



Comparative Analysis of Process Mining Algorithms in Python

André Filipe Domingos Gomes¹, Ana Cristina Wanzeller Guedes de Lacerda²,
and Joana Rita da Silva Fialho³✉

¹ Polytechnic Institute of Viseu, Viseu, Portugal
estgv15362@alunos.estgv.ipv.pt

² Escola Superior Tecnologia e Gestão de Viseu, CISeD, Polytechnic Institute
of Viseu, Viseu, Portugal
cwanzeller@estgv.ipv.pt

³ Escola Superior Tecnologia e Gestão de Viseu, CI&DEI, Polytechnic Institute
of Viseu, Viseu, Portugal
jfialho@estgv.ipv.pt

Abstract. In many sectors, there is a large amount of data collected and stored, which is not analyzed. The health area is a good example. This situation is not desirable, as the data can provide historical information or trends that may help to improve organizations performance in the future. Process mining allows the extraction of knowledge from data generated and stored in the information systems.

This work aims to contribute to the aforementioned knowledge extraction, comparing different algorithms in process mining techniques, using health care processes and data. The results showed that Inductive Miner and Heuristic Miner are the algorithms with better results. Considering the execution times, Petri Net is the type of model that takes longer, but it is the one that allows a better analysis.

Keywords: Big data in healthcare · PM4Py · Process mining · Process discovery · Conformance checking

1 Introduction

Health processes are complex and involve steps performed by people from various disciplines and sub-areas. This complexity makes this area interesting, but difficult to analyze and understand. These processes make use of information systems that record large volumes of data, but which are difficult to exploit.

Process mining intends to gain knowledge about a particular running process and allows to have an accurate model of its behavior. The purpose is to improve the implementation and evaluation of health care processes. Moreover, the model can help configuring any additional requirements not included in the system [1].

To evaluate the feasibility of some process mining algorithms, using health care processes, each one will be tested to understand its limitations and advantages. For this

purpose, the Process Mining for Python framework (PM4Py) [2] will be used, as it allows an algorithmic customization that other tools do not allow. Furthermore, it has a good variety of other features of interest.

Usually, the event logs used for analysis provide timestamps for the steps/activities that compose the process, as well as their description and other information. The dataset used was extracted from the MIMIC-III database [3].

Subsequently, specific scenarios with certain characteristics are created in order to test different situations that may expose limitations of the algorithms. In addition, the variants are analyzed [4]: select the variants with more occurrences and exclude specific cases that could generate noise in the process analysis. It is also interesting to filter the dataset, taking into account features that make sense in the set of logs. Finally, we intend to test techniques and tools to verify the conformance of the generated model.

It is also possible to calculate different statistics on the event logs of the dataset, as well as to create graphs that allow to understand various aspects of the dataset.

This document is organized into 6 sections. In Sect. 2, the main process discovery algorithms are presented. Section 3 analyzes PM4Py, justifying the choice of this tool. Section 4 explains the experimental scenarios. Section 5 shows the results. Finally, Sect. 6 presents conclusions about this work and future work.

2 Process Mining Algorithms

In the last decade, process mining has emerged as a new field of research that focuses on process analysis, using event data. Classic data mining techniques do not focus on business process models [5]. Thus, process mining focuses on processes, step by step, because the availability of event data and new techniques are increasing, allowing the discovery and the conformance verification of the processes [6].

Process models are used to analyze process execution through Business Process Management (BPM) systems. These process management tools are widely used to support operational process administration. However, they do not use event data [7].

The activities performed by people, machines, and software leave traces in the so called event logs [5]. Process mining techniques use these logs to discover, analyze and improve business processes [8].

Process mining is used to find patterns and understand the causes of certain process behaviors. On the other hand, process mining helps to understand how processes are being performed. For this purpose, specialized mining algorithms are applied to identify patterns from event data recorded in the information management systems [9].

There are several algorithms for process mining. The internal local relations between the activity data are modeled by the Heuristic Miner algorithm. This is the most widely used algorithm, mainly due to its ability to deal with noisy¹ and incomplete data, common in the health area. Global or external relationships between activities are modeled by Genetic Miner algorithms [10] and Fuzzy Miner [11].

Alfa Miner algorithm examines the event log for specific patterns. This algorithm works, simultaneously, a set of sequences of events, following a certain activities order

¹ Noise is the result of data quality problems, such as registration errors, which infrequently manifest themselves in the behavior of the process [13].

in the event log, and shows the result in a Petri Net² project diagram. For example, if activity X is followed by Y, but Y is never followed by X, then it is assumed that there is a causal dependency between X and Y. However, Alpha Miner is unable to highlight the bottlenecks of the process [12].

