University of Camerino

School of Science and Technology

Master Degree in Computer Science (Class LM-18)

PROCESS MINING PROJECT

A comparative analysis of Process Discovery Algorithms with PM4Py

Student

Sofia Scattolini

Matricola 130791

A.Y. 2024/2025

Abstract

This project explores the application of Process Mining techniques through a comparative analysis of two Process Discovery algorithms: Alpha Miner and Heuristic Miner, using the PM4Py Python library.

The goal is to evaluate the behavior, strengths and limitations of each algorithm to discover process models from event logs. The analysis is based on the "Road Traffic Fine Management Process" event log, a real-life dataset that describes the management of traffic fines.

Both algorithms were applied to the same event log, and the resulting models were assessed using conformance checking metrics (specifically, fitness and precision) to evaluate how well each model fits with the real event data.

Contents

List of Figures

Listings

List of Tables

1. Introduction

1.1 Project Objective

The objective of this project is to provide a demonstration and comparison of two process discovery algorithms (**Alpha Miner** and **Heuristic Miner**) implemented within the **PM4Py** process mining framework. Process discovery is a fundamental task in process mining that aims to extract a process model from an event log that represents the actual execution of business processes.

The aim of this work is to:

- Apply the Alpha Miner and Heuristic Miner algorithms to the same event log using the PM4Py tool;
- Analyze the differences in the process models generated by the two algorithms;
- Evaluate and compare the resulting models using conformance checking techniques, with a focus on `fitness` and `precision` metrics.

1.2 Document Structure

The structure of this report is organized as follows:

- **Section 2: Event Log Description** → This section presents an overview of the dataset (event log) used for the project, including its size, structure and relevant characteristics;
- **Section 3: Execution and results** → This section provides a theoretical overview of the Alpha Miner and Heuristic Miner algorithms, describes the PM4Py framework, explains how each algorithm was applied to the event log, and presents the resulting models along with fitness and precision metrics;
- **Section 4: Conclusions** → This final section provides concluding remarks.

2. Event Log Description

This chapter provides an in-depth description of the event log dataset used in this project. The chosen dataset is the *Road Traffic Fine Management Process*, a publicly accessible event log published by the 4TU Centre for Research Data [1]. This dataset reflects a real-life business process related to the management and resolution of traffic fines.

2.1 Log Structure and Attributes

The dataset is provided in the `.xes` (eXtensible Event Stream) format [2], which is the standard format for storing event logs in Process Mining. The log contains approximately **150,370** traces (cases), and over **561,470** individual events in total.

Each trace represents a unique process instance (i.e., a fine case), and each event corresponds to the execution of a specific activity. In total, the log includes **11** unique activities.

Every event contains several attributes that provide contextual and behavioral information:

- `concept:name` → name of the executed activity;
- `time:timestamp` → when the event occurred;
- `org:resource` → resource or performer of the activity;
- `lifecycle:transition` → status of the event (e.g., `complete`);
- Additional domain-specific attributes such as `amount`, `article`, `points`...

2.2 Example Trace from the XES Log

To better understand the structure of the log, Listing 2.1 shows a representative example of a trace. The trace includes two events: the creation of a fine and its subsequent payment. Each event specifies the activity performed, the timestamp, the responsible resource, and relevant process metadata.

Listing 2.1: Snippet from XES file

```
<trace>
  <string key="concept:name" value="A10005"/>
  <event>
    <float key="amount" value="36.0"/>
    <string key="org:resource" value="537"/>
    <string key="dismissal" value="NIL"/>
    <string key="concept:name" value="Create␣Fine"/>
    <string key="vehicleClass" value="A"/>
    <float key="totalPaymentAmount" value="0.0"/>
    <string key="lifecycle:transition" value="complete"/>
    <date key="time:timestamp" value="2007-03-20T00:00:00+01:00"/>
    <int key="article" value="157"/>
    <int key="points" value="0"/>
  </event>
  <event>
    <float key="totalPaymentAmount" value="36.0"/>
    <string key="concept:name" value="Payment"/>
    <string key="lifecycle:transition" value="complete"/>
    <float key="paymentAmount" value="36.0"/>
    <date key="time:timestamp" value="2007-03-21T00:00:00+01:00"/>
  </event>
</trace>
```

3. Execution and Results

This section describes the theoretical background of the process discovery algorithms used in the project, namely **Alpha Miner** and **Heuristic Miner**, and presents the tool used to perform the analysis: the **PM4Py** Python library. Finally, it details the experiments conducted, including how each algorithm was applied and how the results were evaluated through fitness and precision metrics.

