Process Mining project

MSc Computer Science, Process Mining course

Yuri Paoloni 115554

A.Y. 2020/21

Abstract. Application of Process Mining techniques and algorithms on artificial logs generated from the execution of a process model. The goal of the project is to show the different output and behaviors depicted by the Process Mining techniques and algorithms when applied on artificial logs with or without noise on the control flow.

1 Introduction

The following elaborate intention is to present the application of Process Mining techniques and algorithms on artificial logs generated from the execution of a process model. A detailed description of the different results and behaviors depicted by the Process Mining techniques and algorithms when applied on artificial logs with or without noise on the control flow. The artificial logs under analysis are three: one without noise, one with a low level of noise on the control flow, and one with a high level of noise on the control flow. These events logs have been randomly generated from a sufficiently complex process model generated randomly using PLG [1], an application capable to generate random business processes and events logs, starting from some general "complexity parameters". The logs have been produced in XES format [6] using, again, PLG through the tuning of parameters for the noise on the control flow. Logs with noise have been used to simulate a sort of "real-world" behavior where not everything goes as expected, unlike during the process model design phase. The produced events log have been analyzed using PM4Py [4], a Python library for Process Mining. In the first place, the logs have been imported along with the printing of some statistics to point out their discrepancies. Afterward, the Process Discovery algorithms and Conformance Checking techniques were applied along with some considerations on their behaviors and results.

2 Tools

The tools employed in the project execution were PLG and PM4Py.

2.1 PLG

PLG[1] stands for Process Log Generator and is an application capable to generate random business processes, starting from some general "complexity parameters". PLG is also able to "execute" a given process model to generate a process log. The main features are:

- Random generation and simulation of complex process models;
- Random generation and simulation of complex data objects;
- Detailed tweaking of activities time features;
- Random evolution of a process model, to generate a slightly different version of it (to simulate concept drifts);
- Deep control on the generated simulation noise;
- Event stream generation, and, while the stream is running, straightaway switch of the underlying simulated model, to simulate concept drifts;
- Process model import from PLG2 or BPMN XML files (designed with Signavio);
- Models export as BPMN 2.0 XML, BPMN/Petri net Graphviz Dot files, PNML and PLG2.

PLG has been used to generate the process model and the related artificial logs.

2.2 PM4Py

PM4Py[4] is a Python library that supports (state-of-the-art) Process Mining algorithms in Python. It is completely open-source and intended to be used in both academia and industry projects. PM4Py is a product of the Fraunhofer Institute for Applied Information Technology. The main features are:

- Handling event data: importing and exporting event logs stored in various formats;
- Filtering event data: specific methods to filter an event log based on a timeframe, case performance, start/end activities, variants, and attribute values;
- Process discovery: provides alpha miner, inductive miner, heuristic miner, directly-follows graphs, and others...
- Petri Net management;
- Conformance checking: provides token-based replay and alignments
- Statistics: it is possible to calculate different statistics on top of classic event logs and data frames
- Log-Model evaluation: it is possible to compare the behavior contained in the log and the behavior contained in the model, to see if and how they match;
- Simulation: it offers different simulation algorithms, that starting from a model, can produce an output that follows the model and the different rules that have been provided by the user:

PM4Py does not provide a user interface but only Python API. It is a low-level tool compared with others in the market and this is exactly why I decided to adopt it since working at a deeper level can improve the understanding of the activities within the process.

The library has been used to import the events log and to apply the Process Discovery algorithms and Conformance Checking techniques on them.

3 Execution

The Python scripts, the models, and the events logs are all accessible in the project's GitHub repository [7]. The execution of the project was split into three phases.

3.1 Getting the data

First of all, the process model under analysis, showed in figure 1 has been generated. The model is, compared to real-world models, really simple but within the scope of the project's end, it is adequately complex. Then, the model was submitted to PLG for the creation of artificial logs. The tool permits the generation of artificial events logs with the possibility of specifying different values for the level of noise on the control flow such as trace missing head/tail probability, perturbed event order probability, trace missing episode probability, etc...

