Formal Methods in Computer Science

Case study Report

Process mining with Petri nets: an application to daily routine modeling

Author
Davide Lofrese
(matr. 722950)

A.Y. 2022/2023

Contents

Introduction

Process mining is the application of data science techniques to extract insights from event logs describing the execution of a process to gain a better understanding, monitor or improve the process.

There are three main types of process mining: discovery, conformance and enhancement. The first is related to the automatic discovery, from event logs, of a process model which tries to capture the behavior of the process and can be expressed in various formalisms (e.g., process trees, Petri nets, etc.). The second is related on comparing an existing process model with an event log to check if one conforms to the other. Finally, the third type is related to the improvement of an existing process model using the insights gained by analyzing the process execution.

Although process mining is typically used to analyze business processes, in the recent years it has been successfully applied also to human processes (i.e., routines of daily activities) to extract insights from event logs recorded by sensors in a smart home environment. Hence, by leveraging process discovery techniques, a model of the daily routine of the user can be discovered and applied in multiple ways. For instance, the smart environment may use the model to predict the next activity and adapt consequently to support the user, or deviations from the typical routine could be detected in order to recognize emergency situations. This is of particular importance in Ambient Assisted Living (AAL) which is a research area that focuses on giving impaired or elderly people the opportunity to stay independent in their home.

The goal of this case study is the practical application of process mining techniques to discover and evaluate a process model, expressed with Petri nets, of the daily routine of a user based on a labeled event log containing the activities carried out during the day.

This report describes the techniques that have been applied and the results of the process discovery. In particular, the document is divided in two main sections. The first introduces the process discovery activity by giving a general outline of the input required, the algorithms, the evaluation metrics and the software tools. Then, in the second section, a general analysis of the studied event log used is provided, followed by a description of the methodology which has been adopted to discover, evaluate and compare multiple models. Finally, the results of the discovery activity are presented, analyzed and discussed.

The event logs and the models discovered in the case study are available on GitHub: `https://github.com/davidelofrese/daily-routine-discovery`

1 Process discovery

Process discovery is the process mining task which deals with the automatic construction of a process model that captures the behavior of the process.

1.1 Event log

The starting point for process discovery (and for any other process mining activity) is an event log, that is a collection of events related to a single process where each event is part of a single process instance (case). In more detail, a case is made up by a sequence of events which represent the various steps taken during the process.

An event has multiple attributes with the related information. In particular, each event has a case identifier which specifies the case containing the event, an activity identifier which specifies the name of the activity performed and, furthermore, it can have additional attributes such as the date and time of execution, the cost, the resource who carried out the activity, etc.

Note that each case has to be uniquely identified, and the events within a case must be ordered for discovering the causal dependencies while learning the process model. The order can be assigned by using the timestamps of the events or, if no timestamps are provided as events attributes, by considering the position of the event within the case.

Given an event log, a process discovery algorithm is a function that maps the log onto a process model representative for the behavior of the process. In other words, the algorithm identifies patterns and regularities in the log and translate them in a process model expressed in a process modeling notation.

1.2 Process modeling notations

In general, process discovery algorithms are able to generate models expressed in various notations (e.g., Petri net, BPMN, process tree, etc.). In this case study, it has been chosen to use Petri nets as a formalism for the discovered models. This is due to the fact that Petri nets are a simple, but powerful model representation language and they allow to express parallelism, choice and iteration. Furthermore, they have an intuitive graphical notation which allows to quickly grasp the behavior of a process by looking at the representation of the net.

It is important to point out that, typically, when modeling processes with a Petri net, they are in fact modeled with a subclass of Petri nets, namely the WorkFlow nets (WF-nets). In more detail, a WF-net is a Petri net with a unique source place where the process starts and a unique sink place where the process ends. Furthermore, all the nodes in a WF-net are on a path from source to sink. A simple example of WF-net is shown in Figure 1 where it is possible to observe the unique source place (on the very left) and the unique sink place (on the right).

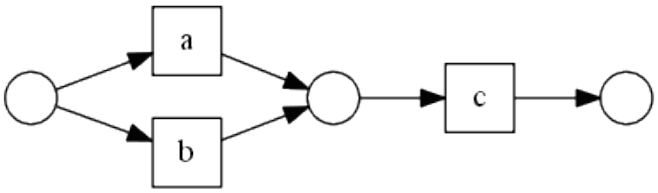


Figure 1: Simple example of a WorkFlow net

In process discovery, WF-nets are preferred to Petri nets since they allow to model the life-cycle of the cases by expressing explicitly the creation and the completion of a case. In particular, the former is modeled by putting a single token in the source place, while the latter is modeled by reaching a marking where there is a single token just in the sink place.

Note that not every WF-net represents a correct process. This is caused by the fact that a net may have deadlocks or unreachable transitions. Therefore, the focus is on correct WF-nets, that is sound WF-nets. These have four properties: *safeness* (each place cannot hold multiple tokens at the same time), *proper completion* (if the sink place is marked all the other places must be empty), *option to complete* (it is always possible to mark the sink place) and *absence of dead parts* (all the transitions can be enabled).

Before proceeding, it is worth mentioning that it is possible to automatically translate a model expressed in a specific notation into a model expressed in another notation. For instance, a process tree can be translated in a Petri net. This is useful as some process mining algorithms are not able to directly discover WF-nets, but instead they discover process trees that can be, then, converted in a sound WF-net. However, this conversion may require the introduction of silent transitions in the model, i.e., transitions that do not correspond to an actual activity in the process, but are just used to model some structures in the WF-net. As an example, consider the start and end of a parallel execution where multiple activities are carried out. This can be modeled with two silent transitions, one which starts the concurrent flow and one at the end.

1.3 Model evaluation metrics

Before giving an overview of some of the process discovery algorithms, the metrics which can be used for the evaluation of the discovered models are presented. In particular, the evaluation of a discovered process model is characterized by four main quality metrics: fitness, precision, generalization and simplicity.

An overview of the four quality criteria is shown in Figure 2 and they are explained in more details in the following paragraphs.

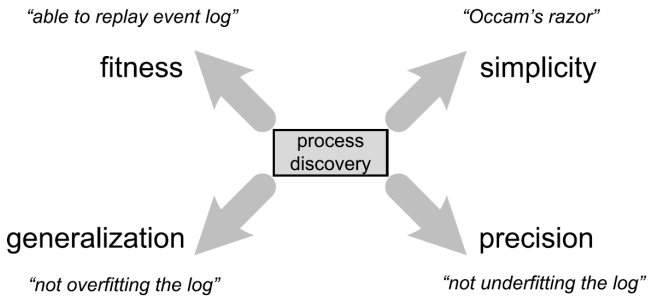


Figure 2: Overview of the four quality criteria

1.3.1 Fitness

This metric estimates the model ability to replay the log. More specifically, it measures the proportion of behavior in the event log which is allowed in the model. It can be computed at the case level (considering the fraction of cases in the log that can be fully replayed on the model) or at the event level (considering the fraction of events in the log that are possible in the model).

Typically, the fitness value is quantified between 0 and 1. Thus, a fitness value equal to 0 means that the model is not able to replay any of the cases/events in the log. On the other hand, a value of 1 means that the model has a perfect fitness and all the events in all the cases of the log can be replayed.