3.1 PM4Py Tool

PM4Py [3] is an open source Python library that provides a set of tools for Process Mining tasks.

PM4Py supports the following core functionalities:

1. **Event Log Management**: PM4Py supports importing and exporting event logs in different formats such as `.xes`, `.csv`, and `.json`. It also includes utilities for filtering event logs based on attributes, time ranges, case lengths, start/end activities, and more.
2. **Process Discovery**: The library implements several process discovery algorithms, including:
   - Alpha Miner
   - Heuristic Miner
   - Inductive Miner
   - Directly-Follows Graphs
   - ...
3. **Conformance Checking**: PM4Py provides techniques for assessing the alignment between an event log and a process model. It includes:
   - Token-based replay
   - Alignment-based conformance checking
   - Evaluation metrics such as fitness, precision, generalization, and simplicity
4. **Visualization**: PM4Py allows for the visualization of discovered process models and diagnostics related to conformance checking.

It is important to note that PM4Py does not provide a graphical user interface; it is designed to be used as a low-level library using Python scripts.

In this project, PM4Py was used to:

- Apply *Alpha Miner* and *Heuristic Miner* to discover process models;
- Perform conformance checking using *token-based replay* technique;
- Evaluate the discovered models using *fitness* and *precision* metrics.

3.2 Process Discovery

Process discovery is a core functionality in process mining that focuses on creating a process model based on events recorded in a log. Identifies the typical order in which activities are performed during the execution of the process. In this work, the Alpha Miner and Heuristic Miner algorithms were applied to discover the process model from the event log. These algorithms were implemented using the Python API provided by the PM4Py framework.

3.2.1 Alpha Miner

The **Alpha Miner** is one of the earliest process discovery algorithms. Detects control-flow patterns based on the direct succession relation in the event logs. Although it is simple and provides a clear Petri Net model, it has some limitations:

1. Cannot handle short loops (length 1 or 2);
2. Sensitive to noise;
3. Cannot detect parallelism or invisible transitions.

The Alpha Miner was applied to the event log using PM4Py. The resulting model had low complexity but was not able to correctly represent some key parts of the real process, especially loops and concurrent activities.

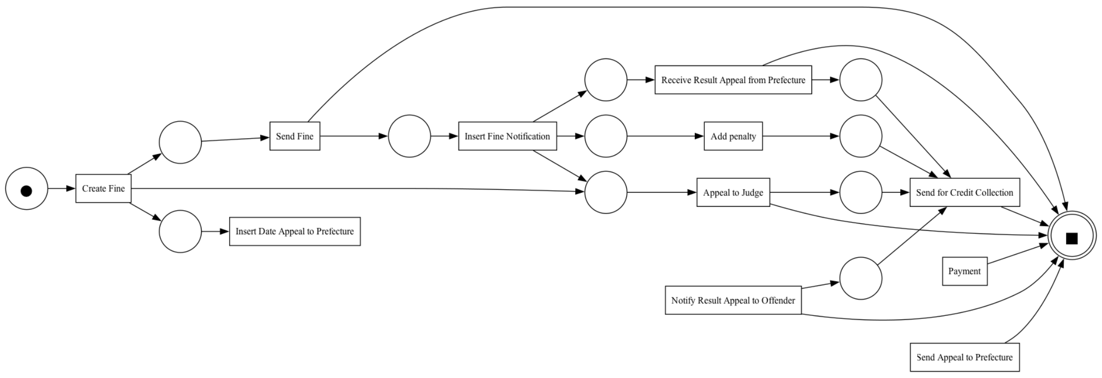
Create Fine
Send Fine
Insert Date Appeal to Prefecture
Insert Fine Notification
Receive Result Appeal from Prefecture
Add penalty
Appeal to Judge
Send for Credit Collection
Payment
Notify Result Appeal to Offender
Send Appeal to Prefecture

Figure 3.1: Process model discovered using Alpha Miner

3.2.2 Heuristic Miner

The **Heuristic Miner** is a more advanced algorithm that uses frequency-based heuristics to handle noise and detect short loops and parallelism. The output is a **Heuristics Net**, a specific type of process model that highlights the frequency and strength of the relationships between activities.