Most business process mining tools use Directly-Follows Graph (DFGs) as a first approach of exploring event data. To deal with complexity, DFGs are simplified by removing nodes and edges based on frequency restrictions. This simplicity can make these DFGs misleading, as they can be misinterpreted, leading to different conclusions. In addition, bottleneck information can be misleading, especially after simplifying the model. This can lead to all kinds of interpretation problems, due to “invisible gaps” in the model [15].

Heuristic algorithms use the order or sequence of activities and the events frequency. They find the frequent and infrequent paths in the process. In this sense, they are more robust relatively to the process frequencies [16]. Heuristic Miner is very similar to Alpha algorithm, because it deals with similar problems. Furthermore, it catches more real problems. The Heuristic Miner uses logical XOR and AND connectors of dependency relationships. The result of this miner is a heuristic network that helps to visualize the process and predict the flow [12].

Inductive Miner is used widely in different areas, with very promising results. This algorithm has an improvement over the Alpha and Heuristics Miners, as it explore easily an event log. It ensures solidity, as it is able to deal with infrequent behaviors and large event logs. The basic concept of Inductive Miner is to detect a pattern in the logs and then search for that pattern until a base case is found [17].

Table 1 shows the algorithms comparison, according to their characteristics and limitations [18].

3 Process Mining for Python (PM4Py)

Process Mining for Python framework (PM4Py) [2] is a process mining software, easily extensible. It allows conducting large scale experiments easily, and also algorithmic customization. In addition, it is possible to integrate large-scale applications, through a new process mining library. Other libraries can be integrated, such as pandas, numpy, scipy and scikit-learn [20].

The main advantages of the PM4Py library are:

- Allows algorithmic development and customization more easily, when compared to existing tools like ProM [21], RapidProM [21], Disco [22] or Celonis [23];
- Enables easy integration of process mining algorithms with algorithms from other areas of data science, implemented in several state-of-the-art Python packages.
- PM4Py provides support for different types of event data structures, namely event logs, where each line is a list of events. Events are structured as key-value maps;

² A Petri net has two types of elements, positions and transitions. A position can contain one or more tokens. A transition is enabled if all inputs (positions connected to itself) contain, at least, one token [14].

Table 1. Comparison of Process Mining algorithms.

	Alpha Miner	Directly-Follows Graph	Heuristic Miner	Inductive Miner
Description	First mining approach that allows discovering a Workflow network from event logs.	Used as a first approach to exploit event data but can be misleading.	Generates a process model based on different frequency metrics.	It is an improvement over Alpha Miner and Heuristic Miner.
Output	Workflow network.	Directly-Follows Graph (DFG ^a).	Heuristic/casual network.	It can generate several types of models.
Challenges	Noise; Data incomplete; Loop involving one or two stages; Choose not free.	Activities that have a flexible order lead to Spaghetti ^b DFGs with loops	Split and join rules are only considered locally, which results in networks that are not solid.	Generates a solid model from a recursive pattern search.
Result	Extensions can face some challenges.	Performance information can be misleading; Interpretation problems due to gaps in the model.	Can mine long outbuildings successfully; Sometimes it generates many dependencies.	Generates a model that guarantees solidity.
Event logs	Does not deal with incomplete data or noise.	Does not handle long processes, due to its frequency limit.	Possibly, it can handle with incomplete data.	Handles with infrequent behavior and large event logs.

^a Directly Follows are graphs where nodes represent events/activities in the log. Directed links between nodes exist if there is at least one trace in the log where the originating event/activity is followed by the target event/activity. On the top of these targeted links, metrics, such as frequency (counting the number of times the source event/activity is followed by the target event/activity) and performance (the average time between the two events/activities), are represented [15].

^b For processes that are not well structured and have many different behaviors, existing process mining techniques generate highly complex models that are often difficult to understand; these models are called spaghetti models, or spaghetti [19].

- Provides conversion features to transform event data objects from one format to another. Also, PM4Py supports the use of Pandas data frames, which are efficient in case of using larger event data. Other objects currently supported by PM4Py include heuristic networks, Petri networks, process trees³ and transition systems⁴.

PM4Py provides several main process mining techniques, including:

- Process discovery, based on Alpha Miner algorithms [24], Directly-Follows Graph [15], Heuristic Miner [16] and Inductive Miner [17];
- Conformance verification, through token-based alignment and reproduction [25];
- Measurement of suitability, precision, generalization, and simplicity of process models [26];

³ Process tree is a tree-structured process model, where leaf nodes represent activities, and non-leaf nodes represent control flow operators [28].

⁴ Transitional system is used to describe the potential behavior of discrete systems. It consists of states and transitions between states [29].