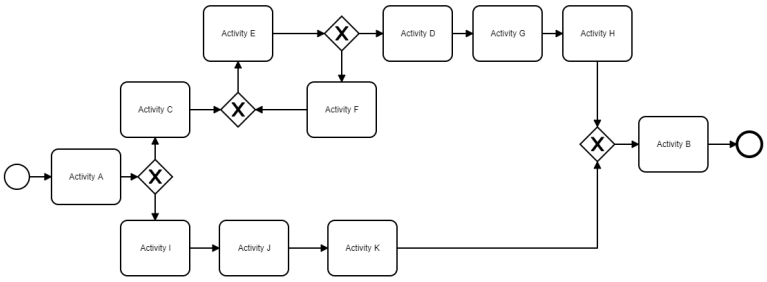

Figure 1: Process model used

From the model, 3 logs of 5.000 traces were generated, one without noise (log A), one with a low level of noise on the control flow (log B), and the last one with a high level of noise on the control flow (log C). The log B and log C have, respectively, 5% and 70% probability on these parameters:

- trace missing head;
- trace missing tail;
- trace missing episode;
- perturbed event order.

Using PM4Py, a Python script was defined along with some "exploratory operations" to print out some values and highlight the unlikeness of the logs. The following are the statistics obtained from the 3 logs during the import:

1. Log A statistics:
    - Number of events: 5000
    - Event level attributes: ['concept:name', 'time:timestamp']

- Attribute values and their counter: 'Activity A': 5000, 'Activity I': 2444, 'Activity J': 2444, 'Activity K': 2444, 'Activity B': 5000, 'Activity C': 2556, 'Activity E': 4915, 'Activity F': 2359, 'Activity D': 2556, 'Activity G': 2556, 'Activity H': 2556
- Number of start activities: 1
- Start activities: 'Activity A': 5000
- Number of end activities: 1
- End activities: 'Activity B': 5000
- Number of variants: 6

2. Log B statistics:
   - Number of events: 5000
   - Event level attributes: ['concept:name', 'time:timestamp']
   - Attribute values and their counter: 'Activity A': 4979, 'Activity C': 2452, 'Activity E': 4784, 'Activity F': 2327, 'Activity D': 2456, 'Activity G': 2454, 'Activity H': 2451, 'Activity B': 4976, 'Activity I': 2532, 'Activity J': 2532, 'Activity K': 2530
   - Number of start activities: 8
   - Start activities: 'Activity A': 4974, 'Activity C': 7, 'Activity E': 8, 'Activity K': 1, 'Activity I': 5, 'Activity B': 1, 'Activity J': 2, 'Activity G': 2
   - Number of end activities: 8
   - End activities: 'Activity B': 4972, 'Activity J': 9, 'Activity G': 8, 'Activity K': 4, 'Activity A': 1, 'Activity H': 4, 'Activity E': 1, 'Activity I': 1
   - Number of variants: 56

3. Log C statistics:
   - Number of events: 5000
   - Event level attributes: ['concept:name', 'time:timestamp']
   - Attribute values and their counter: 'Activity A': 4635, 'Activity I': 2407, 'Activity J': 2485, 'Activity K': 2401, 'Activity B': 4628, 'Activity C': 2351, 'Activity E': 4674, 'Activity D': 2427, 'Activity G': 2416, 'Activity H': 2324, 'Activity F': 2261
   - Number of start activities: 11
   - Start activities: 'Activity A': 4583, 'Activity J': 100, 'Activity E': 85, 'Activity C': 94, 'Activity I': 117, 'Activity K': 7, 'Activity B': 6, 'Activity G': 2, 'Activity D': 1, 'Activity H': 3, 'Activity F': 2
   - Number of end activities: 11
   - End activities: 'Activity B': 4594, 'Activity K': 105, 'Activity H': 79, 'Activity G': 99, 'Activity J': 101, 'Activity D': 2, 'Activity I': 7, 'Activity A': 6, 'Activity C': 1, 'Activity E': 4, 'Activity F': 2
   - Number of variants: 147

In the first log A, we have the statistics we would have expected since it has no noise. The 6 variants are due to the 4 exclusive gateways inside the model. Regarding the other two logs (B and C), we can see that the starting and ending activities (activity A and activity B) are the most executed ones in both the logs, and in general the number of occurrences of the other activities is almost similar in both of them. What varies are the number of diverse

starting and ending activities with their occurrences along with the number of variants (56 in log B and 147 in log C). This proves how much of a discrepancy there is between the logs in terms of different executions of the model.

3.2 Process Discovery

Process Discovery algorithms want to find a suitable process model that describes the order of events/activities that are executed during a process execution based on an events log. The process discovery algorithms applied are the Alpha miner and the Heuristic miner, both of them were executed using the Python API provided by PM4Py.