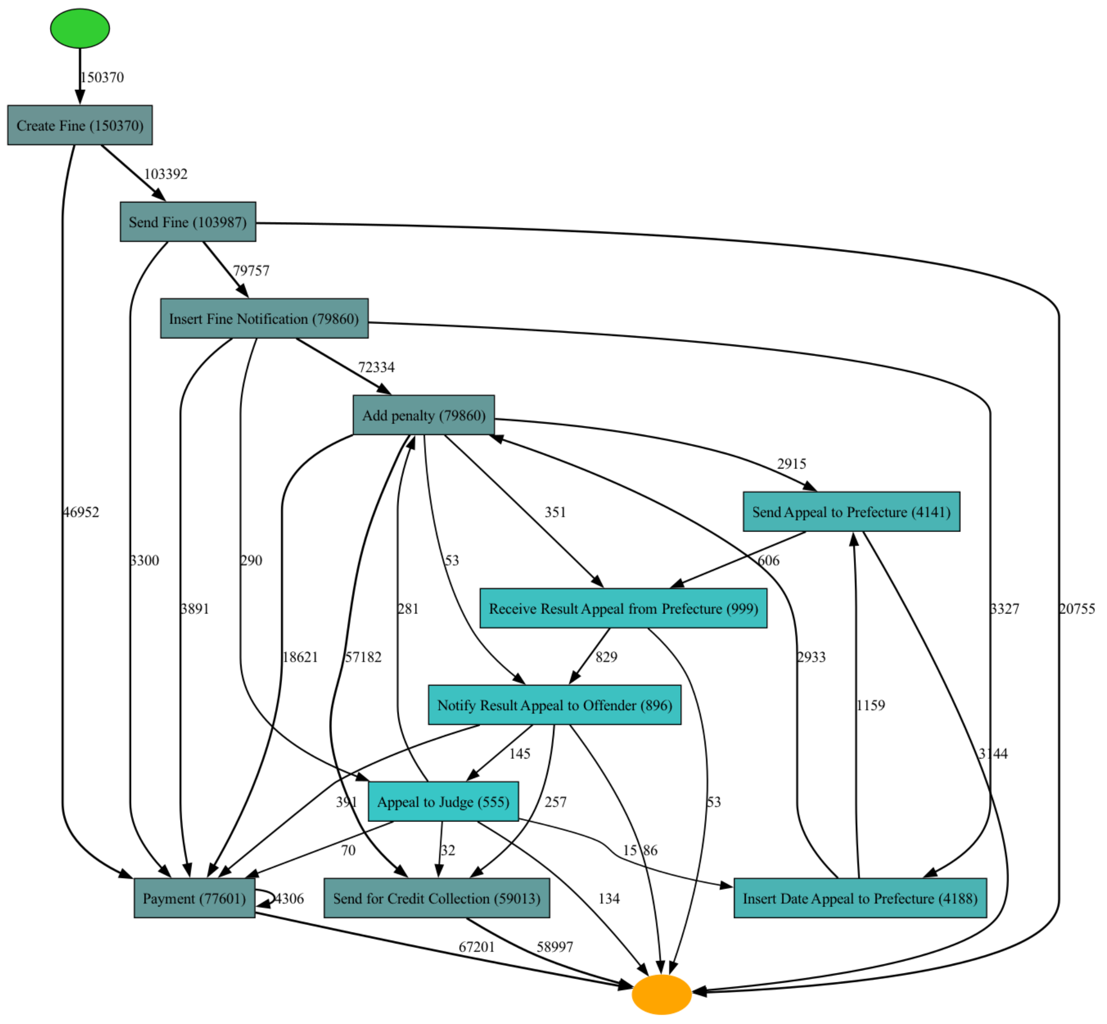
150370
Create Fine (150370)
103392
Send Fine (103987)
79757
Insert Fine Notification (79860)
72334
Add penalty (79860)
2915
Send Appeal to Prefecture (4141)
351
606
Receive Result Appeal from Prefecture (999)
829
Notify Result Appeal to Offender (896)
145
Appeal to Judge (555)
Payment (77601)
4306
Send for Credit Collection (59013)
Insert Date Appeal to Prefecture (4188)
46952
3300
290
3891
18621
57182
281
53
2933
1159
3327
20755
3144
391
257
53
70
32
15
86
134
67201
58997

