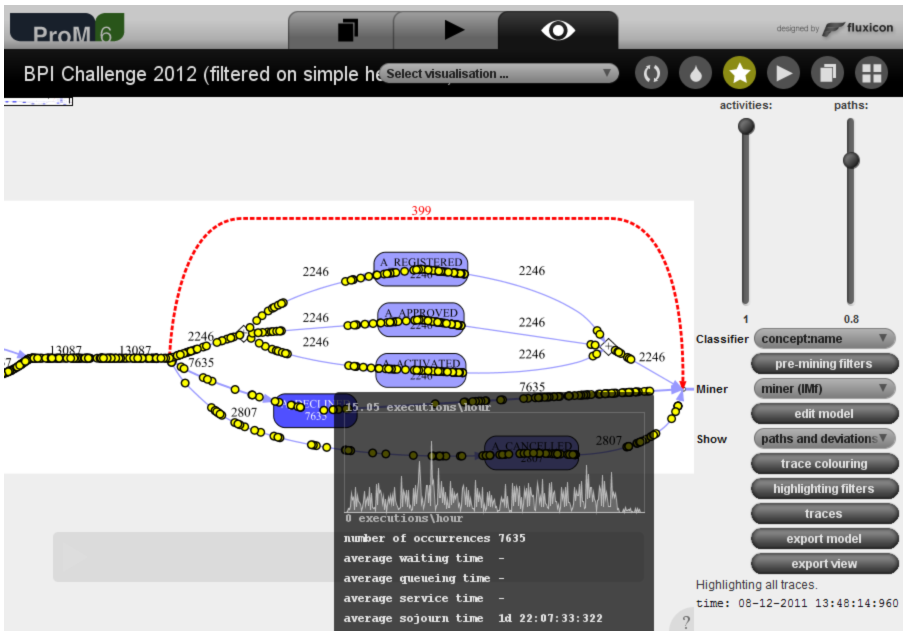
Inductive visual Miner
manual

Sander J.J. Leemans

ProM 6.7, June 7, 2017

Contents



BPI Challenge 2012 (filtered on simple he

```
15.05 executions\hour
0 executions\hour
number of occurrences 7635
average waiting time  -
average queueing time -
average service time  -
average sojourn time  1d 22:07:33:322
```

Classifier concept:name

Miner miner (IMf)

Show paths and deviations

Highlighting all traces.

`time: 08-12-2011 13:48:14:960`

Figure 1: A screenshot of the Inductive visual Miner.

The *Inductive visual Miner* (IvM) [5] combines business process discovery with conformance checking to provide users with an easy-to-use process mining exploration tool. Given an event log, the Inductive visual Miner automatically discovers a process model, compares this model with the event log and visualises several enhancements such as performance measures, queue lengths and animation. All steps are automated: if you change any parameter or change any filter, IvM will automatically update everything necessary.

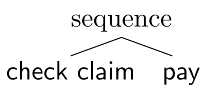
This document describes the user interface of the Inductive visual Miner and touches on its inner workings. For more detail, please refer to [4, Ch.9]. If any further questions arise, please do not hesitate to post them on the ProM forum at `https://www.win.tue.nl/promforum/`.

1 Usage

The Inductive visual Miner is present in the ProM framework, which can be downloaded from `http://promtools.org`. In this document, we describe the Inductive visual Miner as included in ProM 6.7. We do not recommend the use of ProM Lite, as it is meant for education purposes and has a more conservative update strategy.

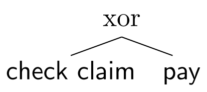
IvM can be started by loading an event log into ProM and applying the plug-in “mine with Inductive visual Miner”. Alternatively, if you already have a process tree and want to compare it to an event log, use the plug-in “Visualise deviations on process tree" to start IvM without the mining controls & options, but with alignments, deviations, animation and highlighting filters.

2 Process Trees

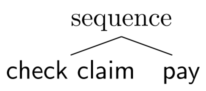
Before we introduce the notation, we first take a closer look at process trees. A process tree is hierarchical and consists of several *nodes* that have *children.* For instance, the process tree 

consists of three nodes: 'sequence', 'check claim' and 'pay'. The nodes 'check claim' and 'pay' are both children of the node 'sequence'.

