International Journal on Software Tools for Technology Transfer (2022) 24:549–561
https://doi.org/10.1007/s10009-021-00618-y

COMPETITIONS AND CHALLENGES

Regular

The Log Skeleton Visualizer in ProM 6.9

The winning contribution to the process discovery contest 2019

H. M. W. Verbeek[1]

Accepted: 6 May 2021 / Published online: 17 May 2021

Abstract
Process discovery is an important area in the field of process mining. To help advance this area, a process discovery contest (PDC) has been set up, which allows us to compare different approaches. At the moment of writing, there have been three instances of the PDC: in 2016, in 2017, and in 2019. This paper introduces the winning contribution to the PDC 2019, called the Log Skeleton Visualizer. This visualizer uses a novel type of process models called log skeletons. In contrast with many workflow net-based discovery techniques, these log skeletons do not rely on the directly follows relation. As a result, log skeletons offer circumstantial information on the event log at hand rather than only sequential information. Using this visualizer, we were able to classify 898 out of 900 traces correctly for the PDC 2019 and to win this contest.

Keywords Log skeletons · Process discovery · Event logs · Process discovery contest

1 Introduction

In the field of *process mining* [2], *event logs* (*logs* for short) play a key role. In a nutshell, a log corresponds to a collection of *traces*, where every trace corresponds to a single execution of the process at hand that is captured by a sequence of *events*. These events then may contain attributes like the name of the activity involved, the resource (like an employee) that initiated the event, or the time at which the event occurred. In the area of *process discovery*, process models are discovered from these logs. Such a process model describes the activities and the interplay between these activities in some way. Ideally, of course, the discovered process model describes the process at hand (that generated the log) perfectly. In the area of *process conformance*, the extent to which a process model agrees with the traces as captured in a log is measured. For a process model to completely agree on a log, every trace from that log can be replayed in the process model successfully. In the area of *process extension*, the process models are enriched with additional perspectives based on the data as captured by logs. For example, if the log contains timing information for every event, then (aggregated) timing information can be added to the process model, which could show clear bottlenecks (like activities in the process that take very long to be completed).

To help advance and stimulate the process discovery area, in the year 2016, the task force on process mining [3] initiated the process discovery contest (PDC) [10]. This PDC has now been organized in the years 2016 [8], 2017 [9], and 2019 [7]. In general, a PDC works as follows. The contestants are provided with a collection of 10 logs, called the *training logs*. For each of these training logs, the contestants have to discover some process model of their choosing. It is important to mention here that the type of process model is not prescribed by the PDC, it could be any process model. The contestants are also provided with a second collection of 10 logs, called the *test logs*: one test log for every training log. Using the process model as discovered from a training log, the contestants should then check for every trace in the corresponding test log whether the discovered process model agrees with it. As such, they should classify every trace in the test log as positive (process model agrees with it) or negative (otherwise). This classification is then compared to the ground truth classification, which results in a score: the number of

✉ H. M. W. Verbeek
h.m.w.verbeek@tue.nl

[1] Eindhoven University of Technology, Eindhoven, The Netherlands

correct classifications. Either the contestant with the highest score wins the contest directly, or the contestants with high enough scores (at least 95% of the highest score) go to a second round which involves a jury that assesses the discovered models and decides on the winner.

This paper introduces the novel *Log Skeleton Visualizer* (Visualizer for short), which visualizes a log as a so-called *log skeleton*. Such a log skeleton contains a collection of nine relations as they are contained in the log: (1) three unary activity relations, (2) five binary activity relations, and (3) one equivalence activity relation. The Visualizer shows us these nine relations in a comprehensible way as a directed graph. Using the Visualizer, we were able to create process models that classify 898 out of the 900 traces from the test logs of the PDC 2019 correctly. This enabled us to win this contest, as the jury considered our process models to be better than the process models of the only other contestant that classified at least 853 (95% of 898) traces correctly.

The remainder of this paper is organized as follows. Section 2 introduces log skeletons. Section 3 introduces the Visualizer as it is implemented in ProM 6.9. Section 4 shows how we have used the Visualizer on the PDC 2019 logs. Section 5 discusses log skeletons and their evolution. Section 6 concludes the paper.

2 Log skeletons

We start with a definition of a log. For sake of simplicity, an event in this log simply corresponds to an occurrence of an activity, that is, no other attributes are involved. For sake of convenience, we assume that some total order $<$ exists on the activities. As a result of this, a collection of activities always contains a smallest activity.

Definition 1 *(Trace, log)* Let A be a set of activities. A *trace* T over A is a sequence over A, that is, $T \in A^*$. A *log* L over A is a bag[1] (or multi-set) of traces over A, such that every $a \in A$ occurs at least once in the log L.

As we assumed that an event corresponds to an activity, traces may not be unique. We would like a discovered model to capture at least the frequent traces, and if possible the infrequent traces. As a result, frequencies matter while doing process discovery, which explains why we consider a log to be a multiset of traces here.

We first introduce the basic structure for a log skeleton, which contains an extended set of activities, three unary (counting) relations, an equivalence relation, and five binary relations.

[1] A bag is a set which may contain the same element more than once. We allow ourselves to use set notations for bags as well.

Definition 2 *(Log skeleton)* A *log skeleton* S is a 10-tuple $(A, c, h, l, E, R, P, \overline{R}, \overline{P}, \overline{C})$ where:

- A is a set of activities.
- $c : A \to \{1, 2, 3, \ldots\}$.
- $h : A \to \{1, 2, 3, \ldots\}$.
- $l : A \to \{0, 1, 2, 3, \ldots\}$.
- $E \subseteq A \times A$ such that E is an equivalence relation on A.
- $R \subset A \times A$ such that R is irreflexive, antisymmetric and transitively closed.
- $P \subset A \times A$ such that P is irreflexive, antisymmetric and transitively closed.
- $\overline{R} \subset A \times A$ such that $\overline{R}$ is irreflexive and antisymmetric.
- $\overline{P} \subset A \times A$ such that $\overline{P}$ is irreflexive and antisymmetric.
- $\overline{C} \subset A \times A$ such that $\overline{C}$ is irreflexive.

To link a log skeleton to a log, we define the notion of a *valid* log skeleton. A log skeleton is valid for some log if the relations exactly capture some properties in the log.

