Supporting Provenance and Data Awareness in Exploratory Process Mining

Francesca Zerbato[0000−0001−7797−4602]1, Andrea Burattin[0000−0002−0837−0183]2, Hagen Völzer[0000−0003−3547−3847]1, Paul Nelson Becker[2], Elia Boscaini[2], and Barbara Weber[0000−0002−6004−4860]1

[1] University of St. Gallen, St. Gallen, Switzerland
{francesca.zerbato|hagen.voelzer|barbara.weber}@unisg.ch
[2] Technical University of Denmark, Kgs. Lyngby, Denmark
{s194702|s194720}@student.dtu.dk, andbur@dtu.dk

Abstract. Like other analytic fields, process mining is complex and knowledge-intensive and, thus, requires the substantial involvement of human analysts. The analysis process unfolds into many steps, producing multiple results and artifacts that analysts need to validate, reproduce and potentially reuse. We propose a system supporting the validation, reproducibility, and reuse of analysis results via analytic provenance and data awareness. This aims at increasing the transparency and rigor of exploratory process mining analysis as a basis for its stepwise maturation. We outline the purpose of the system, describe the problems it addresses, derive requirements and propose a design satisfying these requirements. We then demonstrate the feasibility of the central aspects of the design.

Keywords: Process Mining · Exploratory Analysis · System Requirements and Design · Analytic Provenance · Data Awareness · User Support

1 Introduction

Process mining comprises methods to analyze event data generated in information systems during the execution of business processes. Process mining is quickly growing in adoption, and so is its business impact [9].

Like other data science disciplines, process mining requires the substantial involvement of humans, e.g., process analysts, to obtain insights from raw event data [7]. Analysts often freely explore the data with the available tools to gain a basic understanding of what it represents, investigate different scenarios, and create hypotheses. Hypotheses can then be tested using best practices, but more exploration is required if the test fails or the results are inconclusive [19]. Each insight that emerges during the analysis informs which subsequent analysis steps are chosen. On the one hand, the choices made during the analysis yield many possible reasonable results that need to be assessed. On the other hand, such choices might give rise to potential inconsistencies in the analysis process [14].

Due to its knowledge-intensive character and emergent course of action, an exploratory analysis includes many manual and error-prone steps that are often

hard to pre-specify and can be challenging to track and validate over long periods of time without tool support [3]. Existing process mining tools do not explicitly support analysts in tracking, validating, and communicating their analysis process and their insights. As a result, analysts must carry out these activities manually, which can quickly become impractical for extensive analyses [6,19].

Toward establishing a more rigorous and reliable analysis process, we suggest increasing the transparency of the analysis process, as a first important step. To this end, in this paper, we propose a system to support process analysts in tracking their analysis steps and the dependencies among them, as well as the results and the goals of their analysis. This aims to support rigor in the analysis process itself as well as communication in reviews, audits, and storytelling activities. The corresponding components of the support system are a *replayable history* of user interactions with a process mining tool and a *provenance view*, which are inspired by reliable system engineering, viz. configuration- and change management [4], and requirements tracing [18]. Moreover, we propose a novel integrated *data view*. The data view aims to support analysts in maintaining awareness of the current data selection, understanding the effect of the data transformations applied to it, comparing the current data selection with previous "states" of the analysis, and increasing the analyst's confidence that each analysis step indeed serves its intended goal or justifies its specified result.

The rest is structured as follows. Sect. 2 motivates our approach with a realistic process mining scenario. Sect. 3 presents the requirements of the system, while Sect. 4 describes its core components. Sect. 5 evaluates the system design. Sect. 6 discusses related work. Sect. 7 closes with an outlook on future research.

2 Motivating Scenario

In this section, we present an example process mining analysis derived from behavioral data we collected in two observational studies with more than 50 experienced process analysts overall [21,22]. We will refer to this scenario throughout the paper to illustrate the problems we address and the proposed solution.

An example process mining analysis. Rob is a process analyst in charge of analyzing an event log recording instances, also known as *cases*, of a road fine management process [5]. Tom, a member of the police acting as a business stakeholder, asks him "What are scenarios in which offenders do not pay their fines?" To answer this question, Rob applies many operations in a process mining tool. Below we describe some steps of his analysis, which we also report in Fig. 1.

(A) Parameter sweeps. First, Rob familiarizes himself with the raw event log (L_0), i.e., the one provided by the police. With the goal in mind to reduce complexity (G1), he removes infrequent behavior with a *variant filter*. He applies the filter three times (cf. o_1-o_3 in Fig. 1) using the relative number of cases as a parameter, i.e., 75%, 85%, and 90%. After applying the filter, he generates descriptives to better understand the effect of the filter on the number of cases

(v_1-v_3). While the first two filter configurations remove too much process behavior, o_3 results in a reasonable number of cases. Thus, he settles for o_3 and annotates his choice, also reporting the reasons for discarding the other filters.

(B) Focus on a subset of the log. Then, Rob focuses on the question (G2), starting from cases corresponding to unpaid fines. He applies an *activity filter* to the previously filtered log to select cases with a credit collection (CC) activity (o_4), which he hypothesizes correspond to unpaid fines (H1). He then creates and inspects the process map of these cases (v_4). From the map, he notices that some cases still include a payment (P) activity; thus, his hypothesis was not precise. He makes a new hypothesis that unpaid fines do not include P (H2) and applies the corresponding filter (o_5) to create a new process map (v_5).