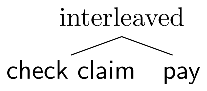
Nodes express behaviour, in terms of their children. For instance, our example process tree expresses that first 'check claim' is to be executed, and afterwards 'pay'. We use six types of nodes:

- xor expresses that one of its children needs to be executed. For instance, the process tree 

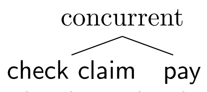
expresses that either 'check claim' or 'pay' must be executed.

- sequence expresses that all of its children need to be executed in order. For instance, the process tree 

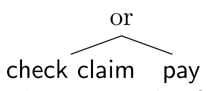
expresses that first 'check claim' and afterwards 'pay' must be executed.

- interleaved expresses that all of its children needs to be executed, but that these executions cannot overlap in time. For instance, the process tree 

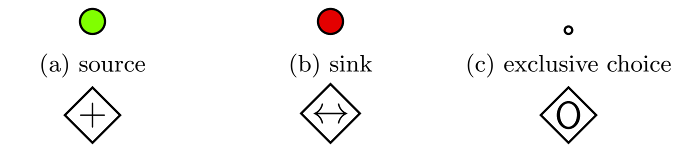
expresses that both 'check claim' or 'pay' must be executed, but whichever is first must finish before the other one can start.

- concurrent expresses that all of its children need to be executed, and they may overlap in time. For instance, the process tree 

expresses that both 'check claim' or 'pay' must be executed independently of one another.

- or expresses that at least one of its children needs to be executed. If multiple children are executed, they may overlap in time. For instance, the process tree 

expresses that either 'check claim' or 'pay' or both must be executed. If 'check claim' and 'pay' are both executed, then these executions are independent.


(a) source (b) sink (c) exclusive choice

(d) concurrency (e) interleaving (f) inclusive choice

Figure 2: IvM model constructs.

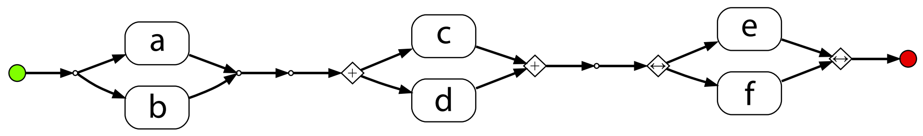


Figure 3: A model in IvM.

- loop expresses that the first child must be executed. After this first child is executed, there is a choice between terminating or executing the second child again followed by the first child and making the same choice again. For instance, the process tree $$\begin{tikzcd} & \text{loop} \arrow[dl, no head] \arrow[dr, no head] & \\ \text{check claim} & & \text{pay} \end{tikzcd}$$ expresses that ‘check claim’ is always executed. After that, ‘pay’ followed by ‘check claim’ may be executed repeatedly.

3 Model Visualisation

A key aspect of IvM is its visualisation of the discovered process model: on this model, all further enhancements, which will be described later on this section, will be visualised. In this section, we discuss our choice for this visualisation.

The IvM uses *process trees* behind the scenes to make sure the model is always free of anomalies such as deadlocks and unconnected parts. For more information about process trees, please refer to [4, Ch.2]. As process trees are a mathematical notation, IvM shows models in an intuitive formalism that closely resembles Petri nets, process trees and BPMN models. Figure 2 shows the constructs of these models. In such a model, each trace traverses edges from the source to the sink, thereby executing each activity on its path. Figure 3 shows an example, which corresponds to the process tree $\text{sequence}(\text{xor}(a, b), \text{concurrent}(c, d), \text{interleaved}(e, f))$. In case of concurrency, the path is “split” in multiple branches, e.g. in our example c and d are both executed, and these paths are merged again at a concurrency join. Inclusive choice and interleaving are similar, corresponding to their process tree semantics, i.e. inclusive choice (or) splits the path into one or more subsequent branches, while interleaving (interleaved) splits the paths but allows only one to be “active” at the same time.

4 Controls & Parameters

As described, IvM will perform the steps described in Section 6, and show intermediate results. It is not necessary to wait for IvM to complete these steps; users can change parameters any time, and IvM will automatically recompute the necessary steps. Figure 4 shows these parameters, and we will explain them in more detail in this section.