- Filtering based on time interval, case performance, input and output events, variants, attributes, and paths;
- Case management: statistics on variants and cases;
- Graphs: duration of the case, events by time, distribution of numeric attribute values;
- Social Network Analysis [27]: work handover, joint work, subcontracting and networks of related activities.

PM4Py also provides Python visualization libraries, such as:

- GraphViz: representation of direct sequence graphs, Petri Nets, transition systems, process trees;
- NetworkX: static representation of social networks;
- Pyvis: dynamic web-based social network representation.

4 Data and Experimental Scenario

For the experimental scenarios, data was selected and processed, from the MIMIC-III database. Then, data was converted into the necessary format for the process discovery algorithms of PM4Py application. Finally, for a better analysis of the algorithms, certain test scenarios were defined in order to expose them to different challenges.

4.1 Data Processing

The table schema of the MIMIC-III database (demo version) was analyzed to find the desired information for the test dataset. A subset of tables was selected satisfying the proposed requirements, Fig. 1.

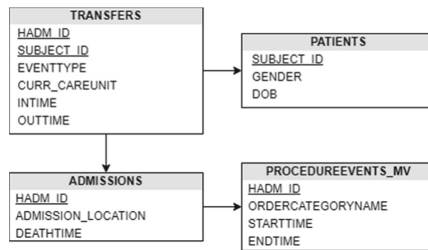


Fig. 1. Scheme of test data.

Analyzing the scheme, the main table is TRANSFERS. It contains the physical locations of patients during hospitalization. The main attributes of this table are: the care unit (CURR_CAREUNIT), if it is a specialty; the entry date and time (INTIME) and the exit date and time (OUTTIME); the type of event (EVENTTYPE) which can be one of three: *admit*, for procedures performed in the patient's admission/evaluation phase; *transf* for the patient's transfer/stay phases and *discharge* for the patient's discharge phases.

Note that, when there is no specialty, the acronym GCU was inserted, translated into General Care Unit. The description of the remaining acronyms of the specialized care units are presented in the Table 2.

Table 2. Description of care units (Adapted from <https://mimic.physionet.org/mimictables/transfers/>).

Care unit	Description
CCU	Coronary care unit
CSRU	Cardiac surgery recovery unit
MICU	Medical intensive care unit
NICU	Neonatal intensive care unit
NWARD	Neonatal ward
SICU	Surgical intensive care unit
TSICU	Trauma/surgical intensive care unit

The SUBJECT_ID attribute connects to the PATIENTS table, which has information about the patients, namely the attributes gender (GENDER) and date of birth (DOB).

Subsequently, HADM_ID attribute is used to access the ADMISSIONS table that contains information about the patient's admission. Using this table, it is possible to collect information about the type/place of admission (ADMISSION_LOCATION) and date of discharge (DISCHTIME) or death (DEATHTIME). It also allows accessing to the PROCEDUREEVENTS_MV table to obtain data related to the events performed in each admission.

The PROCEDUREEVENTS_MV table includes the name of the process (ORDER-CATEGORYNAME) and the date and time of start (STARTTIME) and end (ENDTIME) of the process. Notice that, at this stage, the processes are synchronized with the physical locations, by the respective start/entry and end/exit dates.

4.2 Preparation of the Dataset

From the excel data importation, the respective treatment was made to obtain the dataset format required for the application of the PM4Py process discovery algorithms.

The required format consists in 3 types of information:

- Case ID - a unique identifier for each process;
- Event - a step in the process, any activity that is part of the process;
- Timestamp - date and time for a given event.

The HADM_ID was used as a case identifier, because it is unique. For each stage of the process, the type of event (admission, transfer, or discharge), the care unit (an identifying acronym) and the name of the process performed were added. For the timestamp of the stage, the start date of the process was used or, in cases where a process was not identified, the date of entry into the care unit.

Moreover, other information was used, such as the type of admission, the type of exit (death or discharge), the patient's date of birth, the gender, and the day of the week on

which the event has occurred. In the end, possible duplications were removed from the synchronization of processes with physical locations. Notice that, to use the algorithms in this dataset, the dataset was converted into log format, ordered by timestamp, getting a total of 1163 logs.

4.3 Test Scenarios for the Algorithms

Through the algorithm's analysis and comparison, it was verified that loops between steps and duplications may arise. Thus, admissions were selected to allow testing all these scenarios in isolation.

In an initial scenario, simple admissions were chosen, where none of the cases described above were verified. This scenario, Fig. 2, has, as main objective, a first interaction to test algorithms and their models.



Fig. 2. Simple scenario.

Next, a scenario with duplicate steps, Fig. 3, was selected: there is a step that occurs repeatedly.



Fig. 3. Scenario with duplicate steps.

In the last scenario, the algorithms were exposed to loops between steps. Figure 4 shows a loop occurrence between 2 steps.