Figure 3.2: Heuristics Net discovered using Heuristic Miner

In this project, the algorithm was first applied to generate a Heuristics Net, then this net was converted into a Petri Net, which is required for conformance checking using token-based replay technique.

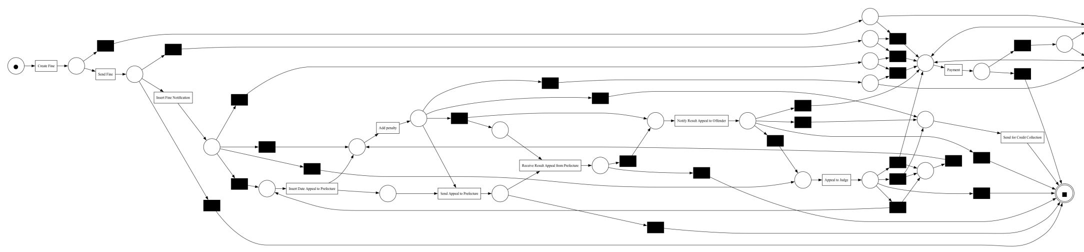
Create Fine
Send Fine
Insert Fine Notification
Add penalty
Insert Date Appeal to Prefecture
Send Appeal to Prefecture
Receive Result Appeal from Prefecture
Notify Result Appeal to Offender
Appeal to Judge
Payment
Send for Credit Collection

Figure 3.3: Petri Net obtained by converting the Heuristics Net (used for conformance checking)

3.3 Conformance checking

Conformance checking is a technique in process mining used to compare a discovered process model with the original event log. Its main objective is to check whether the model can reproduce the observed behavior in the log and to what extent the model allows for behavior that was not actually recorded.

3.3.1 Token-based replay

In this project, conformance checking was carried out using the **token-based replay** technique. This method evaluates how well a trace can be reproduced on a Petri net by simulating the movement of tokens from the initial to the final marking. Identifies which transitions are executed correctly and where tokens are missing or remain unused, indicating deviations or mismatches.

To compare the models discovered by the Alpha Miner and Heuristic Miner algorithms, we used two standard evaluation metrics:

- **Fitness**: indicates how much of the behavior observed in the event log is correctly captured by the process model.
- **Precision**: evaluates how much extra behavior the model allows that was not present in the event log.

In the following, we report the token-based replay results for sample traces using the models discovered by both the Alpha Miner and the Heuristic Miner.

The result obtained from the Alpha Miner model is shown in Listing 3.1. Despite missing tokens, the model does not fully replay the trace, with a fitness score of 0.75. This is due to the the leftover tokens and enabled transitions that are consumed properly, which confirmsconfirms Alpha Miner's limitations in capturing complex control-flow patterns like parallelism and optional paths.

Listing 3.1: Token-based Replay Result on a Sample Trace (Alpha Miner)

```
{
  ’activated_transitions’: [(’Create␣Fine’, ’Create␣Fine’),
                            (’Send␣Fine’, ’Send␣Fine’)],
  ’consumed_tokens’: 3,
  ’missing_tokens’: 0,
  ’produced_tokens’: 6,
  ’remaining_tokens’: 3,
  ’trace_fitness’: 0.75,
  ’trace_is_fit’: False,
  ’enabled_transitions_in_marking’: {
    (’Insert␣Fine␣Notification’, ’Insert␣Fine␣Notification’),
    (’Insert␣Date␣Appeal␣to␣Prefecture’, ’Insert␣Date␣Appeal␣to␣
       Prefecture’),
    (’Appeal␣to␣Judge’, ’Appeal␣to␣Judge’),
    (’Send␣Appeal␣to␣Prefecture’, ’Send␣Appeal␣to␣Prefecture’),
    (’Notify␣Result␣Appeal␣to␣Offender’, ’Notify␣Result␣Appeal␣to␣
       Offender’),
    (’Payment’, ’Payment’)
  },
  ’reached_marking’: {
    ’end’: 1,
    ({’Send␣Fine’}, {’Insert␣Fine␣Notification’}): 1,
    ({’Create␣Fine’}, {’Insert␣Date␣Appeal␣to␣Prefecture’}): 1,
    ({’Insert␣Fine␣Notification’, ’Create␣Fine’}, {’Appeal␣to␣Judge’}): 1
  },
  ’transitions_with_problems’: []
}
```