4.1 Activities Slider

The activities slider controls the fraction of activities that is included in the event log on which a discovery algorithm is applied. That is, before discovery, the event log is filtered. The position of the slider (between 0 and 1) determines how many of the activities remain in the filtered event log. For instance, the log $[\langle a, b, c\rangle, \langle a, b\rangle, \langle a\rangle]$, has the frequency table $[a^3, b^2, c]$, and if the activities slider would be set to 0.4, then all events corresponding to the activities that occur more than 0.4 times the occurrence of the most-occurring activity would be included. In out example, the filtered event log would be $[\langle a, b\rangle^2, \langle a\rangle]$, and to this filtered event log, the discovery algorithm is applied. Notice that this only affects the discovery, i.e. all other parts of the IvM including alignments and animation are not affected by this slider.

Putting this and the paths slider (described next) all the way up to 1.0 and setting the miner selector to IMf guarantees fitness. However, if the event log contains life-cycle transitions besides complete, deviations might be shown.

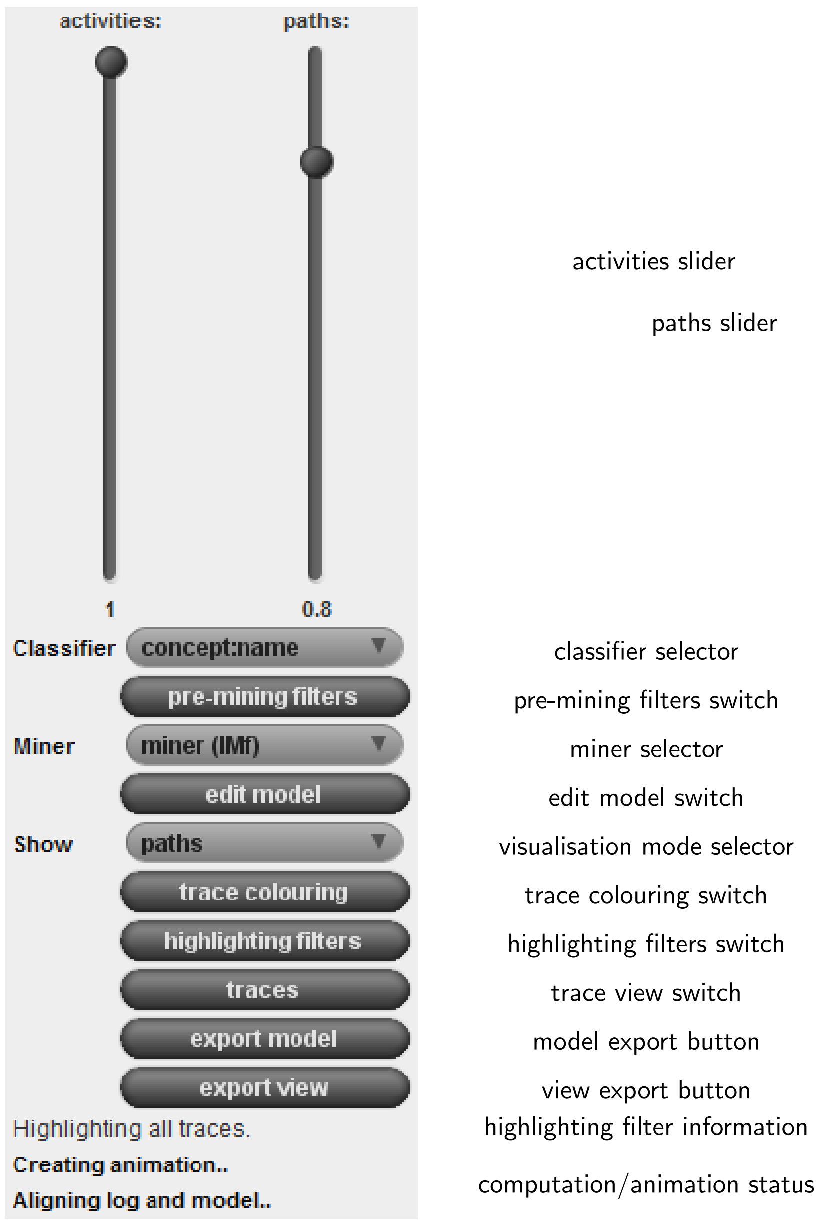
4.2 Paths Slider

The paths slider controls the amount of noise filtering applied: if set to 1, then no noise filtering is applied, while set at 0, maximum noise filtering is applied. Technically, the slider sets the input for the discovery algorithm to 1 - the value of the slider. The default is 0.8, which corresponds to $1 - 0.8 = 0.2$ noise filtering in IMf, IMflc and IMfa. Please refer to [4, Ch.6] for more information on the mining algorithms.