(C) Test hypotheses and compare results. Afterward, Rob focuses on partially paid fines. He hypothesizes that these are complete cases that include either P and some outstanding amount or both activities P and CC (H3). To check his hypothesis, he removes incomplete cases (o_6), selects cases that include P (o_7), and adds a derived attribute to filter for cases with a positive outstanding amount (o_8-o_9). He also mistakenly re-applies o_7. After inspecting the results, he continues with the second part of H3. He applies an activity filter that works on the conjunction of both conditions (o_{10}). However, he is unsure about the logic implemented by the filter in o_{10} because he finds it difficult to understand the

	U	Id	Operation	I/O		Timestamp	User Annotations	Goals and Hypotheses
(A)	R	o_1	variantFilter(cases, keep, 75%)	L_0	L_1	07/10/22 10:01:18	filtered too much	
	R	v_1	nCases()	L_1	#cases	07/10/22 10:01:50		
	R	o_2	variantFilter(cases, keep, 85%)	L_0	L_2	07/10/22 10:02:03	filtered too much	**G1**: Reduce complexity
	R	v_2	nCases()	L_2	#cases	07/10/22 10:02:32		
	R	o_3	variantFilter(cases, keep, 90%)	L_0	L_3	07/10/22 10:03:11	good trade-off	
	R	v_3	nCases()	L_3	#cases	07/10/22 10:03:29		
(B)	R	o_4	activityFilter(cases, keep, "CC")	L_3	L_4	07/10/22 10:15:02		**G2**: Answer Question
		v_4	processMap()	L_4	M_1	07/10/22 10:15:57	there is still P	**H1**: Unpaid fines include CC
	R	o_5	activityFilter(cases, remove, "P")	L_4	L_5	07/10/22 10:18:32	unpaid fines	**H2**: Unpaid fines do not include P
	R	v_5	processMap()	L_5	M_2	07/10/22 10:18:56		
(C)	R	o_6	filterIncompleteCases(cases, remove)	L_0	L_6	07/10/22 10:25:10		**H3**: Partially paid fines are fines that include P and some outstanding amount or include both P and CC
	R	o_7	activityFilter(cases, keep, "P")	L_6	L_7	07/10/22 10:26:45	paid fines with an outstanding amount	
	R	o_8	addDerivedAttr(amountDue)	L_7	L_8	07/10/22 10:29:02		
	R	o_9	attrFilter(cases, keep, amountDue>0)	L_8	L_9	07/10/22 10:29:49		
	R	o_7	activityFilter(cases, keep, "P")	L_9	L_{10}	07/10/22 10:29:49		
	R	o_{10}	activityFilter(cases, keep, "P ∧ CC")	L_6	L_{11}	07/10/22 10:31:36	fines with P+CC	
	R	o_7	activityFilter(cases, keep, "P")	L_6	L_{12}	07/10/22 10:33:18		**G3**: Validate combined filter
	R	o_4	activityFilter(cases, keep, "CC")	L_{12}	L_{13}	07/10/22 10:33:44	filter is correct	
	R	o_{11}	activityFilter(cases, remove, CC)	L_{12}	L_{14}	07/10/22 10:36:51		**H4**: Some partially paid cases do not include CC
(D)	R	Show results to business stakeholders and auditors						**G4**: Storytelling
(E)	J	o_5	activityFilter(cases, remove, "P")	L_3	L_{15}	14/10/22 08:33:17	order of filters checked	**G5**: Internal auditing
		o_4	activityFilter(cases, keep, "CC")	L_{15}	L_{16}	14/10/22 08:33:46		

Fig. 1. The operations applied by Rob and Julie over time. **U** is the user, i.e., (R)ob or (J)ulie. **Id** is an identifier for each operation of the **Operation** column. **I/O** shows the input resp. output of an operation. **Timestamp** is the timestamp of the operation. The last two columns show **User Annotations** and analysis **(G)oals and (H)ypotheses**.

effect of nested operations, and he knows that, in some cases, filters are sensitive to their order. Thus, Rob decides to validate the filter (G3) by applying two separate filters and checking each one of the conditions in o_{10} individually. He documents this check by taking notes. As a result of his analysis, he rejects H3 because the two results don't match as he had expected. Thus, he filters for partially paid cases that do not include CC (o_{11}) as he hypothesizes that the credit collection agency does not handle some partially paid fines (H4).

(D) Storytelling. After the analysis, Rob presents his results to Tom (G4). For each result, Rob points to the supporting evidence from his analysis. In a deeper discussion, he explains to Tom how he obtained the results, including steps that did not directly contribute to the final outcomes but accounted for analysis decisions. For example, when Tom asks why he focused on 90% of the raw event log, Rob shows all the parameter values he had tested in (A) and uses his annotations to share the reasoning behind the decisions he made.

(E) Internal auditing. A week later, Julie looks at Rob's analysis with the goal of auditing it (G5). She inspects Rob's results and runs some validation steps using a different process mining tool. For example, she swaps the filters that Rob applied in (B) to see if their order has any effect on the final result. Indeed, she knows that the semantics of filter operations is often implicit or not communicated unambiguously and that different process mining tools might implement different filter semantics. She also notices that one filter (o_7) does not have any effect on the result and, as such, could be removed in a future step.

Scenario conclusion. Rob's analysis (A)-(C) reflects typical process mining analysis steps, where multiple operations are combined to analyze different subsets of the event log. Such steps reflect the knowledge-intensive and ad-hoc character of exploratory process mining analysis, which develops based on emerging insights, as shown by the observational study in [21]. While process mining tools allow realizing such steps, they lack support for analysts to track their analysis steps and goals (cf. (A)-(C)), e.g., to maintain a resource of "reflection in action" [20], and to conduct other meta-analysis activities, such as storytelling, validation and auditing (cf. (D)-(E)). However, such activities are crucial to increase the rigor of the analysis and make it more reliable and less prone to errors that can, for example, derive from combining and nesting operations with different or ambiguous semantics. This is the main driver of our paper.

