Log Skeleton Filter and Browser

Eric Verbeek
Department of Mathematics and Computer Science
Eindhoven University of Technology
Eindhoven, The Netherlands
Email: h.m.w.verbeek@tue.nl

***Abstract*—In the area of process mining, an event log is a crucial artifact. However, existing event log visualizers do not really help us in gaining many useful insights into the underlying process, that is, into the process that generated this event log. This paper introduces the *Log Skeleton Filter and Browser* tool, which allows us to gain many such insights. This tool can handle noisy event logs, and still show the *structure* of this event log and its underlying process, that is, the *skeleton* that supports the log and the process.**

I. Introduction

In the area of process mining [1], *event logs* are crucial. For process discovery, one needs an event log to discover a process model from. For process conformance, one needs an event log to replay on the process model at hand. For process enhancement, one needs an event log to enhance the process model at hand.

However, an event log visualizer that allows us to gain insights into the process model that underlies the event log, is not yet available. Typically, the existing event log visualizers visualize an event log by visualizing traces in some way. Although this may be useful in itself, it does not really help us in understanding the underlying process model.

At the same time, the two top submissions of the process discovery contest of 2017 [2] have both used an approach which involved a human. The winner of that contest used an interactive approach, where an experienced user would interactively construct a process model from an event log using some tools. The runner-up approach used an earlier version of the tool presented in this paper. Both submitted approaches performed much better than any submitted approach that included only automated discovery. But if we need to keep a human in the loop, then we should provide that human with tools to support him.

Therefore, this paper presents an event log visualization tool called *Log Skeleton Filter and Browser* that can help us in understanding the structural properties of the log, which in turn can help in understanding the system that generated the log. This tool takes the activities as present in the event log together with some constraints between them as main artifacts, builds a so-called *log skeleton* from these activities and constraints, and visualizes this log skeleton. The tool allows us to filter the event log before creating the log skeleton, and to browse the current log skeleton that was created from the filtered event log.

The remainder of this paper is organized as follows. Section II introduces the three main parts of the tool using a noisy example event log. Section III introduces the part of the tool that visualizes the event log at hand using a so-called *log skeleton*. Section IV introduces the browser controls, which can be used by us to change the view on the current log skeleton. Section V introduces the filter controls, which can be used by us to filter the event log before the log skeleton is derived from it. Section VI concludes the paper.

II. The tool

Fig. 1 showcases the *Log Skeleton Filter and Browser* using an event log from [3] containing 12 activities and 10% noise.

The center part of the *Log Skeleton Filter and Browser* visualizes the selected log skeleton, which is explained in Section III. The rightmost part contains the browser controls, which are explained in Section IV. The leftmost part contains the filter controls, which are explained in Section V.

The tool has been implemented as a visualizer plug-in for event logs in ProM [4]. In this paper we have used the ProM Nightly Build[1] of December 3rd, 2018, with version 6.9.86 of the LogSkeleton package installed.

III. Log Skeletons

A. Activities

The log skeleton shown in Fig. 1 shows in total 14 different activities: the 12 activities as found in the log (`b`, ..., `k`, `E`, and `S`) and two artificial activities which denote the start ($|>$) and the end ($[\,]$) of a trace.

Activity `S` occurs 977 times in the entire event log, and either may not occur or occurs once (0..1) in every trace. Furthermore, `S` is considered to be *equivalent* to `E`, which means that they tend to occur equally often in every trace. The activities $|>$, `E`, and $[\,]$ are also equivalent to `E`, and `E` serves as the ringleader for this equivalence class. As we see that `S` occurs 977 times and `E` occurs 975 times, we know that they do not occur equally often in every trace. The selected noise level determines (5%, in this case) how much differences are accepted. Basically, this noise level guarantees that the accumulated occurrence count differences of both activities in all traces is at most 50 (5% of 1000 traces). Clearly, if the noise level is set to 0%, then no differences are allowed for them to be equivalent.

[1] See http://www.promtools.org/doku.php?id=nightly.