Putting both sliders all the way up to 1.0 and setting the miner selector to IMf guarantees fitness. However, the alignment of IvM always takes life-cycle information into account, thus deviations might still be present.

4.3 Classifier Selector

The classifier selector controls what determines the activities of events: events in XES-logs can have several data attributes [3], and this selector determines which one of these data attributes determines the types of activities. Any combination of event attributes can be chosen by using the checkboxes.



activities: 1

paths: 0.8

Classifier concept:name

Miner miner (IMf)

Show paths

Highlighting all traces.

Creating animation..

Aligning log and model..

Figure 4: Controls of IvM.

4.4 Pre-Mining Filters Switch

The pre-mining filters switch opens a panel to set pre-mining filters. A pre-mining filter does not alter the alignment, the performance measures or the animation, but filters the log that is used to discover a model. To activate a pre-mining filter, check its checkbox.

For instance, the pre-mining filter 'Trace filter' allows to discover a model using only the customers who spent more than €10,000.

4.5 Miner Selector

The miner selector allows to select which mining algorithm is to be used. Default is IMf, other included options are the life-cycle algorithm IMflc and the more-operators algorithm IMfa. We limit the choice to ease the users: these algorithms were shown to be the most applicable to real-life event logs in the evaluation of [4].

4.6 Edit Model Switch

The edit model switch opens a panel to manually edit the discovered model, as explained below. This allows users to correct the discovery algorithm if its result is not satisfactory, and to try the effect of a different, custom, model on the same event log. In this panel, the currently discovered process tree is displayed in a custom notation, and can be edited. While typing, the IvM redoes computations automatically. A screenshot is shown in Figure 5.

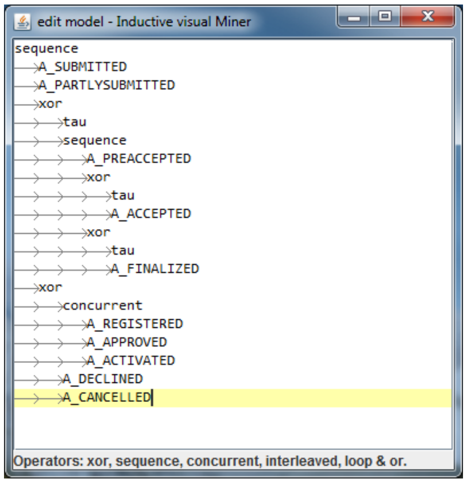
The notation to edit process trees in IvM is as follows: each process tree node should be on its own line. The white space preceding the node declaration matters, i.e. a child should be more indented than its parent. Reserved keywords are *xor*, *sequence*, *concurrent*, *interleaved*, *or*, *loop* and *tau*. Loops should be given in an unary (loop(a)), binary (loop(a, b)) or ternary (loop(a, b, c)) form, in which the c denotes the loop exit, i.e. loop(a, b, c) = sequence(loop(a, b), c). Any other text is interpreted as an activity name. In case a keyword is used as an activity name, it should be put in between double quotes (e.g. "sequence" is the activity called 'sequence').

In case the edited process tree contains a syntactical error, this will be shown at the bottom of the panel, and an approximate location of the error will be highlighted. The manual changes are overwritten if the automatic discovery is triggered, however, *ctrl z* reverts the edit model view to a previous state.

4.7 Visualisation Mode Selector

The visualisation mode selector allows user to choose between several information to be added to the model. There are four options:

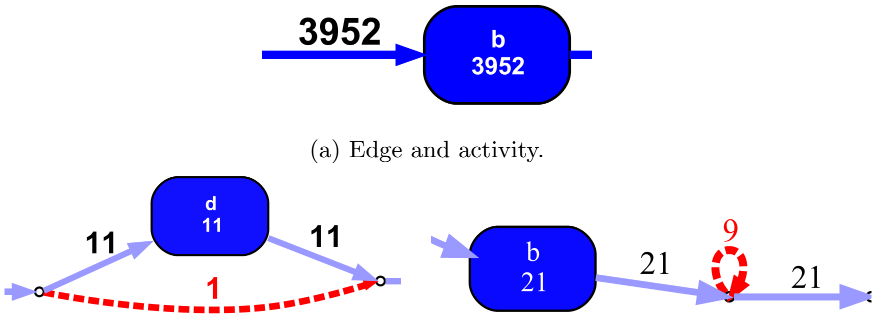
- **paths** This is the default mode, showing the model; the numbers on the activities and edges denote the total number of executions of each of them.



```
sequence
──>A_SUBMITTED
──>A_PARTLYSUBMITTED
──>xor
──>──>tau
──>──>sequence
──>──>──>A_PREACCEPTED
──>──>──>xor
──>──>──>──>tau
──>──>──>──>A_ACCEPTED
──>──>──>xor
──>──>──>──>tau
──>──>──>──>A_FINALIZED
──>xor
──>──>concurrent
──>──>──>A_REGISTERED
──>──>──>A_APPROVED
──>──>──>A_ACTIVATED
──>──>A_DECLINED
──>──>A_CANCELLED
```