3 Requirements

To support reviews, audits, and other validation activities for process mining analysis, we derived the following non-exhaustive set of requirements for a system supporting the *process of process mining*. Such a support system is thought of as complementing an existing process mining tool, not of replacing it.

(R1) Maintain Provenance Information. The support system should enable analysts to capture and browse provenance information about their analysis process and its results.

Provenance information includes the analysis operations performed, their input and output data, as well as dependency information about which results depend on which operations. With the help of provenance information, analysts can reproduce their analysis for validation, storytelling, and auditing.

(R2) Trace Analysis Goals and Insights. The support system should enable analysts to trace which analysis operations were performed to achieve which analysis goals and which insights were obtained from which operations.

The tracing of goals and insights clarifies to the analyst and the stakeholders why a certain analysis operation or set thereof has been performed, supporting validation, storytelling, and auditing. It also supports the identification of ineffective explorations and helps identify candidates for reuse.

For the next requirement, we note that the user interface of many process mining tools is designed in such a way that multiple UI elements, e.g., the process map or the variant inspector, operate on a specific selection of the event log that is defined through active filters and transformations. Whenever the active selection changes, all UI elements change accordingly. In these rapidly-changing settings, it is often challenging to keep track of which data selection was used to generate any (intermediate) result or artifact produced during the analysis.

(R3) Increase Data Awareness. The support system should help analysts to be aware of the data selection and properties of the data that the process mining tool interfaces are operating on. This should apply not only to the current step of the analysis but also to all previous steps, favoring the comparison among the results produced throughout the analysis from different data selections.

Data awareness, i.e., the awareness of the data selection on which the current analysis step is operating, supports the assurance of the effectiveness of the data selection, as well as the understanding of the effects of the operations performed in each step and the comparison among different data selections over time.

4 A System to Support the Analysis Process

In this section, we describe the design, intended use, and underlying assumptions of the support system. The system consists of three main components: (i) a *replayable history* comprising the sequence of all the performed operations, as well as two complementary kinds of linked views: (ii) a *provenance view*, which captures provenance information about the analysis process, and (iii) multiple *data views*, which capture how the working event log is transformed during the analysis at different levels of abstraction.

4.1 Replayable History of Interactions

The proposed support system integrates with an existing process mining tool through a *replayable history* of interactions. That is an append-only record of all interactions of the process analyst with the tool. Columns 3-6 of Fig. 1 represent such a replayable history, i.e., we use the operations, their input and output

and their ordering given by the timestamp. It must be *replayable*, meaning that it contains enough information to reproduce the *state* of the process mining analysis. Based on the design of most process mining tools, we assume that the core part of that state is a *working event log*, i.e., the initially loaded event log, that was transformed through various process mining operations, e.g., filtering, aggregation, and data enrichment. Some operations, which we refer to as *view operations* (cf. Sect. 2), will also produce additional artifacts, e.g., a process map, and consume additional artifacts, e.g., a normative process model, but our main focus within this paper is on the working event log. We discuss the extension of our approach to more complex process mining analysis settings in Sect. 7.

Let L_0 denote the event log initially loaded in the process mining tool in a given session. A *replayable history* is a sequence $op_1, op_2, \ldots$ of *operations* on the working event log, where an *operation* $op_j = (i, o, L_j)$ consists of

(i) an *input log index* $i \geq 0$ such that $i < j$; We say that L_i is the *input log* to operation op_j,
(ii) a parameterized *operation signature* $o = (f, p_1, \ldots, p_k)$ where f is a process mining tool function that accepts an event log L and actual parameters $p_1, \ldots, p_k$, and
(iii) an event log L_j, which we call the *output log of* op_j such that $L_j = o(L_i) =_{\text{def}} f(L_i, p_1, \ldots, p_k)$, i.e., o describes a deterministic[1] process mining tool function f such that the output only depends on parameters $p_1, \ldots, p_k$.

For example, o_4 in Fig. 1 refers to the operation

$$L_4 = \text{activityFilter}(L_3, \text{'cases'}, \text{'keep'}, \text{'CC'})$$

that keeps only cases with the 'CC' activity. We henceforth do not strictly distinguish between an operation and its signature when the difference is clear from the context. Also, we note that the input log to the first operation op_1 of the replayable history must be L_0, which we assume is implicitly part of the history.

In general, the input event log can be any event log produced previously and not just the direct predecessor in history. This means that analysts can navigate back to a previous working event log and continue the analysis from there. This type of navigation might not be a native capability in the process mining tool but can be provided by the provenance view presented in Sect. 4.2.

We consider operations that are "relevant" for process analysts [2], such as data exploration interactions [14], i.e., we choose a level of abstraction comparable to the one in process mining tools. In this way, we can also include view operations, such as v_4 in Fig. 1, which refers to the operation $M_1 = \text{processMap}(L_4)$.

The determinacy in (iii) above guarantees that any intermediate version of the working log can be reproduced recursively. When a replayable history is properly integrated with a process mining tool, then all the analysis artifacts and UI elements that are based on earlier versions of the working log can be

[1] Note that determinacy of an operation that calls a pseudo-random function requires the inclusion of the random seed in the operation parameters.

reproduced to support auditing, storytelling, and other communication use cases. Moreover, the replayable history is the basis for constructing the provenance and data views introduced in the following paragraphs.