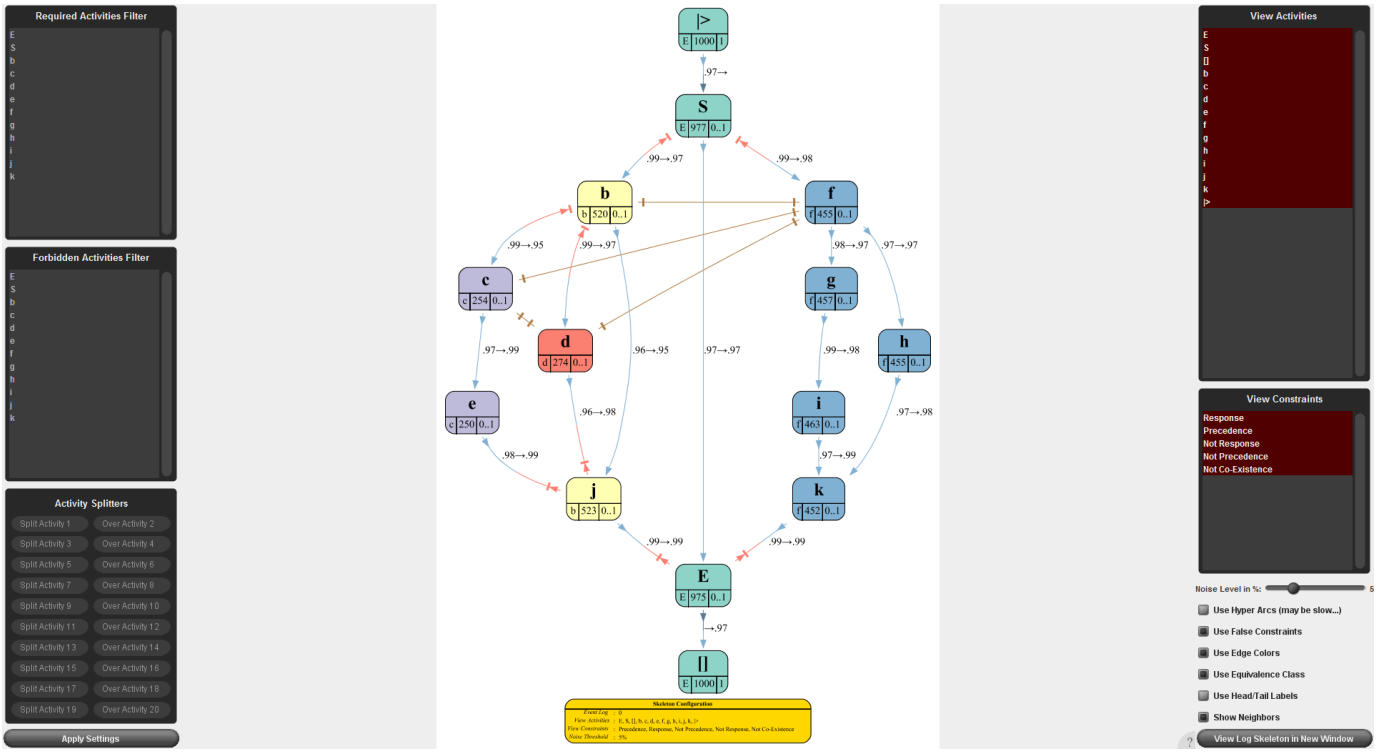

Fig. 1. An example event log viewed with the *Log Skeleton Filter and Browser*.

Equivalent activities use the same background color. As a result, if two activities share the same background color, they are considered to be equivalent. As an example, in Fig. 1 `f`, `g`, `h`, `i`, and `k` are equivalent, with `f` as their ringleader. This suggests that in the process model that underlies the event log these activities occur equally often.

B. Constraints

The log skeleton shown in Fig. 1 shows 5 different kinds of Declare [5] constraints between the activities: *response* (*precedence*) (a blue arrow at the blue tail (head) of an arc), *not response* (*not precedence*) (a red arrow with a red bar at the red head (tail) of an arc), and *not co-existence* (an ocher bar at the head and/or tail of an ocher line). However, in contrast with the original Declare constraints, the constraints also take the noise level into account.