Definition 3 *(Valid log skeleton)* Let L be a log over some set of activities A, and let $S = (A', c, h, l, E, R, P, NR, NP, NC)$ be a log skeleton. The log skeleton S is called *valid* for L if and only if the following properties hold:

- $A' = A \cup \{|>, []\}$, that is, A' extends A with the artificial start activity $|>$ and the artificial end activity $[]$.
- For every $a \in A'$, it holds that $c(a)$ equals the number of times a occurs in L. Furthermore, $c(|>) = c([]) = |L|$.
- For every $a \in A'$, it holds that $h(a)$ equals the maximal (*highest*) number of times a occurs in any trace $T \in L$. Furthermore, $h(|>) = h([]) = 1$.
- For every $a \in A'$, it holds that $l(a)$ equals the minimal (*lowest*) number of times a occurs in any trace $T \in L$. Furthermore, $l(|>) = l([]) = 1$.
- For every $(a, b) \in E$, it holds that for every trace $T \in L$ it holds that a and b occur equally often in T, that is, two activities are equivalent if they occur equally often in every trace. Furthermore, $(|>, []) \in E$, as they both occur exactly once in every trace.
- For every $(a, b) \in R$, it holds that for every trace $T \in L$ it holds that any occurrence of a is always followed by an occurrence of b in T, that is, activity a has activity b as a *response* [4]. Furthermore, for every $a \in A \cup |>$ it holds that $(a, []) \in R$, that is, any other activity is always followed by the artificial end activity.
- For every $(a, b) \in P$, it holds that for every trace $T \in L$ it holds that any occurrence of a is always preceded by an occurrence of b in T, that is, activity a has activity b as a *precedence* [4]. Furthermore, for every $a \in A \cup \{[]\}$ it holds that (a,$|>$) $\in P$, that is, any other activity is always preceded by the artificial start activity.

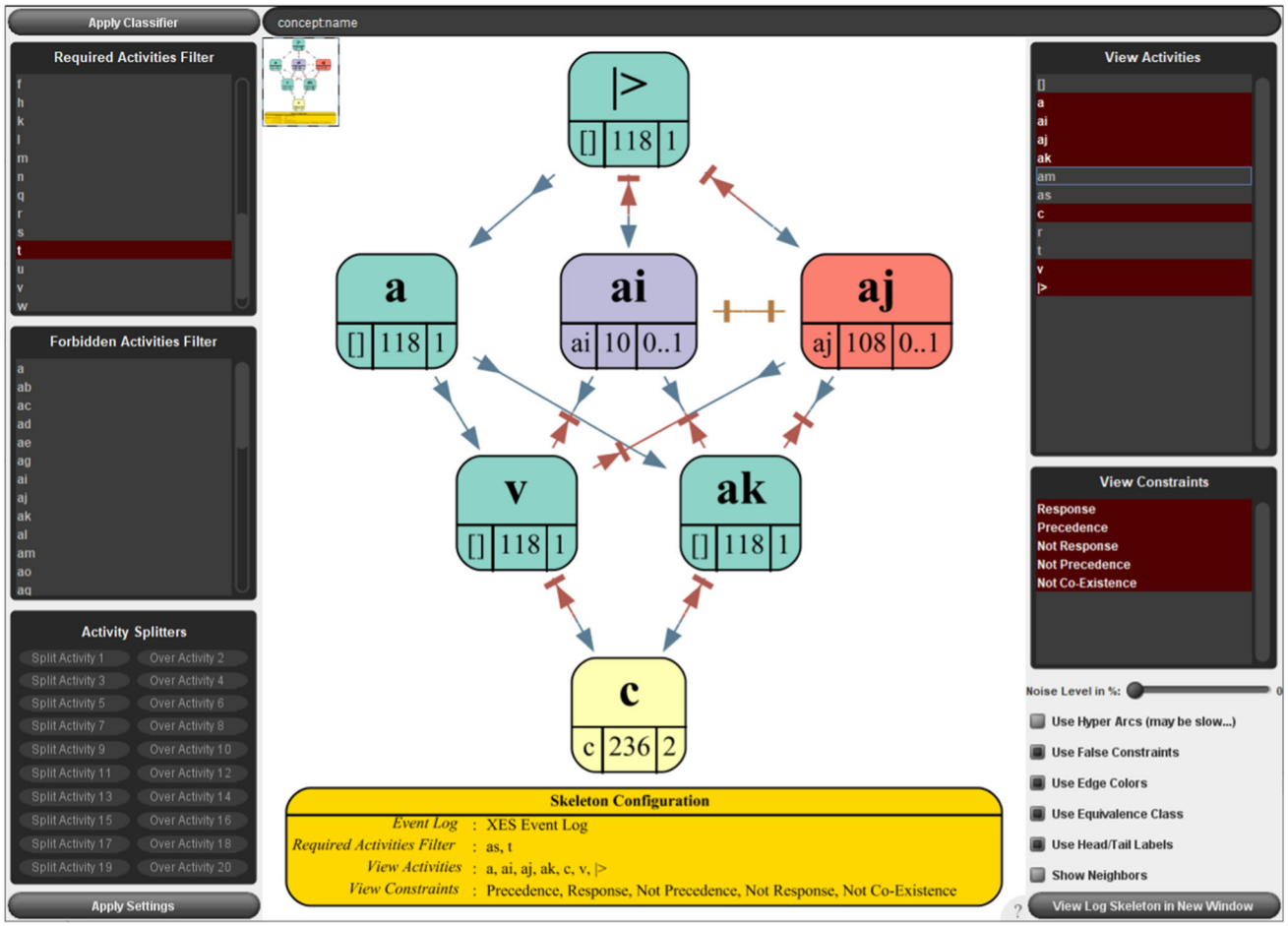

Fig. 1 An example log skeleton shown using the Log Skeleton Visualizer in ProM 6.9

- For every $(a, b) \in \overline{R}$, it holds that for every trace $T \in L$ it holds that any occurrence of a is never followed by an occurrence of b in T, that is, activity a has the *absence* of activity b as a *response*. Furthermore, for every $a \in A \cup \{[]\}$ it holds that $(a, |>) \in \overline{R}$, that is, any other activity is never followed by the artificial start activity.
- For every $(a, b) \in \overline{P}$, it holds that for every trace $T \in L$ it holds that any occurrence of a is never preceded by an occurrence of b in T, that is, activity a has the *absence* of activity b as a *precedence*. Furthermore, for every $a \in A \cup |>$ it holds that $(a, []) \in \overline{P}$, that is, any other activity is never preceded by the artificial end activity.
- For every $(a, b) \in \overline{C}$, it holds that for every $T \in L$ it holds that if a occurs in T then b does not occur in T. That is, activity a has activity b as a *not co-existence* [4].

In the remainder of this paper, we will only consider log skeletons which are valid for the log at hand.

3 Log Skeleton Visualizer

The Log Skeleton Visualizer (or just Visualizer for short) as used for the PDC 2019 has been implemented in ProM 6.9[2] [14]. Figure 1 shows an example log skeleton as a directed graph (a *log skeleton graph*) in the Visualizer. The example log skeleton was built from the log 4 of the PDC 2019 by selecting activities 'as' and 't' as required (see Sect. 3.2) and selecting only the activities as shown by the log skeleton while deselecting the 'Show Neighbors' option (see Sect. 3.3). Note that below the log skeleton graph a legend is shown, which provides useful information on how this log skeleton was build from the log.

The Visualizer shows the log skeleton graph in the middle, surrounded by three control panels: top, left, and right. The top and left control panels use the log to build a new log skeleton, whereas the right control uses the log skeleton to create a new log skeleton graph, that is, a new visualization of the log skeleton.

[2] ProM 6.9 can be downloaded from http://www.promtools.org.