4.2 Provenance View

The *provenance view* is a rooted, directed tree that reflects the analysis steps performed on the working event log as a branching history [6].

The tree of the provenance view is a visualization of the replayable history where each index $j = 0, 1, \ldots$ of the history is a node labeled with the log L_j and where each operation $op_j = (i, o, L_j)$ creates an edge from node i to node j labeled with o. Fig. 2 shows a provenance view derived from the replayable history of Fig. 1, where each node is labeled with an object representing the state current state of the analysis, i.e., the working event log, and each edge is labeled with an operation. For example, node L_3 is the node representing the event log given as input to operation o_4, an activity filter that retains cases with activity CC; L_4 is the filtered event log resulting from o_4.

A path between two nodes represents a sequence of operations that transforms the first node, i.e., the working event log, into the second one. New paths are created via *branching*. We consider tree branches to reflect distinct analysis (sub-)goals, such as testing a specific hypothesis or validating a particular operation, that develop on the same working log. For example, G1 in Fig. 2 recalls the goal of the motivating scenario to reduce event log complexity via variant filters. Analysts can interact with the provenance tree to deliberately create new branches and decompose the analysis into different goals or reuse the results of previously performed steps for further analysis. For example, new branches can be created to test hypotheses or to validate the effects of different operations on the same working log. Also, branching helps visualize the "analysis coverage", i.e., all the steps done in the course of an analysis, highlighting missing steps and discarded results. Discarded results can provide precious information about the process of obtaining the final result. For example, the branching realized by o_1-o_3

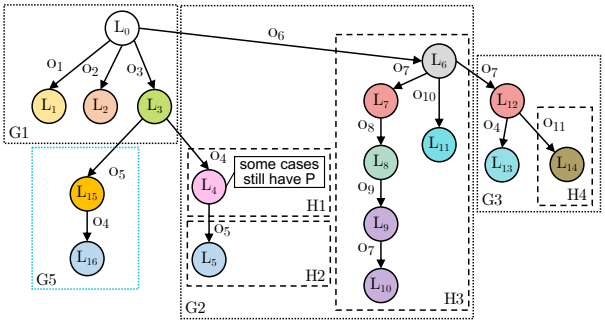

Fig. 2. Provenance view derived from the motivating example of Fig. 1. User annotations, hypotheses (H), and goals (G) are sketched on the tree to exemplify their use.

in Fig. 2 shows that different parameter configurations were tested, indicating that the choice of applying o_3 was informed by the (unsatisfactory) results of o_1 and o_2. Instead, missing steps can indicate unexplored areas needing attention.

One last important feature of the provenance view is annotations. Annotations can be attached to nodes or sub-trees to capture user observations about specific analysis outcomes as well as analysis goals. Annotations are versioned to allow tracking changes in the flow of thoughts as the analysis evolves. Analysts can use annotations to "give meaning" to intermediate results, to document analysis steps and decisions, or to report hypotheses and goals. For example, node L_4 in Fig. 1 is annotated with "some cases still have P", documenting the fact that the filter o_4 did not result in unpaid fines, i.e., cases without a payment (P) activity, as expected (cf. Sect. 2). Similar to literate programming [10], the information included in annotations can support analysts in storytelling tasks.

Overall, the provenance view provides access to analytic provenance information organized into analysis goals via branching and annotation. The provenance view also allows the analyst to reuse previously obtained results, e.g., to extend them or use them as input for further analysis steps, as done with L_6 in Fig. 2. Finally, it provides a transparent view of the analysis process that can help identify unnecessary and missing steps, inform future analyses, and serve as a basis to automate steps that might emerge as repetitive.

4.3 Data Views

Complementing the provenance view, the analysis is also reflected in one or more data views. Fig. 3(a) shows a *data view*, which is a rooted, directed multi-graph. Each node is labeled with an object that represents some aspect of the state, i.e., the working event log, and hence some aspect of the effect of the previous analysis step. Fig. 3(a) shows a data view, which we call the *complete data view*, where each node is labeled with the complete working event log, not just one aspect of it. However, in contrast to the provenance view, the different sequences of operations represented in the edges can lead to the same node whenever they produce the same working event log. For example, starting from node L_3, both sequences o_4, o_5 and o_5, o_4 commute, i.e., they lead to the same node ($L_{5/16}$).

Different sequences of operations resulting in the same log can be expected or unexpected for the analyst. The previous example of commuting filters (o_4, o_5) might be expected, whereas the observation that starting from L_8 the sequences o_8 and o_8, o_9 result in the same log might be unexpected. Indeed, the fact that o_9 had no effect might depend on specific characteristics of the working event log. This is one way in which the data view can help analysts validate the effect of their actions or spot inconsistencies stemming from unclear operation semantics.

Conversely, analysis steps with different working event logs are represented by different nodes. Again, working logs being different can be expected or unexpected by the analyst, the latter, e.g., for non-commuting permutations of the same filters. Different nodes can then be compared with a dedicated *diff* capability within the data view, which allows the analyst to investigate an unexpected difference. A full design of such a diff capability is out of the scope of this paper.