As examples:

- `E` is a response of `k` in 99% (.99) of all occurrences of `k`. This means that 99% of all occurrences of `k` are followed in the same trace by some occurrence of `E`.
- `S` is a precedence of `f` in 98% (.98) of all occurrences of `f`. This means that 98% of all occurrences of `f` are preceded in the same trace by some occurrences of `S`.
- `k` is a not-response of `E` in 99% of all occurrences of `E`. This means that 99% of all occurrences of `E` are *not* followed in the same trace by some occurrence of `k`.
- `f` is a not-precedence of `S` in 99% (.99) of all occurrences of `S`. This means that 99% of all occurrences of `S` are *not* preceded in the same trace by some occurrence of `f`.
- `c` is a not-co-existence of `d` for all occurrences of `c` and vice versa. This means that `c` and `d` never occur in the same trace.

Before the (not) response and (not) precedence relations are visualized, a transitive reduction is performed on them. As a result, although no explicit response constraint from `f` to `i` is shown, implicitly it is there because of the response constraints from `f` to `g` and from `g` to `i`. For the not response and not precedence relations, prior to this reduction, the identity and not-co-existence relations are removed.

Our way of visualizing the constraints differs from the way they are visualized by Declare. In Declare, a dot is used to indicate the 'point of view' for some constraint. In our tool, this role is simply taken by the symbol (arrow, bar, or arrow with bar) put either on the tail or the head. As a result, the symbols replace the dots, but also indicate the types of the constraints. This saves space (and clutter) on the arcs,

The log skeleton also shows that several activities do not occur in the same trace: `f` on the one hand, and `b`, `c`, and `d` on the other hand, and `b` on the one hand and `c` on the other hand. This indicates that there are two choices in the process model: A first between the `f`, ..., `i`, and `k` on the one hand, and `b`, ..., `e`, and `j` on the other hand, and a second between the `c` and `e` on the one hand and `d` on the other hand. Note that the numbers of occurrences of these activities also indicate this, as 455 (`f`) + 520 (`b`) ≈ 977 (`S`), and 254 (`c`) + 274 (`d`) ≈ 520 (`b`).

Although the example at hand does not show this, the not co-existence constraint can also take noise into account.

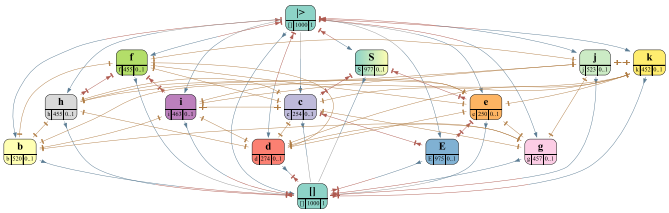

Fig. 2. The example event log viewed with the Log Skeleton Filter and Browser with noise level set to 0.

Assume, for the sake of convenience, that the not co-existence constraint between `b` and `f` has .98 at the `b`-end and .95 at the `f`-end of the constraint. Then 98% of all occurrences of `b` occur in traces where `f` does not occur, and 95% of all occurrences of `f` occur in traces where `b` does not occur. Note that as a result the symmetry of the not co-existence constraint may be broken. For example if we select a noise level of 3%, then `b` will have a not co-existence constraint to `f`, but not vice versa.

C. Configuration

At the bottom of the log skeleton, its configuration is shown. This shows us the main settings that were used to obtain the shown log skeleton from the event log at hand.

IV. The Browser

The browser controls at the right hand side of the log skeleton provide us with means to change the visualization of the current log skeleton. The visualization can be changed by selecting different activities to show, selecting different constraints to show, a slider for the allowed noise level, and a number of visualization options that may be handy. Furthermore, the browser comes with a *View Log Skeleton in New Window* button that allows us to open a new window with the current visualization. This allows us to, say, compare different log skeletons easily.

A. View Activities

This allows us to select which activities are shown. By default, all activities are pre-selected.