Note that when computing fitness, multiple penalties must be set. For instance, it is necessary to decide what is the cost of a move in the event log, but not in the model, or the cost of a move in the model, but not in the event log.

1.3.2 Precision

This metric measures how much of the behavior allowed by the model is actually present in the log.

In particular, a low precision value means that the model allows for a lot of behavior which is not featured in the log. Hence, the model allows for sequences of activities very different from the ones in the log. In other words, a model with low precision is underfitted and over-generalizes the behavior observed in the log. An extreme example of over-generalized model is the so-called «flower model» which is able to generate any sequence of the activities in event log.

Vice versa, a high precision value means that the model does not allow much more behavior than the one in the log. An extreme example is the «enumerating model» where just the cases in the log are allowed.

Also in this case, the value is quantified between 0 and 1.

1.3.3 Generalization

This metric quantifies how much future behavior not present in the log will already be allowed in the model. In particular, it is important that the model does not restrict the allowed behavior just to the examples in the event log (as the «enumerating model» does), but that it is general.

Conversely to precision, a model with a low generalization value is overfitted and, thus, too specific failing to capture the general behavior of the process besides the cases in the log.

1.3.4 Simplicity

This metric quantifies how much complex is the model and how much easy to understand is to a human. This metric is related to the Occam's Razor meaning that the simplest model that captures the behavior in the event log is the best one.

The simplicity can be computed in multiple ways. For instance, it is possible to consider the number of places and transitions in a Petri net or to consider the «structuredness» of the model.

An example of a simple metric to compute simplicity is the extended Cardoso metric which is based on the number of subsets of places reachable from each place in the Petri net. Hence, the higher (lower) the extended Cardoso metric value, the more (less) complex is the model.

1.3.5 Balancing the four metrics

It is important to point out that the four quality metrics can be in conflict among them. For instance, a high precision value usually corresponds to a low generalization value where the model is overfitted. Similarly, a very simple model will have a low fitness value because the model will be too simple to capture the behavior seen in the log.

Note that the previously mentioned «flower model» has perfect fitness (since it is able to replay all the events in the log), very low precision and high generalization (since it does allow much more behavior than the one in the log), and high simplicity. Likewise, the «enumerating model» has perfect fitness, high precision, but very low generalization and simplicity.

Hence, in general, it is important that process discovery algorithms should be able to make trade-offs in order to balance the four criteria.

1.4 Process discovery algorithms

There are many algorithms which can be leveraged for process discovery. Before giving an overview of the ones used in this case study, it is important to mention that, in general, the event log is a collection of specific examples of the process behavior. Hence, similarly to the traditional machine learning context, it is not possible to assume to have a log which contains all the possible process behaviors. In other words, the log contains a fraction of the process behavior, but a lot of additional cases could be possible. Furthermore, differently from the traditional machine learning context, here no negative examples are included in the log. Thus, the log only contains examples of possible behavior, without any explicit description of behaviors that are impossible.

As a consequence, all of the algorithms assume that the event log contains a *representative* sample of behavior. However, there are two issues that might make an event log less representative: *noise* and *incompleteness*.

In particular, noise is caused by the fact that the log contains rare and infrequent events which do not happen commonly in the process execution. Thus, a noisy log is not representative of the typical process behavior. Instead, incompleteness refers to the fact that the log contains too few events for the algorithm to capture the process.

In this case study, it has been chosen to use and compare the models discovered by three algorithms: alpha miner, inductive miner and genetic miner.

1.4.1 Alpha miner

The α-algorithm has been one of the first process discovery algorithms able to manage concurrent activities in the process.

It works by scanning the event log to find four different types of ordering relations among the activities. For example, if an activity a is followed by an activity b, but b is never followed by a, then it is assumed that there is a causal dependency between a and b. Given a log L and two activities a and b, the ordering relations are $a >_L b$ (when a directly follows b in a case), $a \rightarrow_L b$ (when $a >_L b$, but $a \not>_L b$), $a \#_L b$ (when $a \not>_L b$ and $b \not>_L a$) and $a \parallel_L b$ (when $a >_L b$ and $b >_L a$).

Consequently, based on the relations, the α-algorithm discovers the relevant patterns in the log and builds a Petri net which represents the log and the relations among the activities. For instance, the relation $a \rightarrow_L b$ is represented in the Petri net with two transitions (a and b) connected by a place as shown in Figure 3.

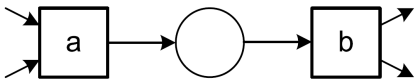


Figure 3: Example of the directly follow relation in a Petri net

Although able to handle concurrency, the original α-algorithm has issues with noise and incompleteness. In particular, this algorithm has a weak notion of completeness meaning that if in the process an activity a can be directly followed by an activity b, then this should appear at least once in the log. Hence, if no cases with this relation are provided, the algorithm will not be able to discover it. Thus, over time, multiple variants of the original α-algorithm have been developed such as the α^+-algorithm (which deals with short loops where the same event is repeated multiple times) and the α^{++}-algorithm (which deals with situations where there is a mixture of choice and synchronization).

1.4.2 Inductive miner

The inductive process discover techniques can be seen as one of the leading process discovery approaches. They work by recursively executing three steps on the event log: directly-follows graph (dfg) construction, cut detection and split.

In particular, given an event log, a directly-follows graph is built based on the directly-follows relation (which corresponds to the $>_L$ relation of the α-algorithm). Then, a cut detection is performed where the algorithm looks for a cut in the dfg (among four possible cut types). If there is a cut, the cut detection finishes and the log is split into one or more sublogs. On the other hand, if there is not a cut, a fall through is selected (i.e., a structure which allows for any behavior involving the activities in the log). Following, the three steps are repeated recursively on each sublog until a base case (i.e., a sublog containing a single activity) is reached.

Note that the inductive miner discovers process trees. The advantage is that process trees are sound by construction. However, as previously mentioned in Section 1.2, they can be easily translated in sound WF-nets (with the introduction of silent transitions if required). Hence, a WF-net learnt by a inductive miner will be always sound.

The basic inductive miner (IM) algorithm guarantees perfect fitness (i.e., a fitness value of 1) and the model is able to replay all the cases in the log. Moreover, the produced models tend to be simple and general. Nevertheless, the basic algorithm is not able to handle infrequent behaviors and incompleteness. This issues can be solved since the inductive miner is flexible and many variants of the basic approach are available.

Some examples of extensions are the inductive miner–infrequent (IMf) algorithm and the inductive miner–incompleteness (IMc) algorithm. The former works similarly to the basic inductive miner, but it filters out from the dfg the directly-follows relations which occur with a frequency below a threshold. Then, the resulting dfg is used for cut detection and splitting. This allows to filter out infrequent behaviors. The IMc algorithm instead, addresses the problem of missing behaviors due to the incompleteness of the event log.

Furthermore, there is an extension which is able to effectively handle the event type information in the event log, namely the inductive miner–life-cycle (IMlc). This algorithm is able to better detect concurrency in the activities by leveraging the event type data. In addition, the IMf and IMlc extensions can be combined to obtain the inductive miner–infrequent & life-cycle (IMflc) extension.