3.1 Log skeleton graph

3.1.1 Nodes

The nodes in the log skeleton graph correspond to the activities in log skeleton (A'), but also provide information on the three unary relations (c, h, and l) and the equivalence relation E. In the top row, we find the activity name, like a. In the bottom row, from left to right, we find the following:

1. The name of the representative activity for activity a. All activities that have the same representative are equivalent. Any equivalent activity can be the representative activity, but we use the smallest activity (using the total order on A). For sake of convenience, the background color of the activity also indicates the equivalence class: Two activities sharing the same background color (not being white) are equivalent.
2. The total number of times activity a occurred in the log, that is, $c(a)$.
3. An interval containing the minimal and maximal number of times activity a occurred in any trace, that is, $l(a)..h(a)$. If $l(a) = h(a)$, then the interval is simply written as the single value $h(a)$.

3.1.2 Edges

The edges in the log skeleton graph correspond to the five binary relations (R, P, $\overline{R}$, $\overline{P}$, and $\overline{C}$). The symbols on the head and tail of the edges have the following meaning, where *outgoing* (*incoming*) indicates that it is pointing to the *farthest* (*nearest*) node, and where the point of reference for relation is always the node *nearest* to the symbol:

- A (dark blue) outgoing arrowhead without a tee bar indicates a *response* relation, that is, R. For example, 'ai' has 'v' as a *response*, that is, any occurrence of 'ai' is always followed by a 'v'.
- A (dark red) outgoing arrowhead with a tee bar indicates a *not response* relation, that is, $\overline{R}$. For example, 'v' has 'ai' as a *not response*, that is, no occurrence of 'v' is ever followed by an 'ai'.
- A (dark blue) incoming arrowhead without a tee bar indicates a *precedence* relation, that is, P. For example, 'c' has 'v' as a *precedence*, that is, any occurrence of 'c' is always preceded by a 'v'.
- A (dark red) incoming arrowhead with a tee bar indicates a *not precedence* relation, that is, $\overline{P}$. For example, 'v' has 'c' as a *not precedence*, that is, no occurrence of 'v' is ever preceded by a 'c'.
- A (dark yellow) tee bar indicates a *not co-existence* relation, that is, $\overline{C}$. For example, 'ai' has 'aj' as a *not co-existence*, that is, no 'ai' is ever followed or preceded by an 'aj'.

To show only the most relevant relations, the *response* and *precedence* relations have priority over the *not co-existence* relation, which in turn has priority over the *not response* and *not precedence* relations. As a result, if a is always followed by b and a is never preceded by b (which may happen), then only the fact that a is always followed by b is shown. This is done as we consider the *response* relation to be stronger than the *not precedence* relation. Observe that we can intuitively rephrase the *not precedence* relation like follows:

> If both a and b occur in a trace, then any a is followed by any b.

Although this is not exactly a conditional *response*, it resembles it to a large extent, especially if there is at most one a and a at most one b in every trace. Likewise, if a and b never occur together, this is shown, instead of the fact that no a is ever followed by a b, which then also holds.

3.1.3 Relation reductions

To avoid showing too many relations, the log skeleton graph does not contain all possible relations. As mentioned, the *response* and *precedence* relations are transitive. That means we can perform a transitive reduction on them, which may result in less relations shown in the graph. We can then still conclude whether one activity always follows or always precedes another. For example, as |> is always followed by 'a', and 'a' is always followed by 'c', we can still conclude that |> is always followed by 'c', even though that relation is not shown in the graph.

The *not response* and *not precedence* relations are not necessarily transitive. For example, consider the log with just three traces: a first one with a followed by b, a second one with b followed by c, and a third one with c followed by a. Clearly, from the facts the a never follows b and b never follows c we cannot conclude that a never follows c. Nevertheless, we consider such a situation to be an exception, and to avoid showing too many relations in the graph, we apply the transitive reduction on these relations as well. We agree that in theory this is not always perfect (some relations which should be shown may not be shown), but we feel that in practice it works much better (as it may remove many relations, and because the shown relations are still valid).

The *not co-existence* relation can optionally be reduced by restricting them to the representative activities. For example, if there would have been an additional activity 'ja' which would have been equivalent to 'aj', and assuming that 'aj' < 'ja', then only the *not co-existence* between 'ai' and 'aj' would be shown. But we could easily deduce that there is

also a *not co-existence* between 'ai' and 'ja', even though the latter relation is not shown in the graph.

3.2 Log controls

The top panel contains a text field where we can specify which event attributes should be used for the activity names in the log skeleton. So far, we have assumed that an event *is* the activity name, but in reality an event often *contains* the activity name among other attributes. In the example, the attribute 'concept:name' is used for the activity name. Typical values for this attribute include 'a', 'ai', 'c', and 'v'. If multiple attributes are specified, the activity name is obtained by concatenating all values separated by '+' signs.

The left panel contains controls that we can use to filter the log and build a log skeleton from this filtered log. In the example, the activities 'as' and 't' have been selected as a required activities, which means that any trace that does not contain both activities is filtered out. In a similar way, by selecting an activity as a forbidden activity, we filter out those traces that do contain the selected activity.

The 'Activity Splitters' allow us to set splitters, more on this in Sect. 4.2. For now, it is sufficient to know that by using splitters we can rename multiple occurrences of the same activity in a trace to different activities. As an example, if a occurs twice in every trace, then we can rename some to $a.0$ and others to $a.1$.

The 'Apply settings' button at the bottom allows us to build a new log skeleton from the log using the settings as provided in the left control panel. As a side effect, selecting this button also creates three new logs in the workspace of ProM 6.9: a log containing the traces that passed the filter ('In'), a log containing the traces that did not pass the filter ('Out'), and a log containing the traces that passed the filter with the splitters applied to it ('Split'). These logs can be accessed in the 'All' tab in ProM 6.9, and can be used for further analysis (like using the Visualizer) in ProM 6.9.

3.3 Log skeleton controls

The right panel contains controls that allow us to change the current log skeleton graph. For example, we can select which activities to show ('View Activities') and which relations ('View Constraints'). We can also set several options, which include noise levels[3], whether to use hyper edges if possible ('Use Hyper Arcs'), whether to use edge colors ('Use Edge Colors'), whether to not reduce the *not co-existence* relation to only representative activities ('Use Equivalence Class'), etc.

At the bottom, we find a single button that allows us to open a new window for the current log skeleton graph. This allows us to easily compare two (or more) different log skeletons graphs.

3.4 Behind the scenes

3.4.1 Building the log skeleton for the log

To build log skeleton S for a provided log L, the Visualizer needs to build the nine relations. First, the log L is filtered on the required activities, the forbidden activities, and updated on the splitters. Every trace in this filtered log is then extended with the artificial start and end activities. This results in an extended filtered log L' that is used in the following steps.

The unary relations (c, h, and l) are quite straightforward, as they only require to maintain three counter for every activity.