3.2.1 Alpha miner

The Alpha miner is one of the most known Process Discovery algorithms. It is also the first formalized algorithm and, for this reason, it has some problems. It cannot handle loops of lengths one and two, it cannot discover an invisible and duplicated task, and it is weak against noise. The weaknesses described are easily shown in the figure 2 where even without noise, the Alpha miner is not able to recognize the correct model by ignoring the activity F that in the real model is inside a short loop. The output of the Alpha miner is a Petri Net.

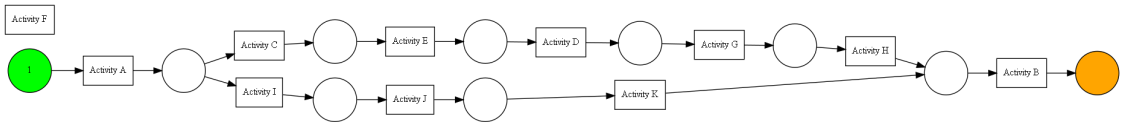

Figure 2: Alpha miner output's Petri Net on the log A

In log B, the algorithm doesn't recognize again the loop leaving apart the activity F. It also returns a different model due to the noise that has produced 56 variants as stated before.

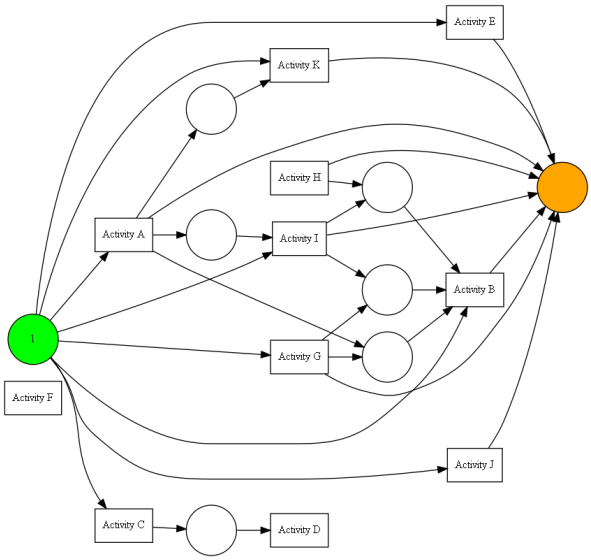

Figure 3: Alpha miner output's Petri Net on the log B

In log C, the algorithm includes the activity F inside the graph but only because of the noise that has generated traces without following the loop of the root process model. The traces generated in log C are so perturbed that no paths from the original model can be recognized.

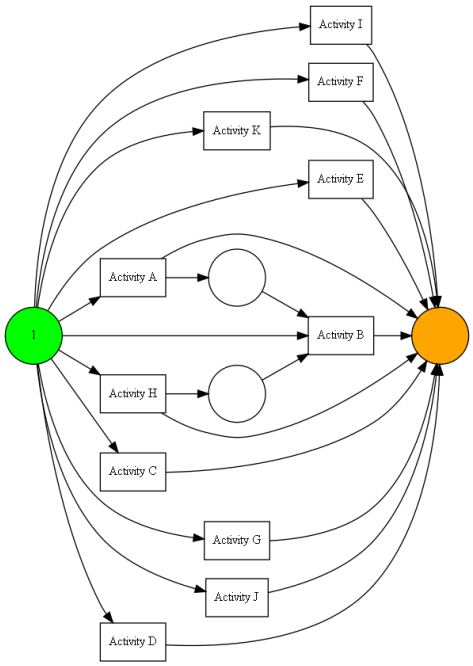

Figure 4: Alpha miner output's Petri Net on the log C

3.2.2 Heuristic miner

Heuristics Miner is an improvement of the Alpha algorithm. It provides a way to handle noise, takes frequency into account, detects short loops but does not guarantee a sound model. The output of the Heuristics Miner is a Heuristics Net that, if needed, can be easily converted into a Petri Net. Thanks to the Heuristic net, it is possible to see the number of times each activity and path is executed. The execution of the Heuristic miner on log A gives back the Heuristic net displayed in figure 5:

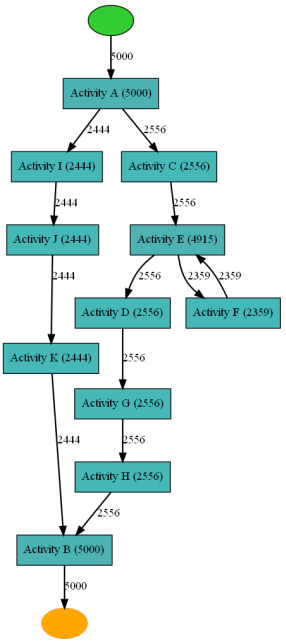

Figure 5: Heuristic Miner output's Heuristic Net on the log A

The registered values in the arcs and on the activities of the Heuristic Net are the same described inside section 3.1 in the statistics list under the heading "Attribute values and their counter". As said before, the Heuristic miner takes frequency into account.