Note that, in contrast with the basic inductive miner, the variants do not always guarantee perfect fitness, but this trade-off is needed to discover a better model.

1.4.3 Genetic miner

Genetic algorithms are a search technique inspired by the natural biological evolution process. Differently from the alpha miner and the inductive miner, this technique is not deterministic and is based on randomization.

An overview of the functioning of the genetic miner is shown in Figure 4. In particular, it works by repeating three steps iteratively, namely: initialization, selection and reproduction.

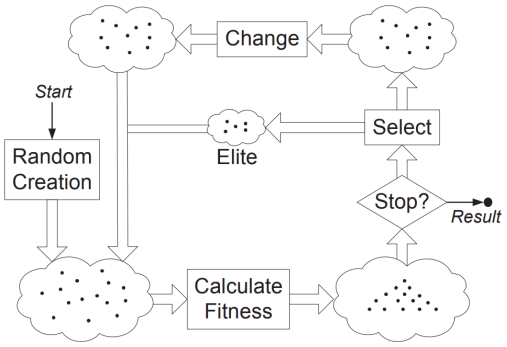


Figure 4: Overview of the steps executed by the genetic miner

Firstly, in the initialization step the first population of individuals is created. More in detail, a large number of random process models is built by considering the activities in the log. For instance, a Petri net might be constructed by randomly adding places and transitions with the activity names. Note that these initial models might be of very poor quality and do not reflect the behavior in the log. However, due to the large number of models, some parts of some of the models might fit some parts of the event log.

Following, in the selection step the quality of each model is computed. In this way, the best models are selected and moved to the next generation without modifications (elitism).

Finally, in the reproduction step some of models (both from the elite and not) are randomly picked as parents to create a new generation of individuals (note that the best models have a higher probability to be selected as parents). In particular, the models from a generation are used to create new models for the next generation by using two operators: *crossover* and *mutation*. In the former, two parents are taken and combined to obtain a new model which has parts of the «genetic material» of both parents. In the latter, the new models are modified with random changes. This second operation is needed to add new genetic material and evolve beyond the first generation. In this way, the next generation contains both the best models from the previous one and the new models generated.

The three steps are repeated iteratively on the new generation. The idea is that the quality of each generation becomes better and better than the previous ones. The algorithm stops when a process model with the desired quality is found or after a certain number of generations has been produced.

One algorithm which implements this approach is the Evolutionary Tree Miner (ETM) which is able to balance the four quality metrics. This algorithm discover process trees that can be, then, converted to sound WF-nets.

The genetic miner is able to deal with noise and incompleteness, but it is not very efficient for large event logs and models meaning that it can take a long time to discover a model with the desired quality.

1.5 Process discovery tools

In order to effectively apply process discovery algorithms and to evaluate the results, software tools with dedicated process mining capabilities are required.

These software support the process mining activity by providing a set of tools which implement various process mining techniques, including the process discovery algorithms and the evaluation metrics.

The following sections describe the XES format, which is the standard format for event logs, and ProM Tools, a popular process mining software.

1.5.1 XES format

The eXtensible Event Stream (XES) format is the standard exchange format for process mining. It has been adopted since 2010 by the IEEE Task Force on Process Mining as the successor of MXML, of which XES is a less restrictive and extensible improvement.

This format is based on an XML document which contains an event log made up of traces each of which contains the list of the events corresponding to a case.

Each XML element in a XES file can have any number of attributes to express its features. The attributes' semantics is given by extensions, each defining a prefix and a set of attributes. For instance, the concept extension defines the *concept* prefix and the *name* and *instance* attributes. The former allows to express the name of traces and events, while the latter expresses the progressive number of an activity in a case. Another example is the life-cycle extension which defines the *lifecycle* prefix and the *transition* attribute which allows to express the event type (e.g., schedule, start, complete, etc.).

1.5.2 ProM Tools

ProM Tools is a popular open-source framework for process mining. It supports the main process mining techniques (including the ones described in the previous sections) by using a set of plug-ins which implement them.

The current version is ProM 6 which has been released in 2010 to support the XES format. It includes a variety of plug-ins that aid in carrying out the process mining tasks. Hence, there are multiple plug-ins for each process mining activity. For instance, some plug-ins are used to load and analyze the event log, some others to apply process mining algorithms and compute the metrics to evaluate the results.

Finally, it implements multiple visualization tools to visualize some features of the event logs and to show the discovered process models. For example, it allows to visualize a Petri net discovered on the event log.

2 Case study: daily routine discovery

2.1 Event log

This case study is based on an event log which contains a collection of events related to the daily routine of a user. More specifically, every Monday, the daily routine of the user has been recorded by means of sensors embedded in a smart home environment. Then, the raw data as collected by the sensors have been labeled assigning to each event a specific activity performed by the user during the day (e.g., reading, writing, sitting down, etc.).

In order to use the event log and apply process discovery techniques, it has been necessary to convert it in a suitable format supported by ProM 6, which is the process mining tool chosen for this case study.

The following sections describes the steps taken for the conversion.

2.1.1 Raw event log

The initial event log has been provided in a plain text file where each row corresponds to an event. To better understand the format of the data, an excerpt of the file is reported in Figure 5.

```
entry(1,begin_of_process,monday,monday6637,none,none).
entry(2,begin_of_activity,monday,monday6637,sit_down,1).
entry(3,begin_of_activity,monday,monday6637,eat_meal,1).
entry(4,end_of_activity,monday,monday6637,eat_meal,1).
entry(5,end_of_activity,monday,monday6637,sit_down,1).
```

Figure 5: Raw event log excerpt

As it can be seen from the figure, each event is an entry expressed as a tuple with six items (T, E, W, P, A, O), each referring to a feature of the event. Specifically, T is the timestamp, E is the type (begin/end of the process, begin/end of the activity), W is the name of the workflow, P is the unique identifier of the case, A is the name of the activity and O is the progressive number of occurrence of the activity in the process execution.

It is important to observe that in this event log the timestamp is, in fact, just a progressive number which identifies each event within a case. Hence, no actual information about the date and time of the events are present. However, this is not an issue since the order of the events can be determined based on their position in the event log.

Furthermore, it has been observed that the name of the workflow is always *monday* in all the rows. This is expected as the event log refers to the daily routine on Mondays.

2.1.2 Conversion to CSV format

In order to import the event log in ProM, it has been converted in CSV format. This has been done with a Python script leveraging the Pandas library.

First of all, each row of the plain text file has been added to a DataFrame and, then, a name has been assigned to each column in order to better identify them.

Consequently, an initial cleaning of the event log has been performed. Specifically, as mentioned in the previous section, the name of the workflow is the same for each event. Hence, no useful information are provided by this value. For this reason, it has been chosen to remove the *workflow* column from the DataFrame. Similarly, the *timestamp* column has been dropped as no actual information of the date and time of the events are available.

In addition, the rows with an event type equal to *begin_ of_ process* and *end_ of_ process* have been removed, and the event types *begin_ of_ activity* and *end_ of_ activity* have been renamed, respectively, to *start* and *complete*. The motivations for these two operations will be explained in the following section when describing the conversion to XES format.