Operators: xor, sequence, concurrent, interleaved, loop & or.

Figure 5: In the edit model panel, the model can be edited.



(a) Edge and activity.

(b) Model move.

(c) Log move.

Figure 6: IvM visualisation mode concepts.

Figure 6a shows an example: activity *b* was executed 3952 times, just as the incoming edge to the left of it.

- **paths and deviations** shows the model; the numbers in the activities denote the total number of executions of each of them. Moreover, red-dashed edges denote the results of alignments (which will be discussed in Section 5): Figure 6b shows a model move, indicating that activity *d* was skipped once in the event log, while the model said it should have been executed. Figure 6c shows a log move, indicating that 9 times in the event log, after the execution of activity *b*, an event happened in the event log while this should not happen according to the model. In Section 5, we will elaborate on deviation enhancements.

- **paths and queue lengths** shows the model, and denotes each activity with the queue length in front of it, i.e. the number of cases waiting for this activity to start. If the event log contains both events with enqueue and start life-cycle information, this queue length is accurate. Otherwise, it is estimated using the method described in [6]. This queue size is updated as the animation progresses.

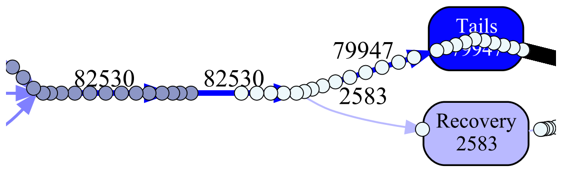
- **paths and sojourn times** shows the model, and denotes each activity with the average sojourn time for that activity. Sojourn times are computed using completion events. The sojourn times are not estimated, i.e. if not both necessary completion events are present and have timestamps, the activity instance is excluded from the average. Performance measures can also be inspected by putting the mouse cursor on an activity: a pop-up will show the performance measures and a histogram. Performance measures are automatically updated when any log filtering is applied.

- **paths and service times** shows the model, and denotes each activity with the average service time for that activity. Service times are computed using start and completion events. The service times are not estimated, i.e. if for an activity instance not both start and completion events are present and have timestamps, that activity instance is not considered in the average.

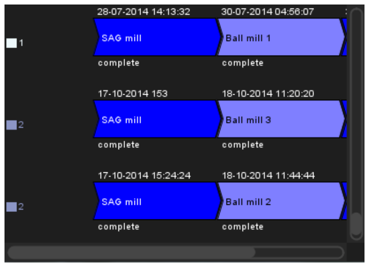
4.8 Trace Colouring Switch

The IvM can colour traces in the animation and the trace view. Using this colouring, different categories of traces can be easily distinguished. For instance, Figure 7 contains a screenshot of coloured traces in the animation and the trace view (which we will explain later). This event log represents an ore mining process, and the traces have been coloured with the hardness of the rock that is being processed.

The trace colouring switch opens a panel to set up the trace colouring. In this panel, a trace attribute can be chosen, as well as the derived properties 'duration' and 'number of events'. IvM supports up to 7 colours, and if the attribute is



(a) In the animation.



(b) In the trace view. The little blocks on the left denote the category of rock hardness (in the log, this was decoded with a number).

Figure 7: Coloured traces in IvM.

numeric, the domain of the numbers is split into 7 even parts automatically. Date and time attributes are handled similarly. If the attribute is literal and there are more than 7 different values, the colouring will remain disabled.

To enable quick enabling and disabling of the trace colouring, a checkbox has been added to the left side of the panel, which should be checked to enable trace colouring.