In contrast, the Heuristic Miner model, shown in Listing 3.2, achieves a perfect replay of the trace with fitness equal to 1.0. There are no missing or remaining tokens and all transitions are executed properly, confirming the ability of the heuristic miner to accurately represent the observed behavior.

Listing 3.2: Token-based Replay Result on a Sample Trace (Heuristic Miner)

```
{
  ’activated_transitions’: [(’Create␣Fine’, ’Create␣Fine’),
                            (’Send␣Fine’, ’Send␣Fine’),
                            (’hid_26’, None)],
  ’consumed_tokens’: 4,
  ’missing_tokens’: 0,
  ’produced_tokens’: 4,
  ’remaining_tokens’: 0,
  ’reached_marking’: [’sink0:1’],
  ’trace_fitness’: 1.0,
  ’trace_is_fit’: True,
  ’transitions_with_problems’: []
}
```

Table 3.1 summarizes the fitness and precision values for both algorithms. The Heuristic Miner clearly outperforms the Alpha Miner in both metrics, showing better alignment with the real process and greater robustness to noise and exceptions.

Algorithm	**Fitness**	**Precision**
Alpha Miner	0.64	0.66
Heuristic Miner	0.93	0.99

Table 3.1: Comparison of Alpha Miner and Heuristic Miner Results

In summary, the Alpha Miner tends to overgeneralize and introduce unrealistic behavior, especially in the presence of loops and parallelisms, while the Heuristic Miner provides a more precise and reliable model of the actual process.

4. Conclusions

This project provided a practical application of Process Mining techniques by comparing two Process Discovery algorithms: **Alpha Miner** and **Heuristic Miner**, using the **PM4Py** library in Python.

The objective was to understand how these algorithms behave when applied to a real event log, specifically the *Road Traffic Fine Management Process*. The comparison was based on the quality of the process models they produced, evaluated using conformance checking metrics such as *fitness* and *precision*.

The main findings are:

- The **Alpha Miner** is simple and fast to apply, but it is very sensitive to noise in the data. It struggles with complex control-flow patterns such as loops or parallel activities, which often leads to incomplete or unrealistic models.
- The **Heuristic Miner** showed much better performance. It is more robust in handling noisy data and can produce models that more accurately reflect the actual process observed in the log.

The Heuristic Miner achieved higher values in both fitness and precision, showing that it provides a better balance between capturing real behavior and avoiding unnecessary or incorrect paths. For this reason, it appears to be more suitable for analyzing real-life business processes.

All the project-related data such as the dataset, discovered models, and Python scripts, are available in the GitHub repository [4].

Bibliography

[1] Wil M. P. van der Aalst, Martin Bichler, Waldemar Böhmer, J. Carmona, M. Leoni, M. Matzner, M. Röglinger, G. Schimm, and Wil van der Aalst. Road traffic fine management process. https://data.4tu.nl/articles/dataset/Road_Traffic_Fine_Management_Process/12683249, 2015.

[2] IEEE Task Force on Process Mining. Ieee xes standard: extensible event stream. https://xes-standard.org/, 2016.

[3] Alessandro Berti, Sebastiaan JJP van Zelst, and Wil MP van der Aalst. Pm4py: Process mining for python. *arXiv preprint arXiv:1905.06169*, 2020. URL https://pm4py.fit.fraunhofer.de.

[4] Sofia Scattolini. Project github repository. https://github.com/Sophisss/pm_project, 2025.