Finally, the DataFrame has been saved in a CSV file, an excerpt of which is shown in Figure 6.

```
case_id,event_type,activity,number
monday6637,start,sit_down,1
monday6637,start,eat_meal,1
monday6637,complete,eat_meal,1
monday6637,complete,sit_down,1
monday6637,start,read,1
```

Figure 6: CSV event log excerpt

2.1.3 Conversion to XES format

The CSV event log has been imported in ProM and, then, converted in XES format by using the plug-in «Convert CSV to XES». As already mentioned in Section 1.5.1, the XES format has been chosen as it is the standard exchange format for process mining.

Considering the four columns in the CSV file, each of them can be mapped directly to a XES attribute. Specifically, the *case_ id* column is mapped to the *concept:name* attribute of the trace, the *event_ type* column is mapped to the *lifecycle:transition* attribute, the *activity* column is mapped to the *concept:name* attribute of the event and the *number* column is mapped to the *concept:instance* attribute.

Note that this explains why, in the previous section, it has been chosen to remove the begin/end of process events and to rename the begin/end of activity. In particular, the lifecycle extension does not support the expression of begin/end of process events, while the begin/end of activity are referred, respectively, as *start* and *complete*.

The event log in XES format obtained in this step has been used in all the following phases of the process mining activity.

2.2 Event log exploration

Before proceeding with the execution of process discovery algorithms, the event log has been explored in order to better understand the nature of the data. This has been done by leveraging the visualization tools provided by ProM.

2.2.1 An example of a case

To illustrate the general case structure, an example of case made up of 14 events is shown in Figure 7. In particular, the presented case refers to a day where the user started his/her routine by sitting down while eating meal and, then, he/she started writing. After getting up, he/she brushed hair, drank and read.

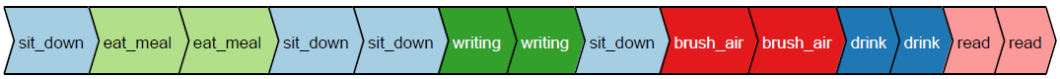


Figure 7: Example of a case

Note that each activity in the case appear at least twice. This is due to the fact that the activities have a start event and a completion event. For instance, the first *sit_down* event refers to the start of the *sit_down* activity, while the second is the completion.

Moreover, the start and completion events of a specific activity may be interleaved with the start and completion events of other activities. This happens when multiple activities are performed concurrently. For instance, the initial *sit_down* start event is followed by the *eat_meal* start event. Then, there is the *eat_meal* complete event followed by the *sit_down* complete event. Hence, the two activities were performed in parallel, that is the user ate while sitting down.

Finally, if on one hand each activity appears in the case at least twice, on the other hand it may appear also more than twice. This happens when an activity is repeated multiple times throughout the case. An example is, again, the *sit_down* activity which is performed one time during the *eat_meal* activity and a second time during the *writing* activity.

2.2.2 Exploration results

The event log is made up of 463 events broken down in 30 cases. Thus, as the number of cases is fairly low, this event log has a moderate level of incompleteness meaning that not so much examples of the routine are available for the algorithms to discover the user behavior.

Considering the cases in more details, it can be observed that each case contains an average of 15 events with the shortest case containing 10 events and the longest one 20 events.

The different activities carried out by the user under consideration during the daily routine on Monday are seven, namely: *brush_air*, *drink*, *eat_meal*, *play_phone*, *read*, *sit_down* and *writing*.

Some of the activities take place in all the cases (e.g., *sit_down*), while others are carried out in just a handful of cases (e.g., *play_phone*). In more detail, Figure 8 shows a dotted chart with the 7 activities on the x-axis and the 30 cases on the y-axis. A pink dot in the chart represents a specific event and, thus, it shows that an activity has been performed in a case.

Looking at the chart it is possible to observe that the activities *eat_meal*, *sit_down*, *writing* and *drink* are performed in all the cases (in fact, *drink* is performed in all the cases but two). On the other hand, the *brush_air*, *play_phone* and *read* activities are carried out in less cases, with *play_phone* performed in about half of the cases. Note that, being *play_phone* a fairly rare activity, it can be considered a source of noise in the log which contains activities not happening very often.

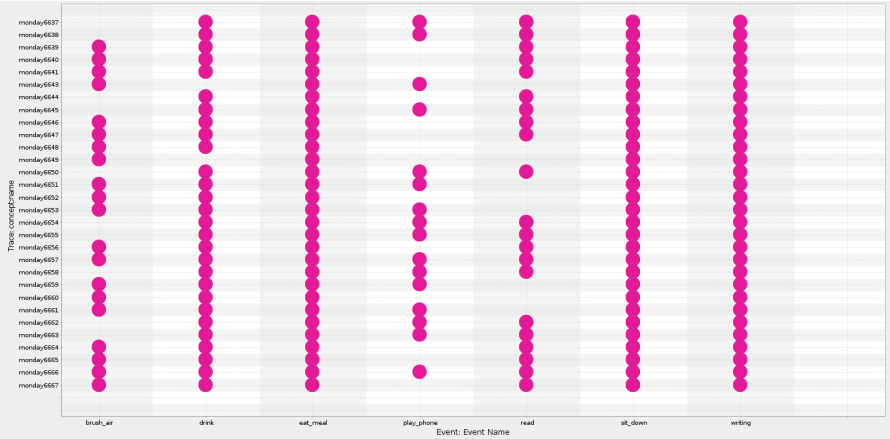


Figure 8: Dotted chart activities by cases

The cases in the event log have a total of 19 variants (i.e., cases which differs for number and sequence of activities). The most common variant is the one shown in Figure 7 which is the sequence of activities performed in 4 cases. The remaining cases are distributed in other variants with most of the variants performed in just a single case.

Considering the initial and final activities for each of the cases, it can be observed that in almost all the cases (precisely, 27 cases) the daily routine started with the *sit_down* activity, while in just a few cases it started with *drink* and *read* (respectively, 2 and 1 cases). Similarly, in 10 cases the routine ended with a *sit_down* activity, in 9 cases with a *read* activity, in 6 cases with *drink* and in 5 cases with *brush_air*.

Finally, given that no timestamp is present, anything can be said regarding the duration of each activity.

2.3 Methodology used to discover and evaluate the models

Following the exploration of the event log, the methodology to run the process discovery algorithms and to evaluate the results has been defined. In particular, the process discovery algorithms to use have been selected, three versions of the event log have been prepared and the quality metrics to evaluate the models have been chosen.

The following sections report each of the steps in more details.

2.3.1 Selection of the process discovery algorithms

As already mentioned, there are multiple techniques that can be used for process discovery. In this case study, it has been chosen to use three classes of algorithms: alpha miner, inductive miner and genetic miner (a general overview of these algorithms is given in Section 1.4).

In particular, despite the alpha miner is not able to handle noise and incompleteness well, it is still interesting to check the results obtained by this simple algorithm when applied to this event log which contains concurrent activities and a moderate level of incompleteness. Hence, it has been chosen to run three variants of this algorithm: original α-algorithm, α^{+}-algorithm and α^{++}-algorithm.

Regarding the inductive miner, as already said, this is one of the leading algorithms for process discovery and, with its variants, it is able to handle noise and incompleteness in the log. For this case study, it has been chosen to use this algorithm in five variants: basic inductive miner (IM), inductive miner–infrequent (IMf), inductive miner–incompleteness (IMc), inductive miner–life-cycle (IMlc) and inductive miner–infrequent & life-cycle (IMflc).