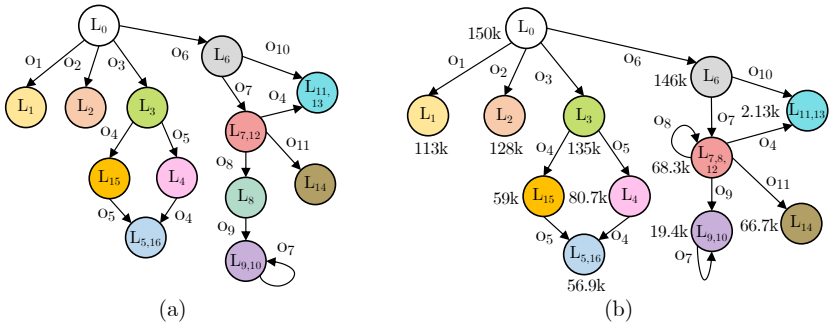

Fig. 3. Examples of data views based on the motivating example. (a) *Complete* data view; (b) data view using the *number of cases* as abstraction.

We assume that the complete data view is always automatically provided to the user. The complete data view can be computed from the provenance view by unifying nodes that are labeled with equal event logs, as shown in Fig. 2 with the same color, or directly from the replayable history as described below.

Fig. 3(b) shows another data view for our example. In general, each data view is created from a pre-defined or user-defined *abstraction*. An *abstraction* α defines, for each working event log L, an object $\alpha(L)$. The set of cases, the number of cases, or the number of case variants are frequently used abstractions in process mining, but abstractions based on data attributes or other event log characteristics may also be defined. *Case abstractions* are of special interest, which are defined through a mapping β that defines an object $\beta(c)$ for each case c and $\alpha(L)$ is then the set or bag of all $\beta(c)$ where c is a case of L. For example, we can disregard the ordering of events in a case c and define $\beta(c)$ as the set of activities that occur in c. A user-defined abstraction may also disregard event data features that are deemed irrelevant for a specific analysis. Fig. 3(b) uses the number of cases as data view abstraction. Whenever an abstraction is just a short object, e.g., a number, we can show it directly in the data view.

A data view for an abstraction α is defined as follows for the general case. An abstraction α induces an *equivalence* over working event logs L_1, L_2 by $L_1 \equiv L_2$ whenever $\alpha(L_1) = \alpha(L_2)$. The *data view* generated by α is a directed graph where the nodes are the equivalence classes of this equivalence: Let $[L] = \{L' \mid L' \equiv L\}$ denote the *equivalence class* of working event logs w.r.t. to L. Then, the nodes of the data view are the classes $[L]$ where L is some working event log of the replayable history and there is an edge from $[L]$ to $[L']$ labeled with operation o whenever there exists an $L_1 \in [L]$ and an $L_2 \in [L']$ such that L_2 was obtained from L_1 in the replayable history by applying the operation o. Again, a diff capability on the abstraction can support the analyst in investigating differences. Consider, for example, two nodes connected by an edge that represents a case filter. In that case, the diff consists of two sets: the set of cases removed by the filter and the set of cases kept by the filter. If the diff view allows the analyst to search for specific test cases from the log, then the analyst can determine in

which of the two sets of the diff the test case ended up. In this sense, the diff view allows the analyst to test whether the case filter had the desired effect.

Different data views allow analysts to explore the effect of their operations at different abstraction levels. Also, they provide orthogonal and overlapping perspectives into the current data selection. Hence the data views can raise the awareness of the analyst for potential mistakes and ease the comparison among different intermediate results. While the provenance view traces the analysis steps to their goals, the data views can help validate that the results were obtained on the proper data selection. Many useful abstractions and their semantic equivalences, such as the examples above, are generic, i.e., they can be provided off-the-shelf to the user and can be reused across projects. Project-specific abstractions that are deemed useful, such as abstractions that use specific definitions of process outcomes or performance metrics, could be reused in similar projects, e.g., when different projects deal with the same process type.

Requirements review. The requirements presented in Sect. 3 are addressed by the design of the support system as follows.

(R1), i.e., the maintenance of the provenance information is addressed mostly through the replayable history, which guarantees that analysis steps are recorded and intermediate results can be reproduced. A specific part of the recording is the ability of analysts to navigate back in history and create a new branch of the provenance tree. This feature is supported by the provenance view.

(R2), i.e., the tracing of analysis goals and insights, is realized by the provenance view and its annotation capability. The annotated provenance tree shows which analysis branches serve which goals and which insights are derived from them. These links can be inspected for consistency or for assessing the analysis coverage, thus supporting the reasoning over analysis steps and results.

(R3), i.e., the support for data awareness is implemented predominantly by the data views, which provide multiple cues for the analysts on what data selection they are currently operating on. The equivalence between working event logs and the capability to compare them at different abstraction levels allow analysts to understand and validate various aspects of their analysis.

5 Evaluation

In this section, we evaluate our approach by demonstrating the feasibility of central aspects of the design. Sect. 5.1 presents tests that evaluate the efficiency of updating the data view based on different equivalences. Sect. 5.2 presents a proof-of-concept and describes how it informed the next evaluation stages.

5.1 On the Efficiency of Computing the Equivalence in Data Views

To demonstrate the feasibility of our design, we focused on the efficiency of computing event log equivalences in the data views. Indeed, establishing whether two logs are equivalent in a data view is computationally not trivial. Whenever

a new analysis operation is performed, it is necessary to decide whether the graph of the data view needs to be updated with a new node, i.e., the working log is different from all the previously computed ones, or with a self-loop, i.e., the working log is *equivalent* to a previously computed one (cf. Sect. 4.3). Since different equivalences could incur different computational costs, which might result in delays if many large logs are compared, we devised some tests to investigate the performance of updating different data views.

We implemented two equivalences for XES-formatted event logs. The first, *size-only*, considers two logs as equivalent if they have the same number of events. The second, named *complete*, considers two logs as equivalent if they have the same set of cases and each case has the same set of events. Two events are the same if they have the same key/value attributes. We implemented this equivalence considering realistic improvements, i.e., we check increasingly restrictive conditions, and as soon as one condition fails, two logs are deemed not equivalent.