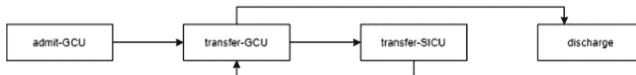


Fig. 4. Scenario with loops between stages.

5 Results

In this section, the results of the tests are presented. The models generated for each tested scenario, execution times, analysis of the log set variants, log set, log set statistics and conformance verification are presented and analyzed.

5.1 Models Analysis for Each Test Scenario

Table 3 presents the models results from the test scenarios. Alpha Miner is unable to create a valid Petri Net model, because it isolates duplicate steps. This result was predictable because this algorithm, admittedly, does not support duplicate steps, neither

Table 3. Models for each test scenario.

Algorithm	Scenario	Result	Models
Alpha Miner	Simple	Valid	
	With duplicate steps	Invalid	
	With loops between 2 steps	Invalid	
Directly-Follows Graph	Simple	Valid	
	With duplicate steps	Valid	
	With loops between 2 steps	Valid	
	For all logs	Invalid	[Spaghetti models]
Heuristic Miner	Simple	Valid	
	With duplicate steps	Valid	
	With loops between 2 steps	Valid	
	For all logs	Valid	[Spaghetti models]
Inductive Miner	Simple	Valid	
	With duplicate steps	Valid	
	With loops between 2 steps	Valid	
	For all logs	Valid	[Spaghetti models]

loops of length one or two [18]. For loops between 2 steps, it generates an invalid Petri Net model, isolating one of the loop steps.

For all logs, Directly-Follows Graph generated a log too large, a spaghetti model. Despite generating this DFG model, it is not a valid one, as the admissions are broken. This result can be justified due the fact that the DFGs are simplified, removing nodes and connections based on frequency limits [15]. However, for other scenarios, the DFG model performance can be considered good.

Heuristic Miner allows the presentation of the frequency of the stages and connections, but it does not mark the most frequent stages and connections [16], which is a disadvantage relatively to Petri Net. This algorithm is compatible with duplicate steps and loop challenges. When the algorithm was converted to Petri Net, the model showed hidden transactions. For a larger number of logs the resulting model is difficult to analyze, as it has created spaghetti models.

Considering all logs, Inductive Miner seems to generate a smaller model, with fewer steps and connections. An explanation of this result may be the improvement that this algorithm has in the search for splits/patterns in the logs. Moreover, it uses many hidden transactions to overcome loops in parts of the model [26].

5.2 Execution Times

Table 4 shows the average of the execution times, in seconds, of the Heuristic Miner and Inductive Miner algorithms for the entire set of logs. These algorithms were able to present a valid model. Notice that each algorithm and model were tested 5 times, under the same conditions, and the average time was calculated after removing the maximum and minimum times. The execution times correspond to the execution of the algorithms, because all logs were already loaded into memory.

Table 4. Execution times for the entire set of logs.

Heuristic Miner	Heuristics Net	Petri Net	Inductive Miner	Process Tree	Petri Net
	4.655	83,254		14,967	34,700

5.3 Variant Analysis

The analysis of variants is extremely important, as it considers the number of occurrences of the variants. This analysis allows to remove the least relevant variants. A variant is a set of cases that share the same perspective of control flow, therefore, a set of cases that share the same events/activities, in the same order [4].

Inductive Miner algorithm was used in this analysis. The results are in Table 5 and contain the description of the variant, the number of occurrences and the respective percentage. Table 6 presents the models generated for different frequencies of variants. The remaining variants have a lower number of occurrences and were discharged. If they were considered, all logs were included, which could turn the analysis impossible and inefficient.

Table 5. Variants.

Variant		Number of occurrences	Percentage
0	admit-MICU, transfer-GCU, discharge	10	7.75%
1	admit-MICU, discharge	5	3.88%
2	admit-TSICU, discharge	4	3.10%
3	admit-MICU, transfer-GCU, transfer-GCU, discharge	3	2.33%
4	admit-MICU-Peripheral Lines, transfer-GCU, discharge	2	1.55%
5	admit-GCU, transfer-SICU, transfer-GCU, transfer-GCU, transfer-GCU, discharge	2	1.55%
6	admit-GCU, transfer-GCU, transfer-SICU, transfer-GCU, discharge	2	1.55%
7	admit-GCU, transfer-GCU, transfer-GCU, transfer-MICU, transfer-GCU, discharge	2	1.55%
8	admit-GCU, transfer-GCU, transfer-GCU, transfer-MICU, discharge	2	1.55%
9	admit-CCU, transfer-GCU, transfer-GCU, discharge	2	1.55%
10... 104	...	1	0.78%

Table 6. Models of different frequencies of variants.