Finally, the genetic miner has been chosen since it is a less «conventional» algorithm and it is able to deal with noise and incompleteness, and to balance the four quality criteria.

It is important to point out that, apart from the IMlc and the IMflc algorithms, all other selected algorithms are not able to directly exploit the event type information associated to the events in the log. Thus, they consider just the atomic events and the activity names without taking into account the event type. However, these algorithms are still able to discover concurrent activities (for instance by using the directly-follows relations) even without using the event type attribute.

All the selected algorithms are already implemented in ProM and, thus, they can be directly applied to the event log. In particular, the alpha miner and its variants are implemented by the «Alpha Miner» plug-in, the inductive miner and its variants by the «Mine using Inductive Miner» plug-in and the genetic miner by the «Mine a Process Tree with ETMd» plug-in. Note that the latter produces a process tree that can be converted to a Petri net by using the «Convert Process Tree to Petri Net» plug-in.

2.3.2 Preparation of the event logs

It has been chosen to apply the selected process discovery algorithms to three different versions of the event log to compare the models obtained by filtering in multiple ways the original event log.

In particular, the first version is the initial event log without any filtering or additional preparation steps. This allows to check the abilities of the algorithms to discover a high quality model when applied directly to the available events in the log without any preprocessing.

The second version is a filtered event log where all the start events have been removed, leaving only the completion events. The motivation is that, as mentioned in the previous section, the selected discovery algorithms (besides the IMlc and IMflc algorithms) are not able to distinguish between start and complete events. Hence, having these events in the log might mislead the algorithms and produce less accurate models. More specifically, since just the event name is used to identify each event, the event type data are not visible to the algorithms. As a consequence, two events with the same event name, but different types are seen as the same event by the algorithms, which can be misleading. The filtered event log has been generated with the «Filter Log using Simple Heuristics» ProM plug-in which allows to remove part of the events in the log based on their name, event type and/or occurrence frequency. In this case, the plug-in has been used to remove all the start events. The filtered log contains 30 cases with 232 events, which is exactly half of the initial event log. This is expected since for each start event there is a corresponding complete event.

Finally, the third version is a filtered event log where all the start events have been removed and the less frequent activities which are carried out in less than 90% of the cases in the log have been removed as well. Hence, this third event log do not contain the infrequent activities that are rarely carried out during the daily routine. Also in this case, the filtered log has been obtained by using the same plug-in mentioned in the previous paragraph. The only difference is that, here, in addition to the removal of all the start events, a threshold of 90 has been set in the «Event filter» section of the plug-in to retain just the activities which are performed in at least 90% of the cases, removing the remaining ones. Clearly, the log obtained is smaller that the second version and it contains 30 cases with 216 events (16 less than the second event log version). The events removed are all the 16 executions of the *play_phone* activity that is carried out in about half of the cases. Hence, the only activity removed is *play_phone*.

2.3.3 Evaluation methodology

Before running the algorithms on the event logs, the quality metrics for the evaluation of the produced models have been chosen. In particular, it has been chosen to calculate, for each model, the values of the four metrics presented in Section 1.3, that is fitness, precision, generalization and simplicity. To do this, four ProM plug-ins have been used, namely: «Replay a Log on Petri Net for Conformance Analysis», «Check Precision based on Align-ETConformance», «Measure Precision/Generalization» and «Show Petri-net Metrics».

More specifically, the first three plug-ins compute fitness, precision and generalization by using an approach based on the alignments of the events in the log with the model and vice versa. The alignments are needed to determine how the event log can be replayed on the model. Hence, after automatically aligning the log with the model, the plug-ins compute the values of the three metrics.

Note that, as mentioned in Section 1.3.1, to calculate the fitness value multiple costs have to be set. For this case study, the default values provided by the «Replay a Log on Petri Net for Conformance Analysis» plug-in have been used which assign a penalty with cost 1 to all the move types.

Regarding the simplicity, the plug-in «Show Petri-net Metrics» is able to measure the simplicity of a model exploiting multiple metrics. For this case study, the extended Cardoso metric described in Section 1.3.4 has been used.

Finally, it is important to mention that the measurement of the four metrics requires a sound WF-net to produce correct results. Hence, before running the plug-ins for estimating the value of the metrics, another ProM plug-in has been used to analyze the Petri net and to check for soundness, that is «Analyze with Woflan».

2.4 Results

After defining the methodology to discover and evaluate the models, the selected algorithms and metrics have been executed to discover and evaluate a total of 34 models on the three event logs.

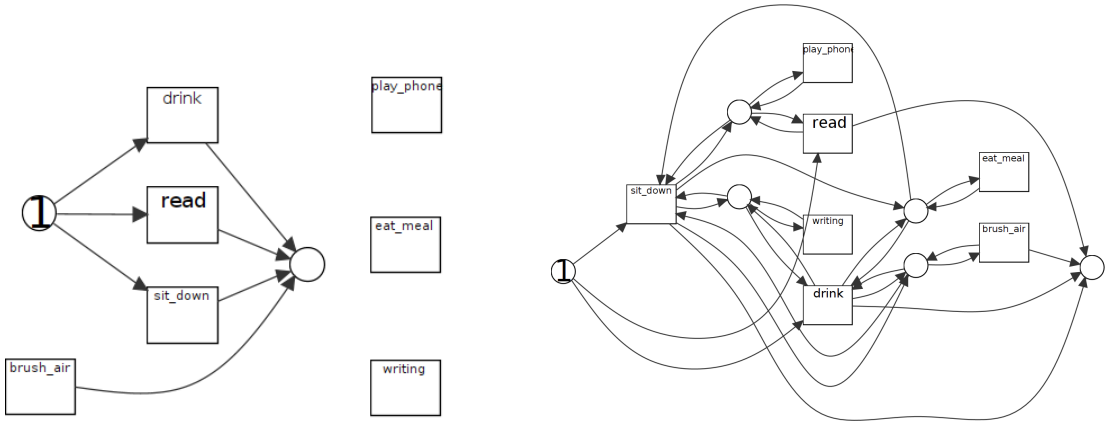
An overview of the metrics values for each discovered model is shown in Table 1 and, then, the results obtained are presented and commented in the following sections.

2.4.1 Alpha miner

The alpha miner has been the worst performing algorithm of the three. This was somehow expected since, as already mentioned, this algorithm has been one of the first to handle concurrent activities but, at the same time, it is also limited and not able to deal well with the noise and incompleteness which are present in the event log considered in this case study.