The binary relations take more effort. Basically, for every event e corresponding to an activity a in some trace in L', we first build the set of activities that follow event e in the trace, say $TF(e)$, and the set of activities that precede event e in that trace, say $TP(e)$. Using these, we can then update set of activities $AF(a)$ ($AP(a)$) that always follow (precede) activity a, and a set of activities $SF(a)$ ($SP(a)$) that sometimes follow (precede) activity a. To update $AF(a)$ and $AP(a)$, we use the set intersection: $AF(a) \cap TF(e)$ and $AP(a) \cap TP(e)$, whereas for $SF(a)$ and $SP(a)$ we use the set union: $SF(a) \cup TF(e)$ and $SP(a) \cup TP(e)$. This provides us with sufficient information in the end for the *response* and *precedence* relation:

- $R = \{(a, b) \in A' \times A' | b \in AF(a)\}$ and
- $P = \{(a, b) \in A' \times A' | b \in AP(a)\}$.

For the *not response* (and *not precedence*) relation, we take for every event in L' the set difference between the set of all activities (A') and the set of activities that follow (precede) event e. This results in the set of activities that do not follow (not precede) event e. As a result, we can also update the set of activities $NF(a)$ ($NP(a)$) that never follow (never precede) activity a, and use the set intersection: $NF(a) \cap (A' \setminus TF(e))$ ($NP(a) \cap (A' \setminus TP(e))$). In the end,

- $\overline{R} = \{(a, b) \in A' \times A' | b \in NF(a)\}$ and
- $\overline{P} = \{(a, b) \in A' \times A' | b \in NP(a)\}$.

For the *not co-existence* relation, we use $SF(a)$ and $SP(a)$ in the end:

- $\overline{C} = \{(a, b) \in A' \times A' | b \notin SF(a) \cup SP(a)\}$.

[3] These noise levels indicate the percentage of cases for which the relation may not hold. As an example, if the noise level of the *response* relation is set to 5%, then activity a has activity b as a *response* if at least 95% of all occurrences of activity a are followed by some occurrence of activity b. As noise levels were not used for the PDC 2019, we consider this option as out-of-scope for this paper. In the entire paper, we simply assume the noise level to be set to 0% everywhere.

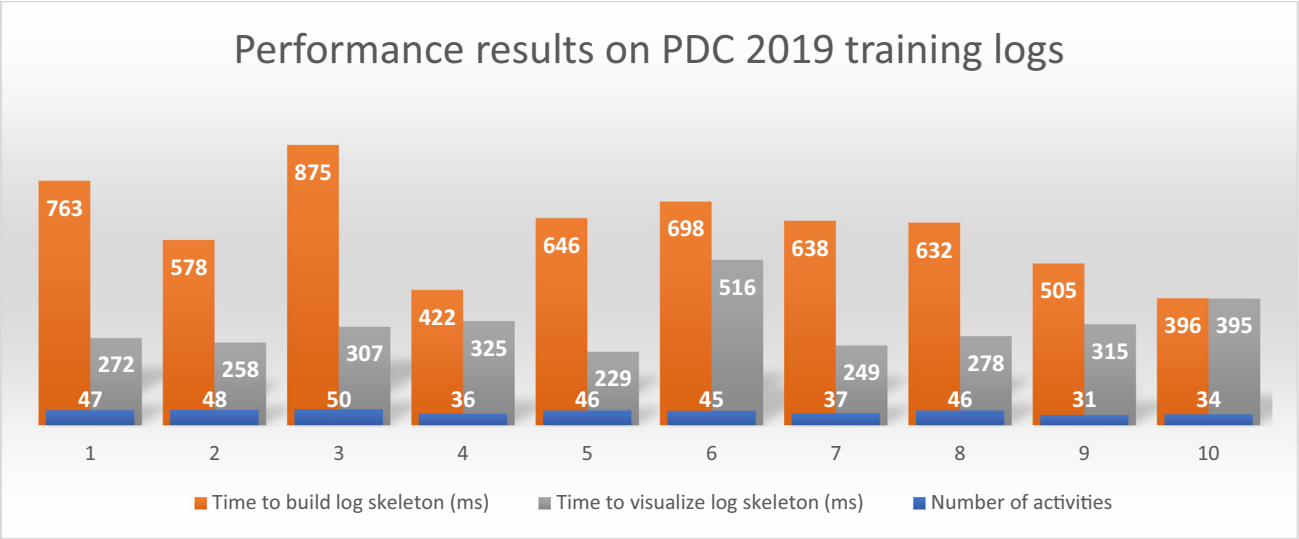

Fig. 2 Time it takes to build and visualize the initial log skeleton for all PDC 2019 logs

The equivalence relation is maintained by keeping a trace count $C \in T \rightarrow \{0, 1, 2, 3, \ldots\}$ for every activity. This trace count maps every trace onto the number of times the activity occurred in that trace. In the end, if activities a and b have identical trace counts (like $C_a = C_b$), then $(a, b) \in E$.

After having built the entire log skeleton like this, the transitive reductions are applied on R, P, $\overline{R}$, and $\overline{P}$, and, if selected, the representative reduction is applied on $\overline{C}$.

3.4.2 Visualizing the log skeleton graph

After the log skeleton has been built, the corresponding log skeleton graph can be shown.

First, we determine the set of candidate activities, say CA. Initially, CA is set to the set of activities as selected ('View Activities') by us. Second, we determine from R, P, $\overline{R}$, $\overline{P}$, and $\overline{C}$ the set of candidate relations, say CR. A relation (a, b) is included in CR if

- the relation type ('View Constraints') is selected and
- $\{a, b\} \subseteq CA$ *or* the 'Show Neighbors' option has been selected and $\{a, b\} \cap CA \neq \emptyset$.

In the latter case, the activity not in CA is considered a *neighbor*: an unselected activity that has a relevant relation to a selected activity. Third, we add the neighbor activities to CA. Finally, all other settings are applied and the log skeleton graph containing CA as nodes and CR as relations is shown. Selected activities will be shown with a border, neighbor activities will be shown without a border.

3.5 Performance results

Figure 2 shows the time[4] it takes to build (top left number, orange bar) and visualize (top right number, gray bar) the initial log skeleton (excludes any filtering) for any of the 10 PDC 2019 training logs, and the number of activities (bottom number, blue bar, includes the two artificial activities) in these logs. For sake of completeness, we mention that each training log contains 700 traces, that the number of events varies from 3728 (log 5) to 18,485 (log 9), and that we ran everything three times and selected the largest time for every training log. As we can see, it takes less than a second to build the initial log skeleton for any of the training logs and also less than a second to visualize it. When combined, it seems fair to say that the Log Skeleton Visualizer takes about a second for any PDC 2019 training log. There also seems to be a positive relation between the number of activities and the time it takes to build the initial log skeleton.

[4] All runs were done on a 64-bit Windows 10 laptop with an Intel(R) Core(TM) i7-7700HQ CPU running at 2.80 GHz with 32 GB of RAM.

4 Using the Log Skeleton Visualizer on log 4 of the PDC 2019

To show how we can use the Log Skeleton Visualizer to discover a process model, we use it on log 4 of the PDC 2019) [7].

4.1 Training log 4 of the PDC 2019

This training log contains 700 traces, 7065 events, and 34 different activities (note that the artificial activities | > and [] are not included in the log). The additional characteristics of this log as provided by the organizers are: *Noise*, *Recurrent activities*, *Inclusive choices*, and *Unbalanced paths*.