Number of occurrences	Number of Variants	Model
10 or more	1	
5 or more	2	
4 or more	3	
3 or more	4	
2 or more	10	

5.4 Filtering Event Data

In this section, filtrations were tested, Table 7, in the most frequent variants, in order to analyze the process in a different detail. Inductive Miner algorithm was used with the result in a Petri Net model.

5.5 Log Set Statistics

In PM4Py, it is possible to calculate different statistics on the event logs. At Table 8 two statistics can be analyzed using the dataset: average case duration and case dispersion ratio. This last is the average distance between the completion of two consecutive cases in the log.

It is also possible to create graphics, Table 9, to understand various aspects of the dataset used in the model, such as, for example, the distribution of a numeric attribute, the distribution of the case duration, or the distribution of events over time.

Table 7. Filtering models.




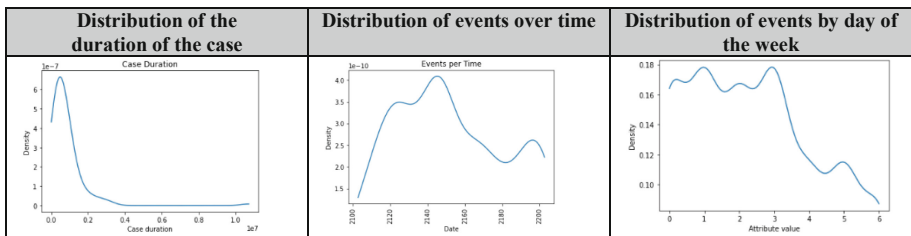
Filtering	Value	Model
Timeframe	"2161-09-19 17:54:42"; "2163-11-21 19:01:00"	
Start activity	"admit-TSICU"; "admit-MICU-Peripheral Lines"	
Attributes values	Type of admission: "TRANSFER FROM SKILLED NUR"	

Table 8. Statistics results to dataset.

Statistics	
Average duration of cases	Average distance
571283.0	93077625.03

Table 9. Event distribution graphs.

5.6 Conformance Verification for Test Logs

Conformance verification is a technique to compare the predicted/expected model of the process with the real model of the process, that is, the set of real event logs for that process. The objective is to verify if the logs are in accordance with the model and vice versa [30]. In PM4Py, two fundamental techniques can be implemented: token-based reproduction and alignments [26].

For this analysis, Inductive Miner algorithm was used with the result in a Petri Net model. Variants with one occurrence were removed. Furthermore, when necessary, a set of two logs was used, where one of them belongs to the predictive model and the other does not.

Token-Based Repetition

The token-based replay corresponds to a Petri Net tracking model, starting from the initial location, to find out which transitions are performed and in which locations there are remaining or missing tokens for the tested log instance. A log conforms to the model if, during its execution, transitions can be triggered without the need to insert any missing tokens [31].

For the model and the set of logs tested, the result is represented in Fig. 5. In the first log, it is clear that the model was unable to satisfy it. The attribute *trace_is_fit* is False, because, in the attribute *transitions_with_problems*, there was a transition in which the path was unable to follow. Hence, 9 produced tokens were consumed and 1 token was missing. Since it managed to satisfy a large part of the route, *trace_fitness* ends up being close to 1, being approximately 0.889.

For the second log, *trace_is_fit* is True. Thus, the model satisfied all the transitions of the log, having consumed all tokens produced, 10, and with no remaining or missing tokens.

trace_is_fit	trace_fitness	activated_transitions	reached_marking	enabled_transitions_in_marking	transitions_with_problems	missing_tokens	consumed_tokens	remaining_tokens	produced_tokens
False	0.8888888888888888	{admit-GCU, tauSplit_2, transfer-MICU, transfer-MICU, transfer-GCU, tauJoin_3, discharge}	{'p_11:1', 'sink:1'}	set()	{transfer-MICU}	1	9	1	9
True	1.0	{admit-MICU, tauSplit_2, transfer-GCU, transfer-MICU, skip_6, transfer-GCU, tauJoin_3, discharge}	{'sink:1'}	set()	{}	0	10	0	10

Fig. 5. Conformance results using token-based repetition.

Alignments

Alignment-based reproduction aims to find one of the best alignments between the log and the model. For each log, the output of an alignment is a list of pairs where the first element is a log event and the second element is a model transition. For each pair, the following classification can be provided [32]:

- Synchronization movement: the classification of the event corresponds to the name of the transition; in this case, the log and the model advance in the same way during the replay;
- Move in the record: pairs where the second element is \gg . This symbol in the second element corresponds to a repetition movement in the log that is not similar in the model. This type of movement is inappropriate and there is a deviation between the log and the model;
- Move in the model: pairs where the first element is \gg . This situation corresponds to a repetition movement in the model that is not similar in the log. For movements in the model, we can make the following distinction:
 - Movements in the model involving hidden transitions: in this case, even if it is not a synchronized movement, the movement is adequate;
 - Movements in the model that do not involve hidden transitions: in this case, the movement is inappropriate and means a deviation between the log and the model.