4.9 Highlighting Filters Switch

The highlighting filters switch opens a panel to set highlighting filters. A highlighting filter does not alter the model or the alignment, but filters the log that is shown in the animation and the information projected on the activities and edges of the model. If a highlighting filter is enabled, the highlighting filter information will show this.

A highlighting filter can also be applied to an activity in the model: by clicking on an activity (i.e. selecting it), the event log is filtered to only contain traces for which this activity was executed in accordance with the model, i.e. log-moves and model-moves are excluded. Hold the control-key to select multiple activities; edges can be selected as well. The highlighting filter information will textually show these click-highlighting filters as well.

To enable quick enabling and disabling of highlighting filters, a checkbox has been added to the left side of each filter, which should be checked to enable

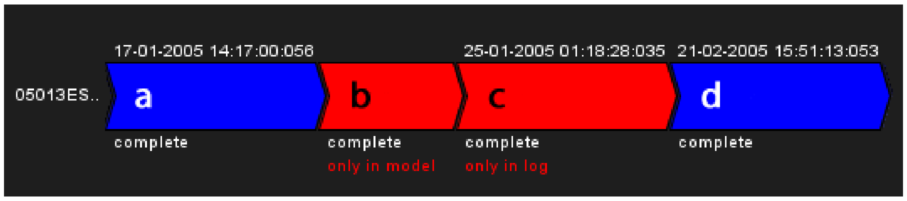


Figure 8: Trace view.

filtering.

4.10 Trace View Switch

To allow inspection of the traces, and to provide insight to the deviations between model and log on the log-level, IvM offers a trace view. The trace view switch enables or disables the trace view.

Figure 8 shows a trace in this trace view: the name of the trace (i.e. the *concept:name* extension) is displayed to the left of the events, which are the coloured wedges to the right. Above the wedges, time stamps are displayed in day-month-year hour:minute:second:millisecond. The wedge itself shows the activity (depending on the classifier selector) of the event. Below the wedge, the first line shows the life-cycle transition information (if that is not present, it shows *complete*). Second, below the wedge the alignment information is shown: in Figure 8, the first event is a synchronous event, the second is a model move ("only in model") and the third one is a log move ("only in log").

4.11 Model Export Button

The model export button allows the current model to be exported as a Petri net or process tree to the workbench of ProM.

4.12 View Export Button

The view export button exports the current image to an image file. Moreover, the animation can be exported (rendered) as a movie, and some statistics about the activities can be exported as a comma-separated-value (csv) file.

4.13 Changing the View

The model can be moved by dragging it, or by using the arrow keys. Zooming in and out can be done with a scroll wheel, or with the key combination *ctrl* = or *ctrl* -. *Ctrl 0* (zero) resets the model to its initial position.

Once zoomed in, a navigation image will appear in the upper left corner. A click on this navigation image will move the model to that position, and scrolling while the mouse pointer is in the navigation image will zoom the navigation image.

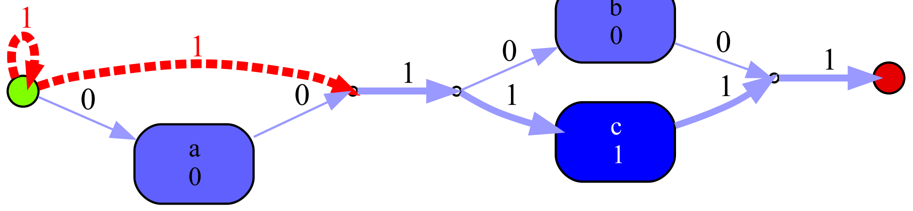


Figure 9: Visualisation of moves.

The graph direction, i.e. the position of the green and red start and end places, can be changed by pressing *ctrl d*. The distance between activities and edges can be altered using the key combinations *ctrl q* and *ctrl w*.

5 Interpreting Deviations

Process discovery algorithms might leave behaviour that was recorded in the event log out of the discovered model, and might include behaviour in the model that is not recorded in the event log, as discovery algorithms try to represent the behaviour of event logs into a certain representational bias. Therefore, the discovered model should be evaluated before reliable conclusions can be drawn from such models. Conformance checking techniques enable the evaluation of models on three levels: summarative measures, projections on models and projections on event logs. Furthermore, the process model can be entered by hand (using the "edit model" function) to circumvent process discovery and assess an idealised or normative model. (See also the plug-in "Visualise deviations on process tree".)