4.1.1 Noise

20% of the traces (that is, 140 traces) are truncated at the tail. As a result, when replaying the trace on the process model, the trace may not lead to the end of the process, but stop somewhere in the middle. In principle, this makes the discovery of the process model more complex. Note, however, that this may affect the set $TF(e)$ (activities following event e in the trace) while building the log skeleton, but not the set $TP(e)$ (activities preceding event e in the trace): For any event e in the log, $TP(e)$ will be correct. As a result, no precedence-based relation will be incorrect because of this particular type of noise.

4.1.2 Recurrent activities

The process model may contain multiple activities with the same name, which is also known as duplicate activities. As a result of this, we need to match an event to either one of the activities in the model, which makes the discovery of the process model more complex.

4.1.3 Inclusive choices

At some points in the model, it is possible to execute any positive number of the branches that follow. Usually, the choices are restricted to only one branch or all branches, but for an inclusive choice it is possible to choose, for example, 3 out of 5 branches as well. This also makes the discovery of the process model more complex.

4.1.4 Unbalanced paths

For every non-inclusive choice (where exactly one possible branch needs to be chosen), one branch has a probability of 90% to be chosen, whereas the other branches together have a probability of only 10%. The branch with 90% probability models the so-called *happy flow* here, whereas the other branches model exceptions. The discovery of the exceptions in the process model is typically more error-prone, as less data (less number of traces) are available for them. As an example, if we only have a few traces, then concurrency may be hard to discover.

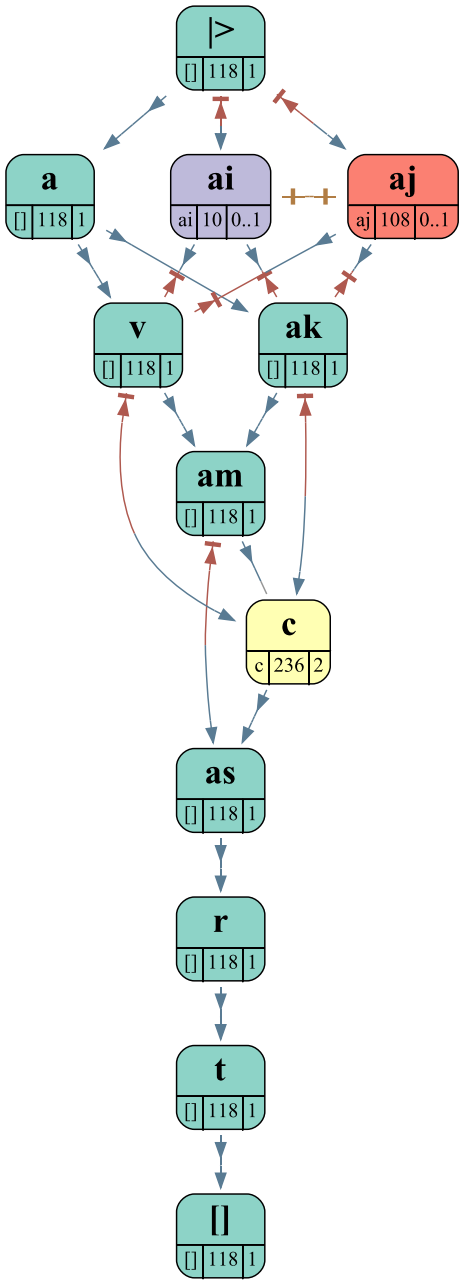

Fig. 3 Log skeleton for the sublog from training log 4 from the PDC 2019 where 'as' and 't' are required

4.2 Using the visualizer

Figure 3 shows a log skeleton graph for a sublog that we encountered while discovering a workflow net for this train-

ing log. This sublog contains only those traces that include both the activities ‘as’ and ‘t’, that is, both ‘as’ and ‘t’ are required activities for this sublog. As the log skeleton shows, 118 out of the 700 traces were included in this sublog, which is a substantial part.

This log skeleton graph clearly shows that the activity ‘c’ occurs exactly twice in every trace and that both occurrences are always preceded by ‘ak’ and ‘v’ and always followed by ‘as’. Furthermore, it shows that activity ‘am’ is always followed by ‘c’. From this, we can construct the hypothesis that one occurrence of ‘c’ is concurrent to ‘am’ while the other follows ‘am’.

To be able to check this hypothesis, we split activity ‘c’ over itself. Splitting an activity a over an activity b results in renaming every occurrence a that occurs after the first occurrence of b to $a.1$ and every other occurrence of a to $a.0$. By splitting ‘c’ over itself, the first ‘c’ in the trace is renamed to ‘c.0’, whereas every other ‘c’ is renamed to ‘c.1’. Figure 4 shows the resulting log skeleton, which confirms our hypothesis.

From this log skeleton, the process for these 118 traces can be easily deduced. First, we do activity ‘a’ concurrent with a choice between ‘ai’ and ‘aj’. Second, we do activity ‘ak’ concurrent with ‘v’. Third, we do activity ‘am’ concurrent with ‘c’. Fourth, we do activity ‘c’, followed by ‘as’, ‘r’, and ‘t’.

This ‘divide-and-conquer’ strategy was followed for all 10 PDC 2019 logs. By selecting appropriate required and forbidden activities, and sometimes using splitters as well, often clear processes were discovered for sublogs that could easily be captured by workflow nets. As an example, Fig. 5 shows the workflow net for the process we have just discovered. Likewise, requiring activities ‘au’ and ‘t’ in log 4 results in another clear log skeleton that covers another 132 traces of the log, and requiring ‘ao’ and ‘t’ would cover another 126 traces. The resulting workflow nets were then merged into a single workflow net by hand. We would then check conformance of the log on the resulting workflow net and follow up on any remaining issues.

As mentioned earlier, the PDC 2019 did not prescribe to use workflow nets. We could also have used the log skeletons themselves, like we did for the PDC 2017. However, like the PDC 2017, the PDC 2019 did include a jury who would look at the discovered models for the top ranking submissions and rank the submissions based on their findings. For people in the process mining field, workflow nets are well-known and are therefore easier to grasp than the (novel) log skeletons. For the PDC 2017, our submission with log skeletons was outranked by a submission with workflow nets, even though our log skeletons classified more traces correctly than these workflow nets. For this reason, we submitted workflow nets instead of log skeletons for the PDC 2019.