B. View Constraints

This allows us to select which constraints are shown. By default, the (not) response/precedence constraints are all pre-selected, and the not-co-existence constraint if the number of these constraints does not exceed 100. The reason for the latter is that the layout algorithm used for the log skeletons may become very slow if many constraints are present. Because of the possible transitive reduction on the (not) response/precedence constraints, the chances of having many of these constraints is considerably lower than the chances of having many not co-existence constraints.

C. Noise Level

This allows us to vary the noise level from 0% to 20%. Allowing a noise level of 50% or more makes hardly any sense, as this could allow multiple contradicting constraints between two activities. As such, we have chosen a maximal noise level of 20%, as this is still some distance away from this problematic 50% level. Fig. 2 shows the usefulness of this slider, as it shows the log skeleton obtained from the event log with noise level set to 0. Obviously, the log skeleton shown by Fig. 1 provides us with more insights about the event log as this log skeleton.

D. Options

1) Use Hyper Arcs (may be slow...): This allows us to visualize a clique of identical (not) response/precedence constraints by a single hyper (not) response/precedence constraint. In some cases this may significantly reduce clutter in the log skeleton as shown. As indicated, this option may make the tool very slow, as it uses a simple recursive algorithm to detect maximal cliques of identical constraints.

2) Use False Constraints: This allows us to ignore the not co-existence constraints (which have no rela direction) when laying the log skeleton out.

3) Use Edge Colors: This allows us to use colors for the constraint edges. If selected, the tail/head of a response/precedence constraint is colored blue, the tail/head of a not precedence/not response constraint is colored red, and the tail/head of a not co-existence constraint is colored ocher. If the constraints are there because of the noise level, the color will be lighter.

4) Use Equivalence Class: This allows us to show only the non co-existence constraints for the ringleaders of the equivalence classes. In Fig. 1, this option is selected. If not selected, the not-co-existence relation between `k` and `e` will also be shown, and many others as well. As equivalent activities typically share the same not co-existence constraints, there is no need to show them all.

5) Use Head/Tail Labels: This allows us to position the tail and head labels (like .98) on the constraint heads and tails. If selected, they are positioned on the tail and head. If not selected, they are positioned near the middle of the edge, like .98→.97 for the response/precedence edge between `f` and `g` (see Fig. 1).

6) Show Neighbors: This allows us to visualize the *neighbors* of selected activities as well. A neighbor of an activity is any other activity that has some selected constraint from or to that activity. This allows us to quickly inspect the constraints of a selected collection of activities. As an example, in the example log skeleton, if we only select `|>` with this option selected, then `S` would also be shown (be it without border), which indicates that `S` typically occurs first. We can then extend the selection with `S`, which causes `b` and `f` to be shown, etc.

V. The Filter

The filter controls at the left hand side allow us to filter the log prior to creating the log skeleton from it. Typically,

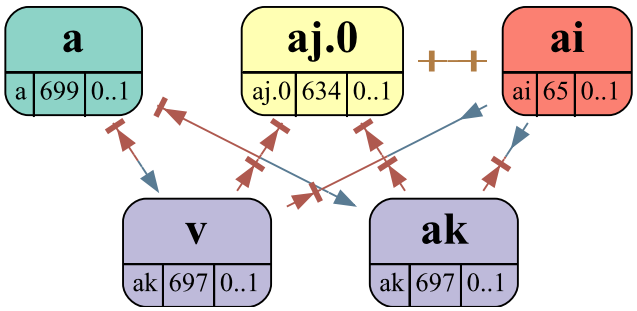

Fig. 3. Some insights for the process discovery contest of 2019.

these controls will result in a different log skeleton. Unlike the browser controls, which take into effect as soon as we change them, we need to select the *Apply Settings* button, which will create and show the new log skeleton using the filter settings as provided.

A. Required Activities

This allows us to select the traces that contain all of the selected activities. If one or more of the selected activities does not occur in a trace, the entire trace will be filtered out.

B. Forbidden Activities

This allows us to select the traces that contain none of the selected activities. If one or more of the selected activities occurs in a trace, the entire trace will be filtered out.

This can be combined with the required activities. Only traces that contain all of the selected required activities and none of the selected forbidden activities will be filtered in.