Event Log	Algorithm	Fitness	Precision	Generalization	Simplicity
Original	α	n.a.	n.a.	n.a.	n.a.
	α^+	n.a.	n.a.	n.a.	n.a.
	α^{++}	n.a.	n.a.	n.a.	n.a.
No start events	α	n.a.	n.a.	n.a.	n.a.
	α^+	n.a.	n.a.	n.a.	n.a.
	α^{++}	n.a.	n.a.	n.a.	n.a.
No start and infrequent events	α	n.a.	n.a.	n.a.	n.a.
	α^+	n.a.	n.a.	n.a.	n.a.
	α^{++}	n.a.	n.a.	n.a.	n.a.
Original	IM	1.00	0.191	n.a.	35.0
	IMf (0.2)	0.816	0.267	0.797	21.0
	IMf (0.1)	0.995	0.207	n.a.	32.0
	IMf (0.3)	0.671	0.336	0.949	12.0
	IMc	1.00	0.218	n.a.	33.0
	IMlc	0.821	0.222	0.71	25.0
	IMflc (0.2)	0.712	0.25	0.847	16.0
	IMflc (0.1)	0.712	0.254	0.833	16.0
	IMflc (0.3)	0.516	0.376	0.631	15.0
No start events	IM	1.00	0.247	0.638	24.0
	IMf (0.2)	0.965	0.306	0.771	16.0
	IMf (0.1)	0.965	0.298	0.777	16.0
	IMf (0.3)	0.826	0.4	0.639	13.0
	IMc	0.996	0.316	0.796	20.0
No start and infrequent events	IM	0.952	0.278	0.632	22.0
	IMf (0.2)	0.919	0.409	0.822	13.0
	IMf (0.1)	0.919	0.403	0.817	13.0
	IMf (0.3)	0.782	0.591	0.766	10.0
	IMc	0.952	0.313	0.702	20.0
Original	Genetic (50)	0.834	0.897	0.99	34.0
	Genetic (500)	0.86	0.814	0.987	38.0
No start events	Genetic (50)	0.889	0.79	0.976	37.0
	Genetic (500)	0.852	0.987	0.993	18.0
No start and infrequent events	Genetic (50)	0.887	0.833	0.986	23.0
	Genetic (500)	0.935	0.755	0.910	41.0

Table 1: Overview of the metrics values of the discovered models



(a) α-algorithm and α^+-algorithm (b) α^{++}-algorithm

Figure 9: Petri nets discovered on the event log (original) by the alpha miner

More specifically, most of the models discovered with this algorithm are not even WF-nets as all of them but two have unconnected nodes. Furthermore, the two models which are WF-nets are not sound as they contain dead transitions which cannot be enabled.

The following paragraphs show in more detail the results obtained on the three event logs with this algorithm. Nevertheless, it is important to point out that no metric has been measured on the models discovered with this algorithm (and its variants) because they are not WF-nets (or they are unsound) and, hence, the plug-ins used to compute the metrics are not able to give a proper evaluation (as already mentioned in Section 2.3.3).

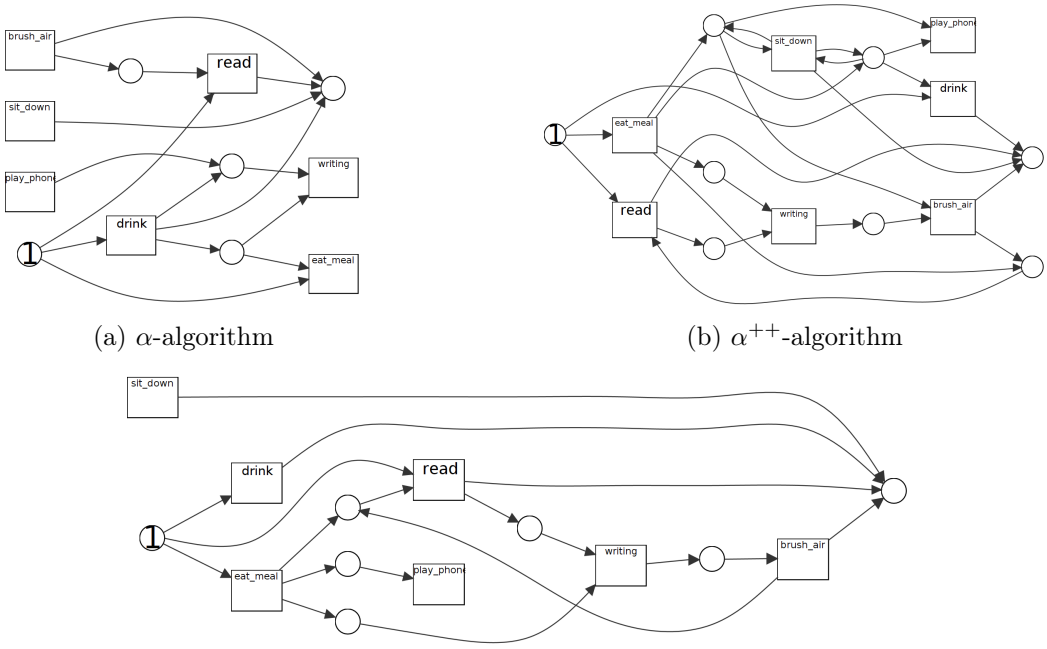
Original event log

The models obtained on this event log are not WF-nets with the exception of one. In particular, the models discovered with the original α-algorithm and with the α^+-algorithm contain unconnected transitions which are not allowed in a WF-net. This is clearly visible in Figure 9a where the Petri net obtained with these two algorithms is shown (the two algorithms produced exactly the same Petri net).

Instead, the model obtained with the α^{++}-algorithm is a WF-net, but it is not sound as it contains transitions which cannot be fired starting from the initial marking. This model is shown in Figure 9b and it can be observed that, in fact, none of the transitions can be fired starting from the initial marking. Hence, according to this model no action can be ever performed.

Filtered event log without start events

On this event log, all three variants of the algorithm discovered models which are not WF-nets as they contain unconnected nodes. This is visible in Figure 10 where the three models discovered are shown and there are transitions without any input place (e.g., *sit_ down*).



(a) α-algorithm

(b) α^{++}-algorithm

(c) α^{+}-algorithm

Figure 10: Petri nets discovered on the event log (no start events) by the alpha miner

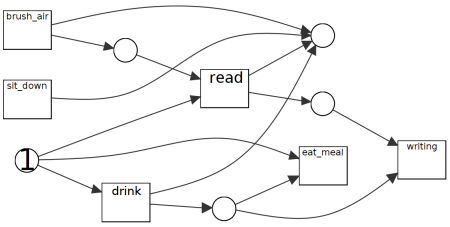
Note that this is a bit surprising given that removing the start events should streamline the cases in the log and provide the algorithms a better chance of capturing the behavior of the process. However, for this event log this is not true and the models are not structurally correct.

Filtered event log without start events and infrequent events

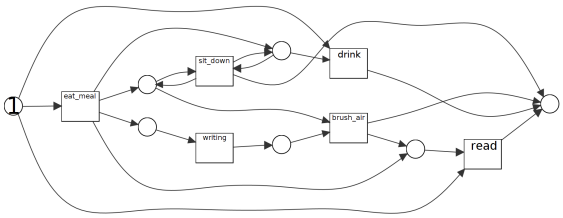
Finally, the results obtained on this event log are similar to the ones obtained on the original event log. Hence, also in this case, the models discovered by the α-algorithm and α^{+}-algorithm are not WF-nets, while the one discovered by the α^{++}-algorithm is a WF-net, but is not sound as it is unbounded. These models are illustrated in Figure 11.