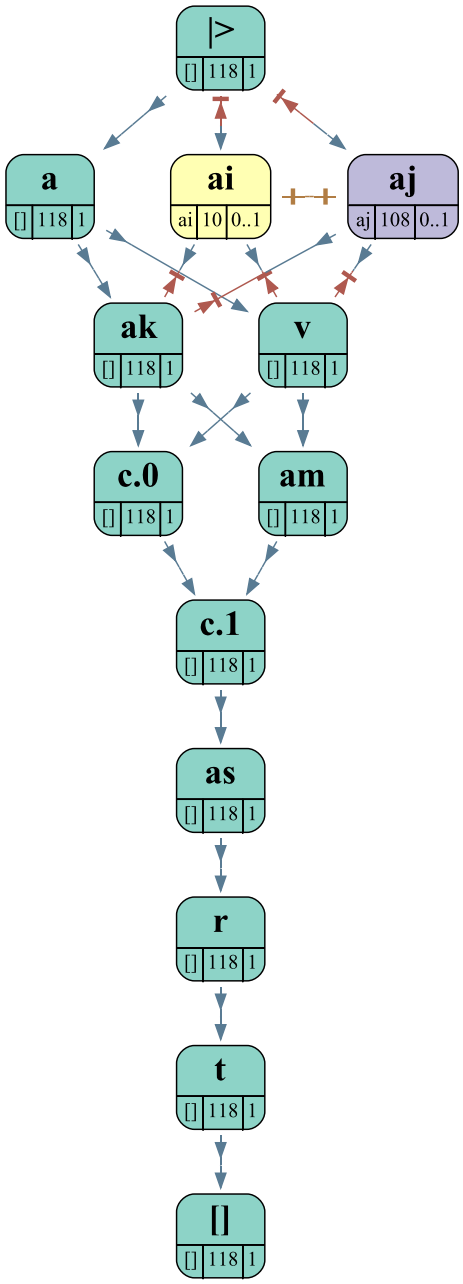

Fig. 4 Log skeleton for the sublog after splitting ‘c’ over itself

4.3 Resulting process model

Based on many such insights as provided by the Visualizer, we were able to construct the *sound* workflow net [1] as shown in Fig. 6 for training log 4 of the PDC 2019. Basically, discovery was done by inspecting relevant sublogs using the Visualizer, creating as many subnets, and then (manually) merging all subnets into a single workflow net. For the record, we did not rely on the Visualizer only in this discovery, but also sometimes used the event log explorer [13] for additional

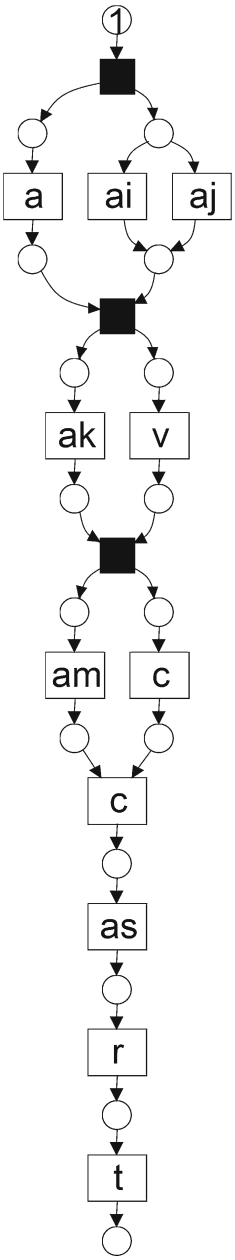

Fig. 5 Workflow net for the log skeleton shown in Fig. 4. The black transitions do not correspond to any activity and are required for routing purposes

insights, and the conformance checker [6] to check whether there were indeed 140 noisy traces.

To convert a log skeleton in general into a workflow net is not a trivial exercise [11]. An attempt was made to do this conversion using an automated evolutionary approach [12], but this only showed that sometimes quite different workflow nets would equally fit some log skeleton. As a result, this conversion (and hence part of the discovery) was done manually.

Fortunately, the log skeletons that resulted from various sublogs were typically easy to convert into a workflow net manually (see also Figs. 4, 5). Although this conversion itself did not take that much time (a few minutes at most), the merging was much more of a challenge, especially for a workflow net that is discovered late, as this requires merging it into a workflow net that is already quite complex. As a result, most time (like many minutes) during the manual discovery was spent in merging the workflow nets into a single workflow net.

In the near future, it may be possible to convert log skeletons that satisfy certain additional criteria automatically. The log skeleton as shown in Fig. 4 is an example log skeleton that should satisfy these additional criteria. An example criterion could be that the number of traces add up in a construct like for the activities 'ai' and 'aj'. The automation of the merging is, however, a different issue. We could try to use existing synthesis techniques like the ones used in [5], but these techniques take a lot of run-time (see also [5]). Doing it manually by an expert may then be a better option.

In the discovered workflow net, four different transitions are needed to model the behavior of activity 'c'. Apparently, this activity occurs in four different contexts in the log. Furthermore, activity 'c' is not the only activity that require multiple transitions. For example, the activities 'aj', 'as', and 't' all require two transitions.

Using this workflow net, we were able to classify all 90 traces for the corresponding test log correctly: 48 were classified as positive and 42 as negative. Furthermore, this log exactly classifies 140 out of the 800 traces in the training log itself as not fitting, which is in accordance with the fact that 140 traces in the training log are known to contain noise. As a result, we may assume that this workflow net is a good representation of the behavior as present in the training log.

5 Discussion

5.1 Why log skeletons?

The work on log skeletons started with the process discovery contest of 2017 (PDC 2017) [9]. Basically, log skeletons were the direct results of trying to think *outside the box* for this contest. As we assumed the organizers of this contest to have a bias toward imperative process models like workflow nets, they were also assumed to have a bias toward logs containing process constructs that are typically hard to capture in such process models. Therefore, creating a different kind of process model might be beneficial to discover and classify these hard process constructs.

To create this other kind of process model, we use the observation that most workflow net discovery algorithm rely on the so-called *directly follows relation* as contained in the log, that is, they only use the information that an activity is directly followed by another activity. As a result, these discovery algorithm only use the *sequential* information as captured in the log, but not the *circumstantial* information [11]. The six log skeleton relations (the equivalence relation