C. Activity Splitters

This allows us to split activities into multiple activities, which can be useful in case of recurrent activities. In the leftmost field of a row, we can specify the activity he wants to split. In the rightmost field, we can specify the activity he wants to split over.

As an example, assume that we want to split `f` over `b`. Every occurrence of `f` is then renamed to either `f.0` (if in the trace `b` does not precede this occurrence of `f`) or `f.1` (if in the trace `b` does precede this occurrence of `f`). We can also split an activity `f` over itself: The first occurrence of `f` in a trace is renamed `f.0`, all other in that trace are renamed `f.1`.

VI. Conclusion

This paper has introduced the *Log Skeleton Filter and Browser* tool, that can be used by a user to gain insights into the process underlying the event log at hand. Using the tool, important relations between activities can be concluded by us, which allows him to understand the process much better.

The tool can handle noisy event logs. From a noisy event log, a log skeleton was obtained that was a spitting image of the log skeleton that was obtained from the noise-free sibling event log. Of course, the numbers of occurrences of activities in the event log were different, and some constraints required a positive noise level to be detected, but with an appropriate noise level the same constraints were detected, and the same equivalence classes for the activities.

The tool has been used by the author to gain insights into the 10 training event logs of the process discovery contest of 2019 [6]. See, for example, this screencast[2], which explains (to some extent) how the model was discovered for training log 4 using the tool, which included some splitting of recurrent activities. Fig. 3 show some insights for this log of this contest, which indicates that activities `a` and a choice between `ai` and (some occurrence of) `aj` can be executed concurrently, after which the activities `ak` and `v` can be executed concurrently.

However, for some other event logs, using only this tool did not provide sufficient insights. These event logs typically contained recurrent activities, which could not be split successfully with the tool, as it can only split a recurrent activity if there is a certain activity in-between the occurrences of the recurrent activity. For the process discovery contest of 2017, this was sufficient, but for the contest of 2019 it was sometimes not. This shows that other ways of splitting recurrent activities could be very interesting as future work.

In the end, the models discovered by the author from the training event logs of the process discovery contest of 2019 were able to classify 898 of the 900 test traces correctly. Hence, the models achieved a classification accuracy of 99.78% on the final test logs. This made the author's submission to this contest the best-classifying submission, and also shows that many useful insights were obtained using the tool on the training event logs. The two traces that were misclassified belong to models 9 and 10, that indeed contain recurrent activities.

Acknowledgment

The authors would like to thank the organizers of the process discovery contests, as without these contests, this tool would not have been.

References

[1] W. M. P. v. d. Aalst, *Process Mining: Data Science in Action*, 2nd ed. Springer-Verlag, 2016.

[2] J. Carmona, M. de Leoni, B. Depaire, and T. Jouck. (2017) Process discovery contest 2017. [Online]. Available: http://www.win.tue.nl/ieeetfpm/doku.php?id=shared:process_discovery_contest

[3] L. Maruster, A. J. M. M. Weijters, W. M. P. v. d. Aalst, and A. v. d. Bosch, "A rule-based approach for process discovery: Dealing with noise and imbalance in process logs," *Data Min. Knowl. Disc.*, vol. 13, no. 1, pp. 67–87, 2006.

[4] H. M. W. Verbeek, J. C. A. M. Buijs, B. F. v. Dongen, and W. M. P. v. d. Aalst, "ProM 6: The process mining toolkit," in *Proc. of BPM Demonstration Track 2010*, vol. 615. CEUR-WS.org, 2010, pp. 34–39. [Online]. Available: http://ceur-ws.org/Vol-615/paper13.pdf

[5] W. M. P. v. d. Aalst, M. Pesic, and H. Schonenberg, "Declarative workflows: Balancing between flexibility and support," *Computer Science - Research and Development*, vol. 23, pp. 99–113, 2009.

[6] J. Carmona, M. de Leoni, and B. Depaire. (2019) Process discovery contest 2019. [Online]. Available: https://icpmconference.org/process-discovery-contest

[2] See https://www.win.tue.nl/~hverbeek/downloads/ICPM2019Demo.mp4