2.4.2 Inductive miner

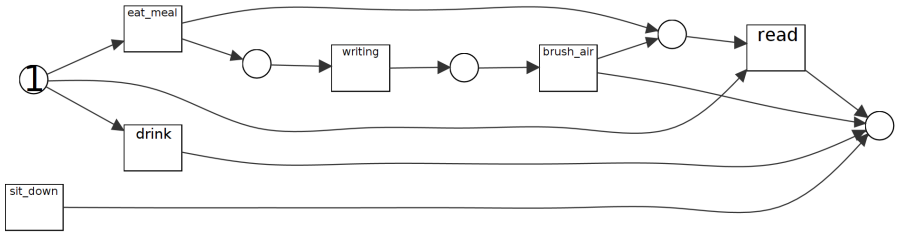
The quality of the models discovered by the inductive miner is average. This is in contrast with the expectations as the inductive miner is one of the main algorithms in process discovery. However, the results are still a significant improvement with respect to the models obtained with the alpha miner.



(a) α^{+}



(b) α^{++}-algorithm



(c) α^{+}-algorithm

Figure 11: Petri nets discovered on the event log (no start and infrequent events) by the alpha miner

In particular, the discovered models are all sound WF-nets (this is expected as this algorithm produces process trees which are sound by construction) and have high fitness values (in some cases, perfect fitness), but this is at the expense of precision which is poor. This implies that these models are underfitted and allow a lot of behavior which is not present in the log.

The following paragraphs report in more detail the results obtained on the three event logs.

Original event log

On this event log, the basic inductive miner (IM) scored a perfect fitness as expected. However, this model is the one with the worst precision (0.19) among all the discovered models. This means that the model is significantly underfitted which is confirmed by the WF-net shown in Figure 12a. As it can be seen, this Petri net has a structure that resembles the «enumerating model». This is further confirmed by a simplicity value of 35, which is the highest among the models discovered by this algorithm. In other words, this model is very complex, has perfect fitness, but low precision just as the «enumerating model». Unfortunately, the plug-in for computing the generalization value got stuck and did not complete the computation after a long running time. Hence, the value for the generalization metric is not available.

The model discovered by the inductive miner–infrequent (IMf) algorithm, which is shown in Figure 13b, achieved more balanced results than the IM algorithm. Specifically, a threshold of 0.2 has been used to filter out the less frequent relations and, clearly, this reduced the fitness reaching a value of 0.82, but it improved the precision which is 0.27. The generalization value is 0.8, while the simplicity is 21. Hence, overall, this model is better than the one obtained with the basic inductive miner, but it is still over-generalized and allows more behavior than the one present in the log. It is worth adding that two additional models have been discovered with this algorithm with a threshold of 0.1 and 0.3 (these are shown in Figure 13a and 13c). In the first case, the lower threshold improved the fitness value (0.995), but decreased the precision (0.21) while making the model more complex (the simplicity of this model is 32). Hence, no meaningful benefits besides the fitness have been obtained by using this alternative threshold. In the second model instead, a fitness value of 0.67 has been reached with a precision of 0.34, a generalization of 0.95 and a simplicity value of just 12. Thus, by increasing the threshold the model gets very simple (as visible in Figure 13c) and becomes unable to replay the log well.

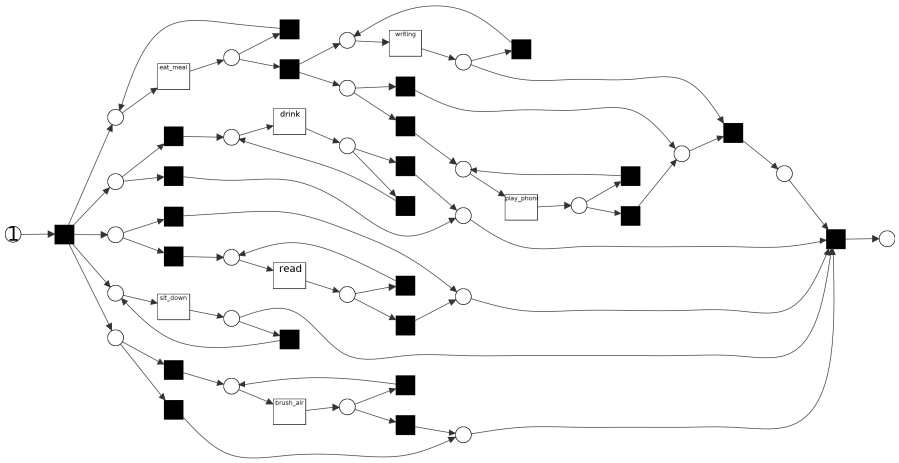
Finally, the results obtained with the inductive miner–incompleteness (IMc) variant are shown in Figure 12b and they are very similar to the ones obtained with the IM algorithm. Thus, the fitness is perfect, but the precision is low (0.22), with a high simplicity value of 33. Also in this case, it has not been possible to measure the generalization.

Original event log (with event type information)

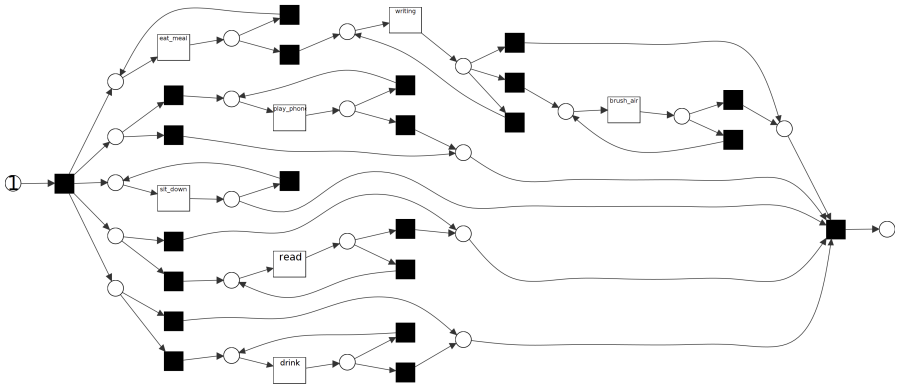
To effectively leverage the event type information attached to the events, the infrequent miner–life-cycle (IMlc) and the inductive miner–infrequent & life-cycle (IMflc) algorithms have been executed (the latter with a threshold of 0.2).

The discovered models did not perform much better than the ones obtained with the other variants. In particular, the model discovered by the IMlc algorithm (shown in Figure 14a) scored a fitness value of 0.82, a precision value of 0.22 and a generalization of 0.71. Similarly, the IMflc obtained a model (shown in Figure 14b) with a fitness value of 0.71, a precision of 0.25 and a generalization of 0.85. Furthermore, the simplicity is 25 for the IMlc model and 16 for the IMflc model.

Note how both these models are not very precise and too general meaning that they allow for the concurrent execution of most of the activities. This is clearly visible in the model discovered by the IMflc algorithm (Figure 14b) which can be considered similar to the «flower model». In particular, after the initial silent transition fires, it creates a marking where all the transitions are enabled. Hence, according to this model, the user might read, brush air, play phone, drink, eat meal, write and sit down all at the same time.