In this section, we show the output and intermediate computation results of alignments ([1]) are projected by IvM on process models and event logs, and what conclusions can be reliably drawn from these projections.

A key property of alignments is that they provide a path through the model that is most similar to the trace in the log. For instance, consider the following alignment of the trace $t = \langle b, c \rangle$ and the model $M =$ sequence:

$$\begin{tikzcd}[arrows=-] & \text{sequence} \arrow[dl] \arrow[dr] & & \\ a & & \text{xor} \arrow[dl] \arrow[dr] & \\ & b & & c \end{tikzcd}$$

trace	b	-	c
model	-	a	c

This alignment provides several pieces of useful information about deviations:

- According to the model, a should have been executed, however no such event was found in the trace. This model move can be considered as a 'skip' of a. In Figure 9, this concept is visualised on the model as a red-dashed edge that bypasses a.

- According to the event log, b should have been executed, however the model did not support this. This log move denotes that the event is considered to be superfluous. In Figure 9, we show this concept being visualised on a model: the red dashed self-loop indicates this log move.

Notice that these concepts are applicable to a variety of process model notations, such as BPMN, Petri nets and YAWL.

Pitfalls. Despite the intuitiveness of the visualisations, these concepts should be interpreted with care, as they might convey more information than is actually available. For instance, the alignment shown in our example is not the only 'optimal' alignment. There may be other alignments with the same number of log and model moves. For instance, the following alignment has the same number of deviations:

trace	-	b	c
model	a	b	-

In this alignment, c is a log move instead of b.

Another example is the model sequence(a, b) and the trace $\langle a, c\rangle$. There two optimal alignments:

trace	a	-	c
model	a	b	-

trace	a	c	-
model	a	-	b

Notice the difference in order between log move c and model move b: this order is arbitrary. Nevertheless, in the visualisation, a choice had to be made to position the log move before or after activity b.

The current alignment implementations traverse the state space defined by log and model, and deterministically choose one option in case there are multiple optimal possibilities. Which possibility is chosen is not always easily determinable by the user, as it depends on internal ordering and sorting. For more details, we refer to [1].

Furthermore, notice that the different alignments discussed here are all 'optimal'. However, there is no guarantee that an optimal alignment reflects reality, i.e. a non-optimal alignment (i.e. with more deviations than an optimal alignment) could also explain the deviations between event log and model, and might explain this better if domain knowledge is taken into account. Thus, one should be careful interpreting alignments and the deviations shown in IvM.