For running the tests, we constructed logs of exponentially increasing size, i.e., different *configurations*. Specifically, we generated logs with 100/1 000/10 000 cases and 10/50/250 events per case, thus ending up with 9 configurations (the smallest event log contains 100 cases with 10 events each = 1 000 total events; and the largest event log contains 10 000 cases with 250 events each = 2 500 000 total events). Each event has 10 attributes. Details are provided in Table 1.

For our tests, we identified five *scenarios* (s1-s5) that represent edge cases from a computational point of view. These scenarios capture differences between event logs that become increasingly subtle and, as such, increasingly harder to compute. In each scenario, we compare a "given" event log with an amended copy, which incorporates the following changes:

s1. One case of the given log is filtered out (for our test, we filtered out the very last case that appears in the given log);
s2. One event of the given log is filtered out (we filtered out the very last event of the last case that appears in the given log);
s3. One attribute of the very last case of the given log is changed;
s4. One attribute of the last event of the last case of the given log is changed;
s5. No changes; the copy is identical to the given event log.

As a result, we obtained 45 event logs (9 configurations × 5 scenarios), where the perturbations represent the worst-case possibility, i.e., finding the difference requires inspecting the entire logs.

We computed the *size-only* and *complete* equivalences while monitoring the required time. For the tests, we used a laptop running Windows 10 64bit and equipped with an Intel Core i7-7500U 2.70GHz CPU and 16GB of RAM. From

	c1	**c2**	**c3**	**c4**	**c5**	**c6**	**c7**	**c8**	**c9**
Number of cases	100	1 000	10 000	100	1 000	10 000	100	1 000	10 000
Events per case	10	10	10	50	50	50	250	250	250
Events in the log	1 000	10 000	100 000	5 000	50 000	500 000	25 000	250 000	2 500 000

Table 1. Cases and events for each configuration used for testing the equivalences.

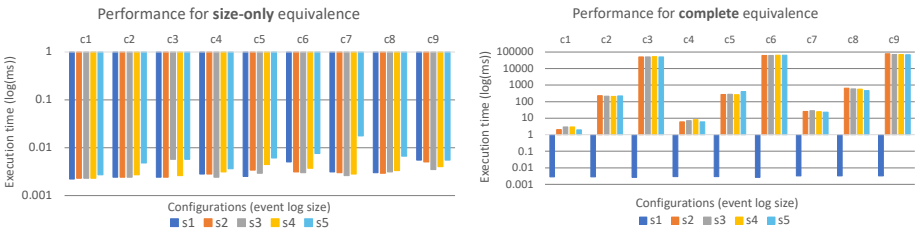

Fig. 4. Performance of the *size-only* and *complete* equivalences against the 9 configurations and 5 scenarios. The y-axes are in a logarithmic scale of milliseconds, i.e., bars below 1 mean that it took less than 1 ms to compute the equivalence.

the performance results in Fig. 4, we can see that the time required for processing the *size-only* equivalence is negligible (less than milliseconds in all cases) as it requires verifying only the size of the log. Instead, the time needed to compute the *complete* equivalence grows linearly with the number of events in the log (the worst time reported is 77 seconds for configuration c9). This holds for all scenarios but S1, where differences are instantly detected. The linear complexity for the *complete* equivalence suggests that the computation of the equivalence between two event logs is scalable. Also, the worst-case scenario is observed only when two logs have no differences, meaning that in other cases the equivalence computation can be more efficient. For example, for the data views of Fig. 3, which both have multiple different nodes, we could imagine a strategy that first computes the *size-only* equivalence and, only if two logs are equivalent, it computes the *complete* equivalence. Besides, since any new node will be equivalent to at most one other node, the worst-case scenario will be observed at most once.

From these tests, we conclude that the equivalences implemented are already promising from a performance viewpoint, despite the fact that no particular optimization was considered. The implementation of the equivalences, the scripts used to generate the logs, the logs, and the detailed results are publicly available.[2]

5.2 Proof-of-Concept Implementation

To better understand how the support system proposed in this paper could be used, we have implemented a conceptual prototype. The implementation, which is available as open source[2], consists of a web application with a backend realized in Python and a frontend written in Javascript.

The prototype implements the core functionality of the three components described in Sect. 4. The replayable history is created directly in the prototype from recorded user interactions: the user can upload an event log and apply different filter operations, e.g., activity, directly-follows, or throughput-based filters. In this way, the prototype can "simulate" the analysis done in a process mining tool and reproduce a basic yet realistic history of interactions. Based

[2]See https://doi.org/10.5281/zenodo.7329844.

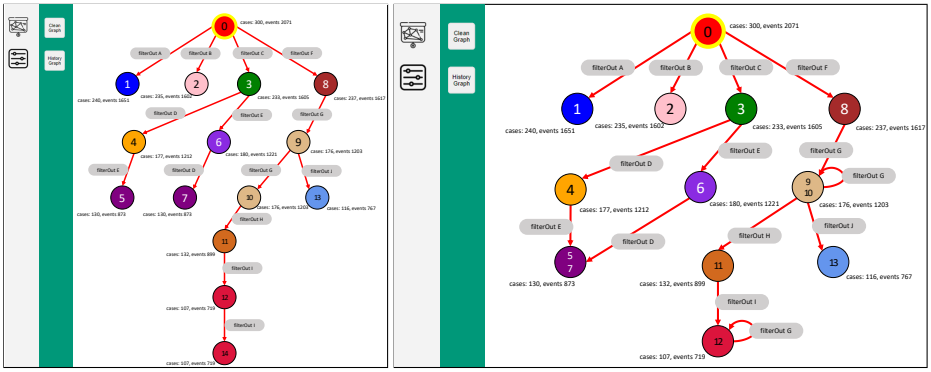

Fig. 5. Screenshots of the developed prototype showing the provenance view (left) and the corresponding data view (right). The yellow ring around a node indicates the "current selection", i.e., the working event log to which operations are being applied.