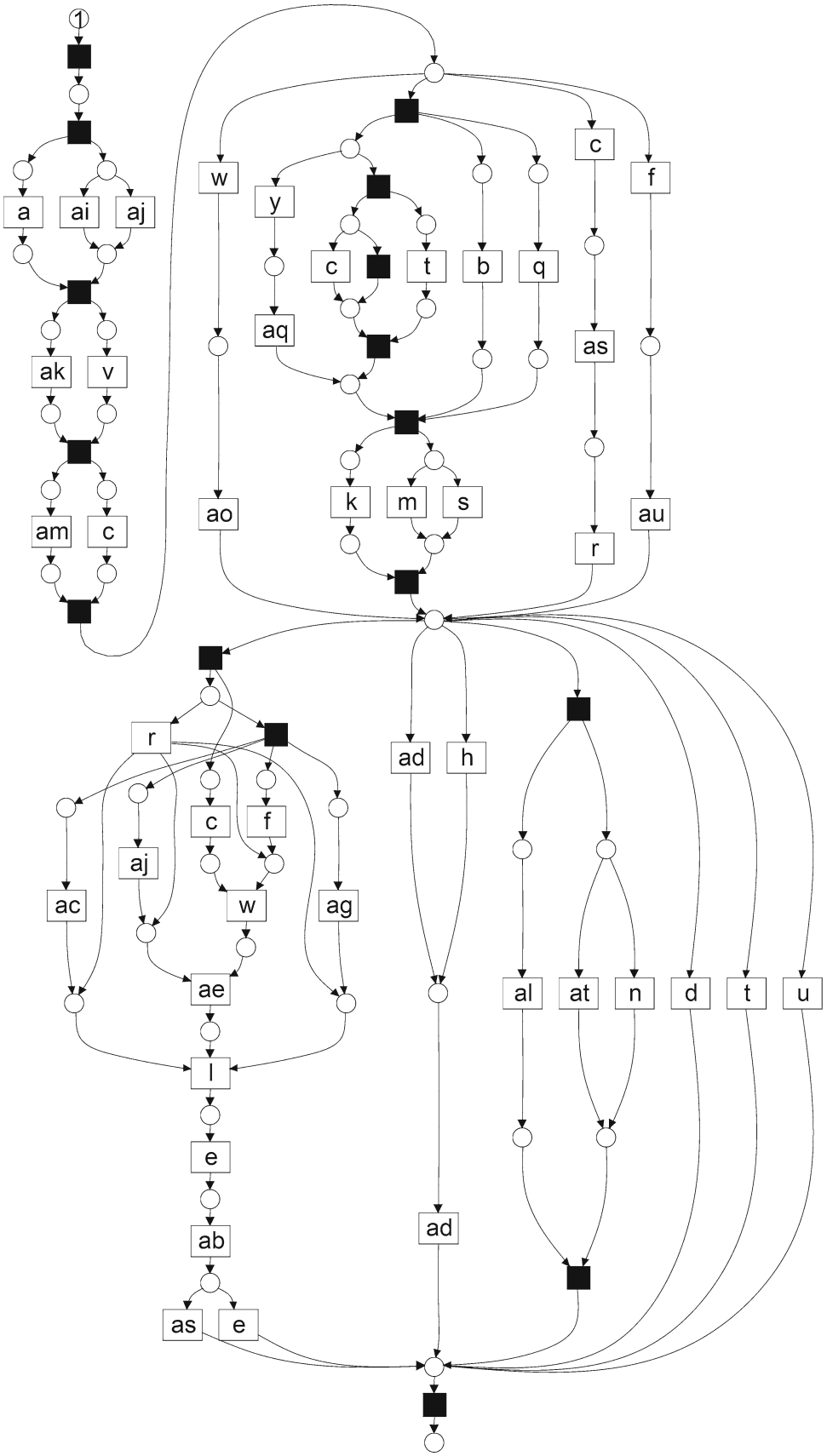

Fig. 6 Workflow net discovered using the Visualizer for training log 4 from the PDC 2019

and the five binary relations) all offer such circumstantial information.

Using a predecessor of the Visualizer [15], we could classify all 200 test traces from the PDC 2017 correctly, and we were the only one to do so. However, this predecessor did not win the PDC 2017, as the jury appreciated the workflow nets submitted by another contestant much more than our log skeletons (see also [15]). For us, this was also an important reason to submit workflow nets rather than log skeletons to the PDC 2019.

5.2 Declare [4]

The realization that the always follows always precedes and never together relations already existed in Declare [4] as *response*, *precedence*, and *not co-existence* followed only after the PDC 2017. As a result, we adopted the Declare nomenclature for these relations, as until then they were names like *always after*, *always before*, and *never together*.

5.3 Duplicate tasks

Using the Visualizer, the divide-and-conquer approach to discover a single workflow net worked for 9 out of 10 logs. The exception was log 7, the result of which is shown in Fig. 7. In the end, the highlighted part in the model was left for us to discover, but because it contained the duplicate activities 'd', 'f', and 'r', we could not single this part out using the filters and the splitters. Apparently, for this log, the splitters are insufficient to *unduplicate* these duplicate activities from the activities in the remainder of the model. As a result, a better way of handling duplicate activities is left for future work.

5.4 Loops

Loops are an issue with relations like *response* and *precedence*. For example, while an occurrence of a may always precede an occurrence of b in a single iteration of some loop, the occurrence of activity a in the next iteration of that loop by definition follows an occurrence of activity b in the current iteration. Currently, we have no way to detect or visualize loops in a log skeleton. This is also left for future work, but a first idea may be to allow us to select activities that jump back in the process, that is, activities that initiate a new instance of a loop. While building the set of activities $TF(e)$ ($TP(e)$) that follow (precede) some event e in the trace, we could then stop at such a back-jumping activity. The log skeleton could then show that a always precedes b under the assumption that some activity c starts a new instance of the loop that contains a and b.

An important observation to make here is also that *none of the PDC 2019 logs actually contained loops*. Although the initial information from the organizers of the PDC 2019 mentioned that the logs 2, 5, 6, 7, and 8 contain loops, this was later corrected by them because the published logs did not contain loops. Clearly, the absence of loops was not a disadvantage for our submission using the Log Skeleton Visualizer. In our defense: Using the Visualizer, we actually discovered that these logs did not contain loops, which triggered the correction as made by the organizers.

5.5 Relation reductions

The relation reductions in this version of the Visualizer occur at the end of building the log skeleton, that is, before the log skeleton is visualized. Hence, selected activities cannot be taken into account while doing these reductions. As a result, sometimes some relations are not shown as they were reduced because of other relations, which may not be shown now because some required activity is not selected. In the future, we want to move these reductions from the log skeleton builder to the Log Skeleton Visualizer, as then we can take the selected activities into account, which would show more relevant relations.

6 Conclusion

This paper has introduced a new *declarative* process model called log skeletons. Log skeletons are more simple than Declare models [4] and can be computed easily. Like Declare models, log skeletons offer us *circumstantial* information, whereas typical discovery techniques result in *imperative* process models that typically offer *sequential* information [11]. As such, log skeletons offer a readily available alternative view on the process model that underlies the log at hand.

A visualizer for log skeletons has been implemented in ProM 6.9 [14], called the Log Skeleton Visualizer, or Visualizer for short. This Visualizer allows us to create sublogs on the fly, to automatically discover a log skeleton for the currently created sublog, and to change the view on the discovered log skeleton.

The Visualizer has been used for discovering workflow nets [1] for the Process Discovery Contest of 2019 (PDC 2019) [7]. Using the Visualizer, we could gain sufficient insights in the training logs of the PDC 2019 to create a sound workflow net for every training log. In the end, these sound workflow nets classified 898 out of the 900 test traces correctly, which was more than any other submission, and they were considered to be more comprehensible than process models from any other submission that classified sufficiently many traces correctly. This made these sound workflow nets the winning submission of this contest, which