Each log conformance check is associated with a dictionary, containing, among others, the following information:

- **Alignment:** contains the alignment (synchronization movements, movements in the register, movements in the model);
- **Cost:** contains the cost of the alignment according to the cost function provided, which can be customized;
- **Fitness:** is equal to 1 if the log is perfectly adequate.

For the model and the set of logs tested, the first log had a fitness close to 1, that is, there is an adaptation to the model close to 1 (but lower than 1), indicating that it was not able to complete the entire process path from the model. On the other hand, in the second log, the process was able to finalize the log path, Fig. 6.

alignment	cost	queued states	visited states	closed set length	num visited markings	exact heu calculations	fitness
[('admit-GCU', 'admit-GCU'), ('transfer-MICU', '>>'), ('>>', None), ('transfer-MICU', 'transfer-MICU'), ('transfer-GCU', 'transfer-GCU'), ('>>', None), ('discharge', 'discharge')]	10000	20	10	7	8	3	0.8571428571428572
[('admit-MICU', 'admit-MICU'), ('>>', None), ('transfer-GCU', 'transfer-GCU'), ('transfer-MICU', 'transfer-MICU'), ('>>', None), ('transfer-GCU', 'transfer-GCU'), ('>>', None), ('discharge', 'discharge')]	0	21	10	8	8	1	1.0

Fig. 6. Results of alignments.

Overall Assessment of the Model by the Set of Test Logs

In PM4Py, it is possible to obtain different information on the comparison between the behavior contained in the test logs and the behavior contained in the model, to verify if and how they correspond. There are four different dimensions of conformance in Process Mining: the measurement of the adequacy of the replay, the measurement of precision, the measurement of generalization and the measurement of simplicity.

The calculation of the adequacy of the replay aims to calculate how much of the behavior in the log is admitted by the process model. Two methods are proposed to calculate the adequacy of replay: replay and alignments, both based on token, previously used for isolated logs.

For precision or accuracy, the set of transitions in the process model is compared with the set of activities logs that follow the model [26]. For that, unvisited branches are counted. Unvisited branches are decisions that are possible in the model and not in the event log. If not, the accuracy is perfect. This analysis can also be obtained from the two methods mentioned in the previous subsection, where token-based reproduction is faster, but based on heuristics. Therefore, the result may not be accurate [31]. Alignments are accurate, work on any type of network, but can be slow [32].

Generalization is the third dimension to analyze how the log and the process model coincide. Basically, a model is general if the elements of the model are visited often enough during a reproduction operation.

Finally, simplicity is the fourth dimension for analyzing a process model. In this case, simplicity is defined considering only the Petri Net model. This metric considers the number of incoming and outgoing connections that each transition has [33]. For all these metrics, the resulting value varies between 0 and 1.

Figure 7 describes those evaluations, showing that, according to the calculation of the adequacy of the replay, the adaptation of the set of logs tested to the model was high, but

not complete. Precision, on the other hand, proved to be quite low, for both approaches. The using of hidden transactions and the fact of one of the logs had not completed the path can explain this result. The set of logs tested has many repeated steps, leading to a low generalization. Besides, it was considered a few steps in the model. As the model has hidden transactions and loops, simplicity is low, since there are situations of join or split in steps.

Replay Fitness		Precision		Generalization	Simplicity
Token-based Replay	Alignments	Token-based Replay	Alignments		
0.9444444444444444	0.9285714285714286	0.34615384615384615	0.34615384615384615	0.12314136557579325	0.5789473684210527

Fig. 7. Results of the evaluation of the Log-Model.

6 Conclusion

From the results with the experience scenarios, Alpha Miner was not able to deal with duplicated steps and loops between two steps. Directly-Follows Graph achieved that, but in turn, for a larger set of logs, the generated model was invalid, not being able to represent cases with more than 5 steps.

For the other algorithms, they were really able to deal with challenges and larger volumes of logs. Inductive Miner was the algorithm that better handled with duplicated steps and loops between 2 steps. It uses hidden steps more recurrently, mainly in loop parts.

Considering the models tested, the Process Trees are the most difficult to analyze due to their syntax. The Petri Net models proved to be more efficient and structured. Based on the execution times, Petri Net is the type of model that takes longer to run for a larger volume of logs but allows a better analysis.

For large amounts of data, the Petri Net model of Inductive Miner was the one that had the longest execution time, but it was also the one that had the best result. Due to the improvement that this algorithm has, the model, in general, is more organized and easier to analyze [26].

