, B., Jans, M.: An Improved Way for Measuring Simplicity During Process Discovery. In: Pergl, R., Babkin, E., Lock, R., Malyzhenkov, P., Merunka, V. (eds.) Enterprise and Organizational Modeling and Simulation, vol. 332, pp. 49–62. Springer International Publishing, Cham (2018), series Title: Lecture Notes in Business Information Processing
14. Mannhardt, F., de Leoni, M., Reijers, H.A., van der Aalst, W.M.P., Toussaint, P.J.: From Low-Level Events to Activities - A Pattern-Based Approach. In: La Rosa, M., Loos, P., Pastor, O. (eds.) Business Process Management, vol. 9850, pp. 125–141. Springer International Publishing, Cham (2016), series Title: Lecture Notes in Computer Science
15. Mannhardt, F., de Leoni, M., Reijers, H.A., van der Aalst, W.M., Toussaint, P.J.: Guided process discovery–a pattern-based approach. Information Systems **76**, 1–18 (2018)
16. Mannhardt, F., Tax, N.: Unsupervised Event Abstraction using Pattern Abstraction and Local Process Models. In: Gulden, J., Nurcan, S., Reinhartz-Berger, I., Guédria, W., Bera, P., Guerreiro, S., Fellmann, M., Weidlich, M. (eds.) CEUR Workshop Proceedings. vol. 1859, pp. 55–63. CEUR-WS.org (2017)
17. Mendling, J.: Metrics for process models: empirical foundations of verification, error prediction, and guidelines for correctness, vol. 6. Springer Science & Business Media (2008)
18. Rehse, J.R., Fettke, P.: Clustering Business Process Activities for Identifying Reference Model Components. In: Daniel, F., Sheng, Q.Z., Motahari, H. (eds.) Business Process Management Workshops, vol. 342, pp. 5–17. Springer International Publishing, Cham (2019), series Title: Lecture Notes in Business Information Processing
19. Sánchez-Charles, D., Carmona, J., Muntés-Mulero, V., Solé, M.: Reducing event variability in logs by clustering of word embeddings. In: International Conference on Business Process Management. pp. 191–203. Springer (2017)
20. Senderovich, A., Rogge-Solti, A., Gal, A., Mendling, J., Mandelbaum, A.: The road from sensor data to process instances via interaction mining. In: International Conference on Advanced Information Systems Engineering. pp. 257–273. Springer (2016)
21. Tax, N., Dalmas, B., Sidorova, N., van der Aalst, W.M., Norre, S.: Interest-driven discovery of local process models. Information Systems **77**, 105–117 (Sep 2018)

22. 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.) Proceedings of SAI Intelligent Systems Conference (IntelliSys) 2016, vol. 15, pp. 251–269. Springer International Publishing, Cham (2018), series Title: Lecture Notes in Networks and Systems
23. Tax, N., Sidorova, N., Haakma, R., van der Aalst, W.M.: Mining local process models. Journal of Innovation in Digital Ecosystems **3**(2), 183–196 (Dec 2016)
24. Van Dongen, B.F., de Medeiros, A.K.A., Verbeek, H., Weijters, A., van Der Aalst, W.M.: The prom framework: A new era in process mining tool support. In: International conference on application and theory of petri nets. pp. 444–454. Springer (2005)
25. van Zelst, S.J., Mannhardt, F., de Leoni, M., Koschmider, A.: Event abstraction in process mining: literature review and taxonomy. Granular Computing (May 2020)