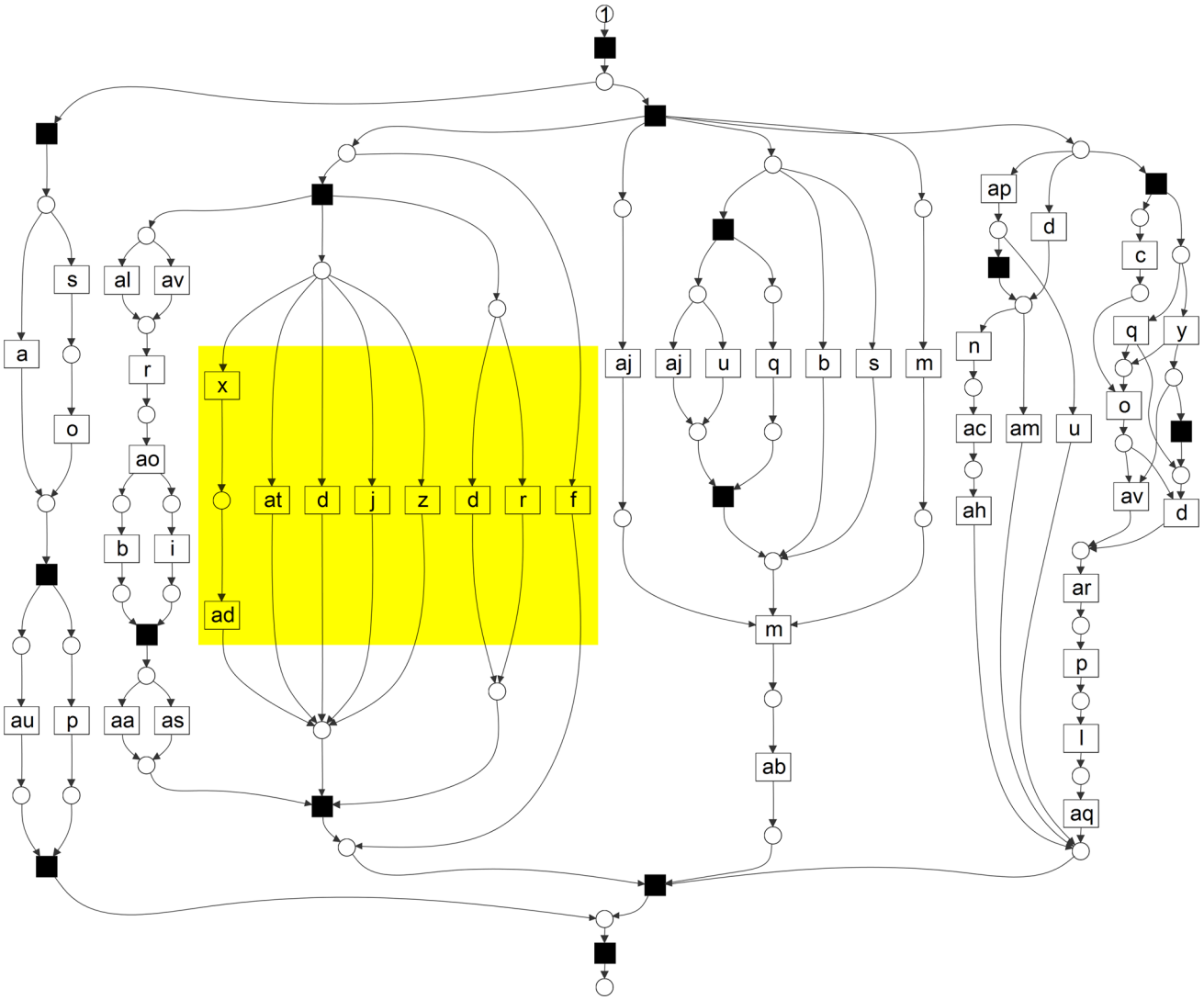

Fig. 7 Workflow net discovered using the Visualizer for training log 7 from the PDC 2019, with a problematic part highlighted. Note that the activities 'd', 'f', and 'r' also appear in other parts of the net. As a result, if we would filter on these activities, other parts of the net would also be affected, which complicates the discovery of this part

shows the potential value of log skeletons and the Log Skeleton Visualizer.

Acknowledgements The author would like to thank the organizers of the entire process discovery contest series: Josep Carmona, Massimiliano de Leoni, Benoît Depaire, and Toon Jouck. Without these contests and without the efforts put into them by the organizers, log skeletons and the Log Skeleton Visualizer would not have been.

References

1. Aalst, W.M.P.v.d.: The application of Petri nets to workflow management. J. Circuits Syst. Comput. **8**(1), 21–66 (1998)
2. Aalst, W.M.P.v.d.: Process Mining: Data Science in Action, 2nd edn. Springer, Berlin (2016). https://doi.org/10.1007/978-3-662-49851-4
3. Aalst, W.M.P.v.d.: Task force on process mining (2019). https://www.tf-pm.org/
4. Aalst, W.M.P.v.d., Pesic, M., Schonenberg, H.: Declarative workflows: balancing between flexibility and support. Comput. Sci. Res. Dev. **23**, 99–113 (2009)

5. Bergenthum, R.: Prime miner—process discovery using prime event structures. In: 2019 International Conference on Process Mining (ICPM), pp. 41–48 (2019). https://doi.org/10.1109/ICPM.2019.00017
6. Carmona, J., Dongen, B.F.v., Solti, A., Weidlich, M.: Conformance Checking. Springer, Berlin (2018). https://doi.org/10.1007/978-3-319-99414-7
7. Carmona, J., de Leoni, M., Depaire, B.: Process discovery contest 2019 (2019). https://icpmconference.org/2019/process-discovery-contest
8. Carmona, J., de Leoni, M., Depaire, B., Jouck, T.: Process discovery contest 2016 (2016). https://www.win.tue.nl/ieeetfpm/doku.php?id=shared:edition_2016
9. Carmona, J., de Leoni, M., Depaire, B., Jouck, T.: Process discovery contest 2017 (2017). https://www.win.tue.nl/ieeetfpm/doku.php?id=shared:edition_2017
10. Carmona, J., de Leoni, M., Depaire, B.T.J.: Process discovery contest (2016). https://www.tf-pm.org/competitions-awards/discovery-contest
11. Fahland, D., Lübke, D., Mendling, J., Reijers, H.A., Weber, B., Weidlich, M., Zugal, S.: Declarative versus imperative process modeling languages: the issue of understandability. In: Halpin, T., Krogstie, J., Nurcan, S., Proper, E., Schmidt, R., Soffer, P., Ukor, R. (eds.) Proceedings of BPMDS 2009 and EMMSAD 2009, *LNBIP*, vol. 29, pp. 353–366. Springer (2009)
12. Hu, X.: Discovering process models from log skeletons using an evolutionary approach. Master's thesis, Eindhoven University of Technology, Eindhoven (2018). https://pure.tue.nl/ws/files/109028572/BIS262_XiaoqingHu_final_thesis.pdf
13. Mannhardt, F.: Multi-perspective process mining. Ph.D. thesis, Eindhoven University of Technology, Eindhoven (2018)
14. Verbeek, H.M.W., Buijs, J.C.A.M., Dongen, B.F.v., Aalst, W.M.P.v.d.: ProM 6: The process mining toolkit. In: Proceedings of BPM Demonstration Track 2010, vol. 615, pp. 34–39. CEUR-WS.org (2010). http://ceur-ws.org/Vol-615/paper13.pdf
15. Verbeek, H.M.W., Carvalho, R.M.d.: Log skeletons: a classification approach to process discovery. CoRR **abs/1806.08247** (2018). arxiv:1806.08247