on the replayable history, the prototype can create a provenance view and a corresponding data view and update them whenever the replayable history is extended. The user can navigate the provenance tree, create new branches from any selected node, i.e., to pursue different analysis goals, and annotate the nodes. In turn, in the data view, the user can visualize the working event logs, which are deemed equivalent if they have the same set of case ids. Fig. 5 shows screenshots of the prototype with a scenario in which two pairs of equivalent nodes occur.

As part of our evaluation, we used the prototype to replay scenarios from [22]. Our ultimate goal was to identify critical aspects for the implementation of the system that should be addressed before conducting a user evaluation. In particular, we identified the following critical points for the next evaluation stage.

- Tighter integration of the support system with a process mining tool should be considered to enable the automated export of replayable histories. Recreating a replayable history from a realistic analysis in a fully-fledged process mining tool is error-prone and expensive.
- Although our tests have shown that equivalences in the data view can be efficiently computed, we have learned that the complexity of both the provenance tree and the data view graph can become large with long repayable histories. The support system should provide means to manage such complexity, e.g., by easing frequent branching and the navigation of large trees.
- The branching in the provenance view is done by users based on different reasons, e.g., wanting to reuse an intermediate result or validate a sequence of steps. Further investigation on the reasons for branching should be conducted to enable the system to differentiate them and instruct users accordingly.

6 Related Work

In this section, we discuss related work comparing our system with relevant literature and existing software in the fields of data analytics and process mining.

The idea to increase analytical rigor is inspired by techniques from systems engineering, such as configuration- and change management [4] and requirements tracing [18]. Although such systems are not conceived to support exploratory data analysis, they inspired the design of the provenance view by manifesting the evolution process and the user goals and enabling deliberate branching.

More closely related to our work are systems that capture and visualize *analytic provenance* to assist data analysis and sense-making processes [20]. Among them, *provenance management systems* [13] have emerged in the areas of scientific workflows and visual analytics. Such systems leverage different kinds of provenance information [14] to assist analysts in the creation and management of data analysis workflows. Notable examples are VisTrails [3] for visualization workflows, Chimera [8] for data derivations, and ZOOM [2] for bioinformatics. Provenance management systems capture provenance information in the form of a history or action log, similar to our replayable history, and often allow users to visualize it, similar to our provenance tree. For example, VisTrails [3] allows users to manage visualization workflows and to maintain and navigate the history of their design. However, these systems focus on repeatable analyses, requiring analysts to pre-specify workflows. As such, they do not directly support exploration processes learned from "free" interactions between the analyst and the tool. Moreover, only a small part of such systems use provenance information to support sense-making in terms of improving the analysis rigor [20], which is what we aim to support with the tracing of user goals in the provenance view and the user-defined equivalences in the data view. An example of tools supporting sense-making is InfoVis [15], a framework for maintaining the provenance of visualization states that allows analysts to externalize their reasoning process. Compared to InfoVis, we propose multi-perspective data views as a novel feature to increase data awareness in contexts like process mining, where the data selection changes frequently and, thus, might make it hard to keep track of how the results were generated from the data.

In process mining, the capture and management of provenance information have received little attention so far. Inspired by scientific workflows, some works have focused on supporting the management of changes in *process mining pipelines*. Examples of tools that offer this kind of support are RapidProM [1], buparFlow [16], and PM4KNIME [11] for process mining pipelines or FilterTree [12] for chaining filters into preprocessing pipelines. Although these tools might resemble our work, they support repeated analyses through reusable pipelines but are not designed to assist exploratory analysis. Also, these tools do not manage provenance information, nor do they explicitly support sense-making or meta-analysis tasks, e.g., through the tracing of analysis goals, or data awareness, e.g., with user-defined abstractions on the working event log. Similar remarks can be made for commercial process mining tools [17], which, to our knowledge, do not yet support the recording and management of analytic provenance information. Existing features like the Process Diff of IBM Process Mining could inform the design of the diff capability in data views defined over (discovered) process models as analysis artifacts (cf. view operations in Sect. 4.1). However,

the comparison among process mining analysis states, such as working logs, is not supported by existing software.

7 Conclusion and Outlook

In this paper, we have proposed a support system aimed at providing process analysts with a transparent overview of their analysis and making them aware of the data selection they work with at each analysis step. The proposed system can support the provenance and reproducibility of the analysis, ease result validation and auditing, and provide a basis to improve the rigor of the analysis process.

Limitations and Outlook. The first limitation of this paper concerns the notion of state of a process mining analysis, which we have assumed being a single event log. As mentioned in Sect. 4.1, a reasonable extension of the system could manage additional analysis inputs, e.g., a normative process model. This extension is straightforward as long as the interaction between different input artifacts is limited. Each artifact can be managed separately with dedicated provenance and data views. However, with the emergence of multi-process mining[3], multiple event logs may have stronger interactions, i.e., they might change simultaneously in interdependent ways. Such cases might require a generalization of the current definition of provenance view, which we will explore in the future. Another limitation concerns the current evaluation of the support system. On the one hand, the experimental evaluation in Sect. 5.1 is limited to simple equivalences. Future work should investigate the feasibility of other equivalences that might be relevant for process mining practice. On the other hand, the effectiveness and usefulness of the proposed system have not been evaluated by users. Towards a user evaluation, we have used a proof-of-concept implementation of our system to replay realistic scenarios and identify critical aspects for future work that should be addressed before designing such a study. For example, we have learned that a new implementation of the system should consider methods to deal with the complexity of large provenance and data views, such as techniques for tagging and storing milestones or optimizing navigation via scrolling and panning.