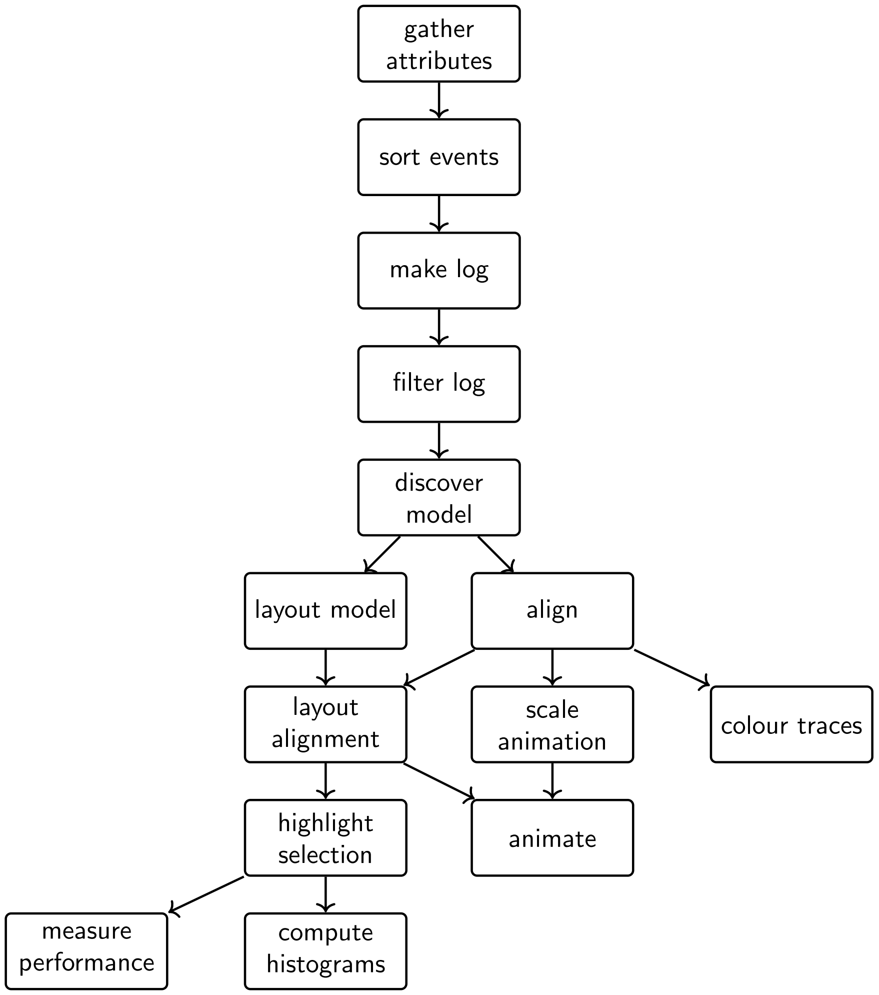


Figure 10: The architecture of IvM. The arrows denote constraints: a task is started as soon as its preceding tasks are finished.

6 Steps & Architecture

The IvM performs several steps automatically. The computations steps can be interrupted by the user at any time, and IvM will automatically redo steps on user input. Figure 10 shows these steps and their dependencies. The main steps are:

- Sort events.
  Some event logs contain traces in which the timestamps are out of order. For instance, in the trace $\langle a^{14:00}, b^{13:00} \rangle$ a happened first according to the order of the events in the trace, but b occurs first according to the timestamps of the events. Such anomalies make animation and performance measures unreliable, so IvM offers the user the choice to either sort the events (in our example, IvM would continue as if $\langle b^{13:00}, a^{14:00} \rangle$ was given) or disable the animation and performance measures.

- Filter log.
  Let L be the event log. This step will remove events of which the activities do not occur enough. See the activities slider and the pre-mining filters in the Controls & Parameters settings for more information. This step is also available (with even more fine-grained options) as a separate plug-in of ProM ("Filter events").

- Discover a process model from log L_1,
  which is done using either the algorithm IMf, IMflc or IMfa (depending on the miner selector).

- Align the model and the log L.
  The alignment is based on work described in [2]. Before aligning, the discovered model is expanded (see [4, Ch.5]), i.e. each activity a is transformed into a nested process tree sequence($a_{\mathrm{E}}, a_{\mathrm{S}}, a_{\mathrm{C}}$). This expansion is used at all times, i.e. the alignment *always* takes enqueue, start and completion events into account.

- Visualise the model and the alignment.

- Animate the alignment.

- Compute performance measures and visualise them.

References

[1] Adriansyah, A.: Aligning Observed and Modeled Behavior. Ph.D. thesis, Eindhoven University of Technology (2014) 12, 13

[2] Buijs, J.C.A.M.: Flexible Evolutionary Algorithms for Mining Structured Process Models. Ph.D. thesis, Eindhoven University of Technology (2014) 15

[3] Günther, C., Verbeek, H.: XES v2.0 (2014), `http://www.xes-standard.org/` 5

[4] Leemans, S.J.J.: Robust process mining with guarantees. Ph.D. thesis, Eindhoven University of Technology (2017) 2, 4, 5, 7, 15

[5] Leemans, S.J.J., Fahland, D., van der Aalst, W.M.P.: Process and deviation exploration with Inductive visual Miner. In: Limonad, L., Weber, B. (eds.) Proceedings of the BPM Demo Sessions 2014 Co-located with the 12th International Conference on Business Process Management (BPM 2014), Eindhoven, The Netherlands, September 10, 2014. CEUR Workshop Proceedings, vol. 1295, p. 46. CEUR-WS.org (2014), `http://ceur-ws.org/Vol-1295/paper19.pdf` 2

[6] Senderovich, A., Leemans, S.J.J., Harel, S., Gal, A., Mandelbaum, A., van der Aalst, W.M.P.: Discovering queues from event logs with varying levels of information. In: Reichert, M., Reijers, H.A. (eds.) Business Process Management Workshops - BPM 2015, 13th International Workshops, Innsbruck, Austria, August 31 - September 3, 2015, Revised Papers. Lecture Notes in Business Information Processing, vol. 256, pp. 154–166. Springer (2015), `http://dx.doi.org/10.1007/978-3-319-42887-1_13` 9