Table 10 summarizes the results achieved where the comparison parameters are presented in order of priority. If an algorithm has limitations to challenges, it is no longer analyzed in the next parameters. Thus, the most suitable algorithm is the Inductive Miner.

Table 10. Summary of the conclusions.

Algorithm	Limitations on challenges	Petri Net model simplicity	Runtime
Alpha Miner	-		
Directly-Follows Graph	-		
Heuristic Miner	+	-	-
Inductive Miner	+	+	+

For future work it is intended to expand these experiences to different areas and types of dataset. Another important aspect would be the execution of these same tests in other existing tools, as they may have different implementations of algorithms and functionalities.

Acknowledgements. This work is funded by National Funds through the FCT - Foundation for Science and Technology, I.P., within the scope of the project Ref UIDB/05583/2020. Furthermore, we would like to thank the Research Centre in Digital Services (CISeD), the Polytechnic of Viseu for their support.

This work is also funded by National Funds through the FCT - Foundation for Science and Technology, I.P., within the scope of the project Ref^a UIDB/05507/2020. Furthermore we would like to thank the Centre for Studies in Education and Innovation (CI&DEI) and the Polytechnic of Viseu for their support.

References

1. Hendricks, R.: Process mining of incoming patients with sepsis. *Online J. Public Health Inform.* **11**(2) (2019). <https://doi.org/10.5210/ojphi.v11i2.10151>
2. Fraunhofer Institute for Applied Information Technology: Process Mining for Python (PM4Py). *Process Mining for Python (PM4Py)* (2021). <https://pypi.org/project/pm4py/>
3. Kurniati, A.P., Hall, G., Hogg, D., Johnson, O.: Process mining in oncology using the MIMIC-III dataset. In: *Journal of Physics: Conference Series*, vol. 971, no. 1 (2018). <https://doi.org/10.1088/1742-6596/971/1/012008>
4. Bolt, A., van der Aalst, W.M.P., de Leoni, M.: Finding process variants in event logs. In: Panetto, H., et al. (eds.) *OTM 2017. LNCS*, vol. 10573, pp. 45–52. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-69462-7_4
5. Van Der Aalst, W.: Process mining: overview and opportunities. *ACM Trans. Manag. Inf. Syst.* **3**(2), 1–17 (2012). <https://doi.org/10.1145/2229156.2229157>
6. Mans, R.S., Van Der Aalst, W.M.P., Vanwersch, R.J.B.: *Process Mining in the Healthcare* (2015)
7. Pegoraro, M., Uysal, M.S., van der Aalst, W.M.P.: Discovering process models from uncertain event data. In: Di Francescomarino, C., Dijkman, R., Zdun, U. (eds.) *BPM 2019. LNBIP*, vol. 362, pp. 238–249. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-37453-2_20
8. Batista, E., Solanas, A.: Process mining in healthcare: a systematic review. In: *2018 9th International Conference on Information, Intelligence, Systems and Applications, IISA 2018*, pp. 1–6 (2019). <https://doi.org/10.1109/IISA.2018.8633608>
9. Rojas, E., Cifuentes, A., Burattin, A., Munoz-Gama, J., Sepúlveda, M., Capurro, D.: Performance analysis of emergency room episodes through process mining. *Int. J. Environ. Res. Public Health* **16**(7) (2019). <https://doi.org/10.3390/ijerph16071274>
10. Shinde, S.A., Rajeswari, P.R.: Intelligent health risk prediction systems using machine learning: a review. *Int. J. Eng. Technol. (UAE)* **7**(3), 1019–1023 (2018). <https://doi.org/10.14419/ijet.v7i3.12654>
11. Wang, L., Du, Y., Qi, L.: Efficient deviation detection between a process model and event logs. *IEEE/CAA J. Automatica Sinica* **6**(6), 1352–1364 (2019). <https://doi.org/10.1109/JAS.2019.1911750>
12. Sundari, M.S., Nayak, R.K.: Process mining in healthcare systems: a critical review and its future. *Int. J. Emerg. Trends Eng. Res.* **8**(9), 5197–5208 (2020). <https://doi.org/10.30534/ijeter/2020/50892020>