Acknowledgment. This work is part of the ProMiSE project, funded by the Swiss National Science Foundation under Grant No.: 200021_197032.

References

1. Van der Aalst, W., Iriondo, A.B., Van Zelst, S.: RapidProM: Mine your processes and not just your data. In: RapidMiner: Data Mining Use Cases and Business Analytics Applications. Chapman & Hall/CRC Press (2018)
2. Biton, O., Cohen-Boulakia, S., Davidson, S.B., Hara, C.S.: Querying and managing provenance through user views in scientific workflows. In: 2008 IEEE Int. Conf. on Data Engineering (ICDE). pp. 1072–1081 (2008)

[3] https://multiprocessmining.org

3. Callahan, S.P., Freire, J., Santos, E., Scheidegger, C.E., Silva, C.T., Vo, H.T.: Vistrails: visualization meets data management. In: Proc. of the 2006 ACM SIGMOD Int. Conf. on Management of Data. pp. 745–747 (2006)
4. Conradi, R., Westfechtel, B.: Version models for software configuration management. ACM Comput. Surv. **30**(2), 232–282 (1998)
5. De Leoni, M., Mannhardt, F.: Road traffic fine management process. Eindhoven University of Technology, Dataset (2015)
6. Derthick, M., Roth, S.: Enhancing data exploration with a branching history of user operations. Knowl Based Syst **14**(1), 65–74 (2001)
7. Doan, A.: Human-in-the-loop data analysis: A personal perspective. In: Proc. of the Workshop on Human-In-the-Loop Data Analytics. HILDA'18, ACM, New York, NY, USA (2018). https://doi.org/10.1145/3209900.3209913
8. Foster, I., Vockler, J., Wilde, M., Zhao, Y.: Chimera: a virtual data system for representing, querying, and automating data derivation. In: Proc. of the 14th Int. Conf. on Scientific and Statistical Database Management. pp. 37–46 (2002)
9. Grisold, T., Mendling, J., Otto, M., vom Brocke, J.: Adoption, use and management of process mining in practice. Bus. Process Manag. J. (2020)
10. Knuth, D.E.: Literate Programming. The Computer Journal **27**(2), 97–111 (01 1984). https://doi.org/10.1093/comjnl/27.2.97
11. Kourani, H., van Zelst, S.J., Lehmann, B.D., Einsdorf, G., Helfrich, S., Liße, F.: PM4KNIME: Process Mining Meets the KNIME Analytics Platform. In: ICPM Demo Track. pp. 65–69. CEUR Workshop Proceedings (2022)
12. Leemans, S.: Filtertree: a repeatable branching XES editor. In: ICPM Doctoral Consortium and Demo Track. pp. 70–74. CEUR Workshop Proceedings (2022)
13. Pérez, B., Rubio, J., Sáenz-Adán, C.: A systematic review of provenance systems. Knowl Inf Syst **57**(3), 495–543 (2018). https://doi.org/10.1007/s10115-018-1164-3
14. Ragan, E.D., Endert, A., Sanyal, J., Chen, J.: Characterizing provenance in visualization and data analysis: an organizational framework of provenance types and purposes. IEEE Trans Vis Comput Graph **22**(1), 31–40 (2015)
15. Shrinivasan, Y.B., van Wijk, J.J.: Supporting the analytical reasoning process in information visualization. In: Proc. of the SIGCHI Conference on Human Factors in Computing Systems. p. 1237–1246. CHI '08, ACM, New York, NY, USA (2008)
16. Steukers, B., Janssenswillen, G., van Hulzen, G.A.W.M., Vanhoenshoven, F., Depaire, B.: bupaRflow: A Workflow Interface for bupaR. In: BPM Demo and Resources track. vol. 3216, pp. 102–106. CEUR Workshop Proceedings (2022)
17. Viner, D., Stierle, M., Matzner, M.: A process mining software comparison. arXiv preprint arXiv:2007.14038 (2020)
18. Watkins, R., Neal, M.: Why and how of requirements tracing. IEEE Software **11**(4), 104–106 (1994). https://doi.org/10.1109/52.300100
19. Wongsuphasawat, K., Liu, Y., Heer, J.: Goals, process, and challenges of exploratory data analysis: An interview study. arXiv:1911.00568 (2019)
20. Xu, K., Attfield, S., Jankun-Kelly, T., Wheat, A., Nguyen, P.H., Selvaraj, N.: Analytic provenance for sensemaking: A research agenda. IEEE Comput Graph Appl **35**(3), 56–64 (2015). https://doi.org/10.1109/MCG.2015.50
21. Zerbato, F., Soffer, P., Weber, B.: Initial insights into exploratory process mining practices. In: Business Process Management Forum. LNBIP, vol. 427, pp. 145–161. Springer (2021). https://doi.org/10.1007/978-3-030-85440-9_9
22. Zerbato, F., Soffer, P., Weber, B.: Process mining practices: Evidence from interviews. In: Int. Conf. on Business Process Management (BPM). pp. 268–285. LNCS, Springer (2022). https://doi.org/10.1007/978-3-031-16103-2_19