The application of the Heuristic Miner on the logs B and C returns the Heuristic nets shown, respectively, in the figures, 6 and 7

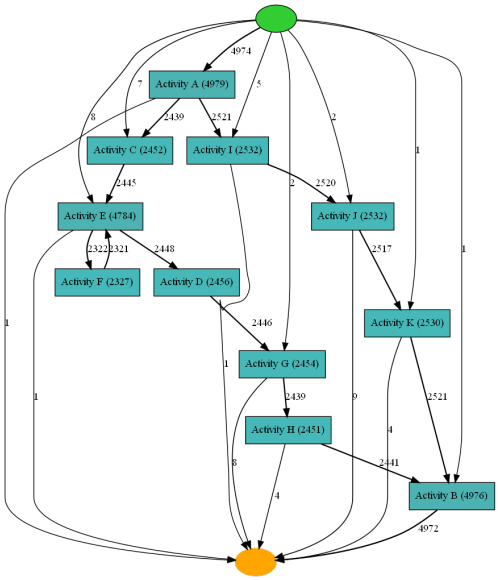

Figure 6: Heuristic Miner output's Heuristic Net on the log B

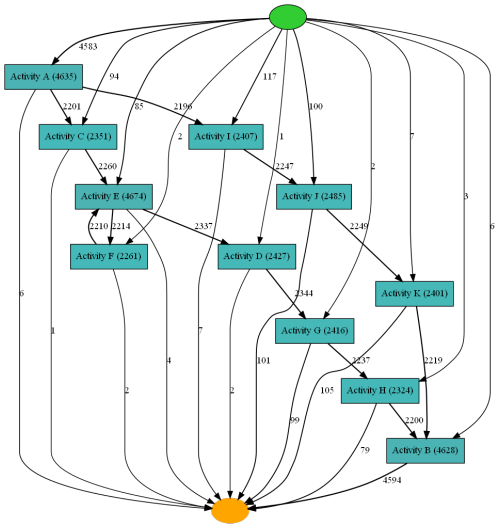

Figure 7: Heuristic Miner output's Heuristic Net on the log C

In all the three returned Heuristic nets, the loop between activities E and F is recognized. This confirms the better performance that the Heuristic miner can offer compared to the Alpha miner. In addition to this, the algorithm provides also a way to handle noise by applying threshold values on activity and arcs to consider only the strong relations excluding the weaker. In PM4Py, this can be done by tuning some input parameters on the algorithm as stated in the documentation [3]. The execution, with the following parameters, has been performed only on logs B and C since log A is cleaned from noise:

- minimum number of occurrences of an edge to be considered: 50;
- minimum number of occurrences of an activity to be considered: 50;
- cleaning threshold of the graph to remove weaker edges: 0.3;
- dependency threshold of the Heuristics Miner: 0.8

The graphs in figures 8 and 9 shows how it is possible to handle noise with the Heuristic miner. In the graph related to log B, all the deviations have been removed and a graph similar to figure 5 have been obtained. In the one related to log C, the deviations have been reduced but not that much to be equal to the figure 5. This happens because higher levels of noise demand higher values of thresholds as well.