13. Conforti, R., La Rosa, M., ter Hofstede, A.H.M.: Noise Filtering of Process Execution Logs based on Outliers Detection. Institute for Future Environments, School of Information Systems; Science & Engineering Faculty, pp. 1–16 (2015)
14. de Petri, R.: Rede de Petri. Wikipédia, a enciclopédia livre (2019). https://pt.wikipedia.org/w/index.php?title=Rede_de_Petri&oldid=55172483
15. Van Der Aalst, W.M.P.: A practitioner’s guide to process mining: limitations of the directly-follows graph. *Procedia Comput. Sci.* **164**, 321–328 (2019). <https://doi.org/10.1016/j.procs.2019.12.189>
16. Weijters, A.J.M.M., van der Aalst, W.M.P., de Medeiros, A.K.A.: Process Mining with the Heuristics Miner Algorithm. Beta Working Papers (2006)
17. Bogarín, A., Cerezo, R., Romero, C.: Discovering learning processes using inductive miner: a case study with learning management systems (LMSs). *Psicothema* **30**(3), 322–329 (2018). <https://doi.org/10.7334/psicothema2018.116>
18. Breitmayer, M.: Applying Process Mining Algorithms in the Context of Data Collection Scenarios (2018)
19. Veiga, G.M., Ferreira, D.R.: Understanding spaghetti models with sequence clustering for ProM. In: Rinderle-Ma, S., Sadiq, S., Leymann, F. (eds.) *BPM 2009. LNBP*, vol. 43, pp. 92–103. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-12186-9_10
20. Berti, A., Van Zelst, S.J., Van Der Aalst, W.M.P., Gesellschaft, F.: Process mining for python (PM4py): bridging the gap between process-and data science. In: *CEUR Workshop Proceedings*, vol. 2374, pp. 13–16 (2019)
21. van Dongen, B.F., de Medeiros, A.K.A., Verbeek, H.M.W., Weijters, A.J.M.M., van der Aalst, W.M.P.: The ProM framework: a new era in process mining tool support. In: Ciardo, G., Darondeau, P. (eds.) *ICATPN 2005. LNCS*, vol. 3536, pp. 444–454. Springer, Heidelberg (2005). https://doi.org/10.1007/11494744_25
22. Lohmann, N.M.: Discover Your Processes Disc. *Proceedings*, September (2012)
23. Badakhshan, P., Geyer-Klingeberg, J., El-Halaby, M., Lutzeyer, T., Affonseca, G.V.L.: Celonis process repository: a bridge between business process management and process mining. In: *CEUR Workshop Proceedings*, vol. 2673, pp. 67–71 (2020)
24. Van Der Aalst, W., Weijters, T., Maruster, L.: Workflow mining: discovering process models from event logs. *IEEE Trans. Knowl. Data Eng.* **16**(9), 1128–1142 (2004). <https://doi.org/10.1109/TKDE.2004.47>
25. Adriansyah, A., Sidorova, N., Van Dongen, B.F.: Cost-based fitness in conformance checking. In: *Proceedings - International Conference on Application of Concurrency to System Design, ACSD 2011*, pp. 57–66 (2011). <https://doi.org/10.1109/ACSD.2011.19>
26. Pohl, T.: An Inductive Miner Implementation for the PM4PY Framework, pp. 1–66 (2019)
27. van der Aalst, W.M.P., Song, M.: Mining social networks: uncovering interaction patterns in business processes. In: Desel, J., Pernici, B., Weske, M. (eds.) *BPM 2004. LNCS*, vol. 3080, pp. 244–260. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-25970-1_16
28. Arriagada-Benítez, M., Sepúlveda, M., Muñoz-Gama, J., Buijs, J.C.A.M.: Strategies to automatically derive a process model from a configurable process model based on event data. *Appl. Sci. (Switzerland)* **7**(10) (2017). <https://doi.org/10.3390/app7101023>
29. See, E.: *Transition System on*, pp. 1–17 (2005)
30. Muñoz-Gama, J., Carmona, J.: Enhancing precision in process conformance: stability, confidence, and severity. In: *IEEE SSCI 2011: Symposium Series on Computational Intelligence - CIDM 2011: 2011 IEEE Symposium on Computational Intelligence and Data Mining*, pp. 184–191 (2011). <https://doi.org/10.1109/CIDM.2011.5949451>
31. Berti, A., van der Aalst, W.M.P.: A novel token-based replay technique to speed up conformance checking and process enhancement. In: Koutny, M., Kordon, F., Pomello, L. (eds.) *Transactions on Petri Nets and Other Models of Concurrency XV. LNCS*, vol. 12530, pp. 1–26. Springer, Heidelberg (2021). https://doi.org/10.1007/978-3-662-63079-2_1

32. Bloemen, V., Van Zelst, S., Van Der Aalst, W.: Aligning Observed and Modeled Behavior by Maximizing Synchronous Moves and Using Milestones (2019)
33. Buijs, J.C.A.M., Van Dongen, B.F., Van Der Aalst, W.M.P.: Quality dimensions in process discovery: the importance of fitness, precision, generalization, and simplicity. *Int. J. Coop. Inf. Syst.* **23**(1), 1–39 (2014). <https://doi.org/10.1142/S0218843014400012>