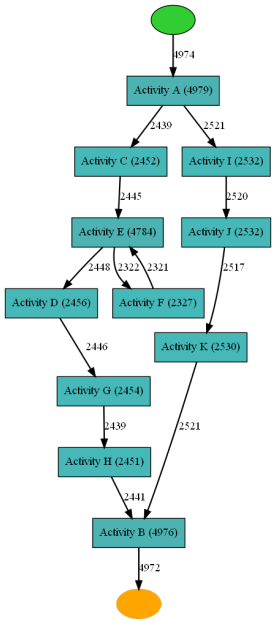

Figure 8: Threshold execution of Heuristic Miner on the log B

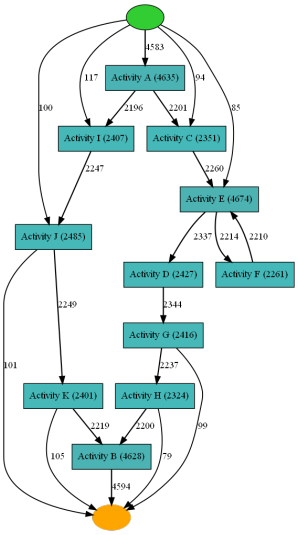

Figure 9: Threshold execution of Heuristic Miner on the log C

3.3 Conformance Checking

Conformance checking is a technique to compare a process model with its events log. The goal is to check if the event log conforms to the model, and, vice versa. In PM4Py, two fundamental techniques are implemented: token-based replay and alignments.

3.3.1 Token-based replay

Token-based replay matches a trace and a Petri net model, starting from the initial place, to discover which transitions are executed and in which places we have remaining or missing tokens for the given process instance. In PM4Py there is an implementation of a token replayer [5] that can go across hidden transitions (calculating shortest paths between places) and can be used with any Petri net model with unique visible transitions and hidden transitions. The execution can be customized with parameters to handle the walk-through of the algorithm.

The following is an example of the output for execution on the Petri Net produced with the Heuristic Miner on log C.

```
{'activated_transitions': [Activity A, Activity I, Activity J,
                           Activity B],
'consumed_tokens': 7,
'enabled_transitions_in_marking': {Activity K},
'missing_tokens': 1,
'produced_tokens': 7,
'reached_marking': ['intplace_Activity J:1', 'sink0:1'],
'remaining_tokens': 1,
'trace_fitness': 0.8571428571428572,
'trace_is_fit': False,
'transitions_with_problems': [Activity B]}
```

PM4Py gives also the possibility to run diagnostic operations to obtain detailed information about transitions that did not execute correctly, or activities that are in the log and not in the model.

3.3.2 Alignments

Alignment-based replay aims to find one of the best alignment between the trace and the model. For each trace, the output of an alignment is a list of couples where the first element is an event (of the trace) or $\gg$ and the second element is a transition (of the model) or $\gg$. In PM4Py there is an implementation of Alignment-based replay that analyzes a Petri Net and can be customized in different ways providing different classifier and cost functions [2].

The following is an example of the output for execution on the Petri Net produced with the Heuristic Miner on log A.

```
{'alignment': [('Activity A', 'Activity A'),
               ('Activity I', 'Activity I'),
               ('Activity J', 'Activity J'),
               ('Activity K', 'Activity K'),
               ('Activity B', 'Activity B')],
'cost': 0,
'fitness': 1.0,
'lp_solved': 1,
'queued_states': 16,
'traversed_arcs': 21,
```

```
'visited_states': 5},
```

4 Concluding remarks

The project can be intended as a playground to understand the basic Process Mining algorithms and techniques in practice. It exposes the limits of the Alpha Miner on dealing with noise and loops, and the improvements the Heuristic Miner brings to address these concerns. Moreover, the main Conformance Checking techniques are introduced to show their practical execution.

PM4PY is a really powerful tool for Process Mining that provides a lot of basic and advanced features. In the project, only a small part of them was described and used in a substantially basic way. PM4Py allows for complex operations such as Petri Net and Process Tree management, feature selection and prediction for Decision Mining, evaluation of quality metrics, simulations, Social Network Analysis, Streaming Process Mining, and more...

All the project-related data such as logs, models, and Python scripts can be found in the GitHub repository [7].

References

[1] Andrea Burattin. Plg2: Multiperspective process randomization with online and offline simulations. *Online Proceedings of the BPM Demo Track 2016*, 2016.

[2] The Fraunhofer Institute for Applied Information Technology. Alignments in pm4py. https://pm4py.fit.fraunhofer.de/documentation#item-5-2.

[3] The Fraunhofer Institute for Applied Information Technology. Heuristic miner in pm4py. https://pm4py.fit.fraunhofer.de/documentation#item-3-3.

[4] The Fraunhofer Institute for Applied Information Technology. Pm4py. https://pm4py.fit.fraunhofer.de/.

[5] The Fraunhofer Institute for Applied Information Technology. Token-based replay in pm4py. https://pm4py.fit.fraunhofer.de/documentation#item-5-1.

[6] IEEE. Xes standard. https://xes-standard.org/.

[7] Yuri Paoloni. Project github repository. https://github.com/yuripaoloni/pm-project.