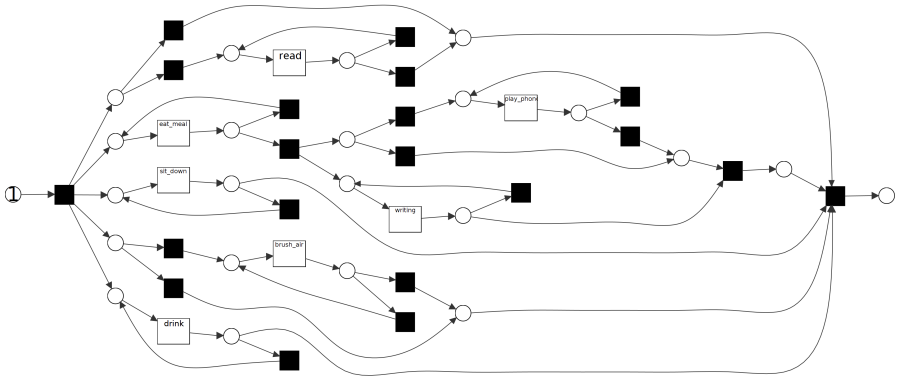


(a) IM

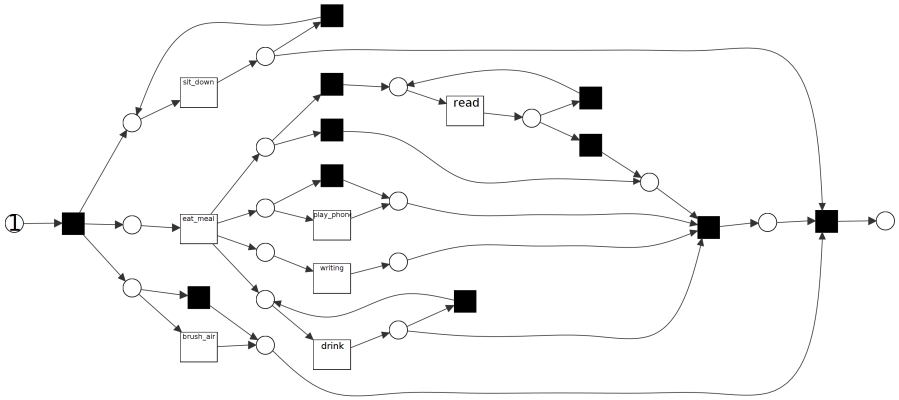


(b) IMc

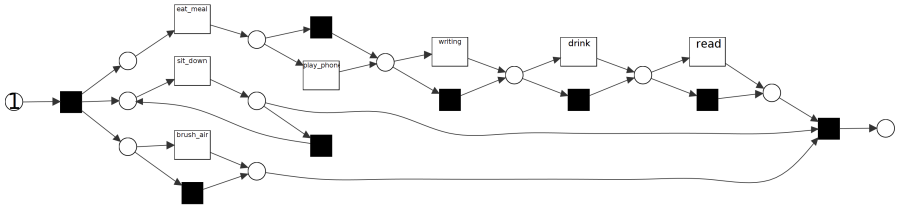
Figure 12: Petri nets discovered on the event log (original) by the inductive miner



(a) IMf (threshold 0.1)



(b) IMf (threshold 0.2)



(c) IMf (threshold 0.3)

Figure 13: Petri nets discovered on the event log (original) by the inductive miner

For the sake of completeness, it is worth mentioning that the IMflc algorithm has been also run with a threshold of 0.1 and 0.3. The first model scored almost exactly the same results of the model obtained with a threshold of 0.2. The latter instead, has the lowest fitness among the models discovered (0.52). However, it improves the precision (0.38) while reducing the generalization (0.63).

Thus, the use of event types did not lead to a significant improvement with respect to the results obtained on the original event log by the IM, IMf and IMc algorithms. In fact, overall, the models discovered by the IMlc and IMflc have a lower fitness than the models discovered by the IM, IMf and IMc algorithms with similar precision values.

Filtered event log without start events

Overall, the removal of the start events from the log allowed to improve the quality of the discovered models with respect to the original event log.

More specifically, the IM algorithm produced a model (Figure 15a) with perfect fitness, a precision of 0.25 and a simplicity of 24. However, the generalization value is 0.64 which is the lowest obtained by all the algorithms. Hence, this model is underfitted and, in addition, it is the least general of the discovered models. Nevertheless, it still is an improvement with respect to the model discovered by the IM algorithm on the original log, especially in the precision value.

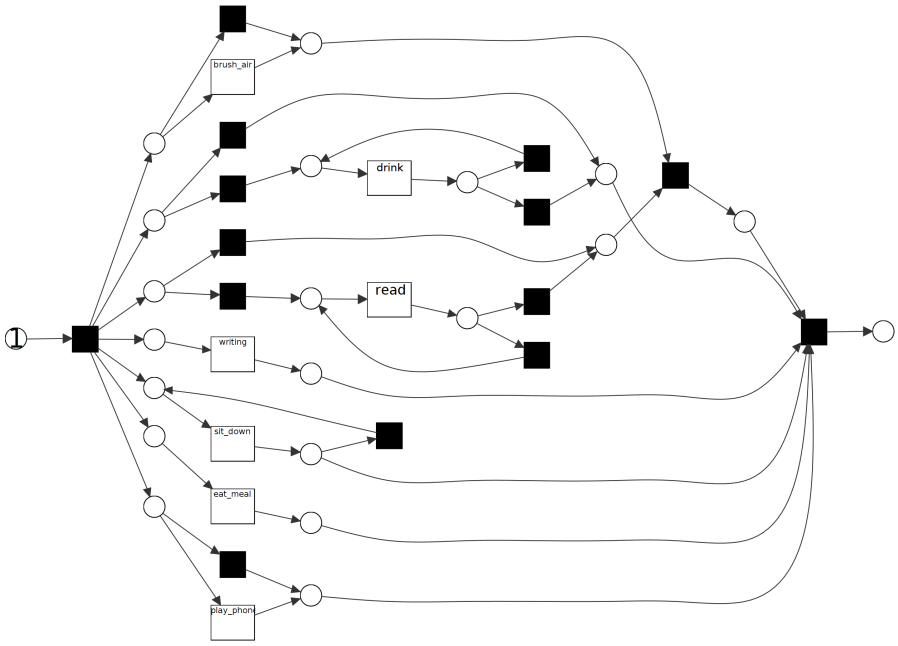
The IMf algorithm has been executed with a threshold of 0.2 and produced a very simple model shown in Figure 15c. This model has a simplicity of just 16, but with a high fitness (0.96). However, the precision value is 0.31 and the generalization 0.77. Hence, the model is simple, but manages to capture the behavior of the process in a better way than the model obtained with this algorithm on the original event log. Furthermore, two more models have been discovered with a threshold of 0.1 and 0.3. The first model is equal to the one obtained with a threshold of 0.2, while the latter (shown in Figure 15d) reduced the fitness (0.83), but increased the precision (0.4, which is the best obtained by this miner on this version of the log). The generalization is of 0.64, while the simplicity is 13. Note that, also in this case, the model allows for the concurrent execution of all activities at once.

Finally, the IMc algorithm obtained a model (Figure 15b) with an almost perfect fitness value (0.99), a precision of 0.32, a generalization of 0.8 and a simplicity of 20. Thus, this algorithm is able to produce a balanced model which achieve a high fitness, while not decreasing the precision too much or becoming too complex (this is in contrast with the results obtained with IMc miner on the original event log).

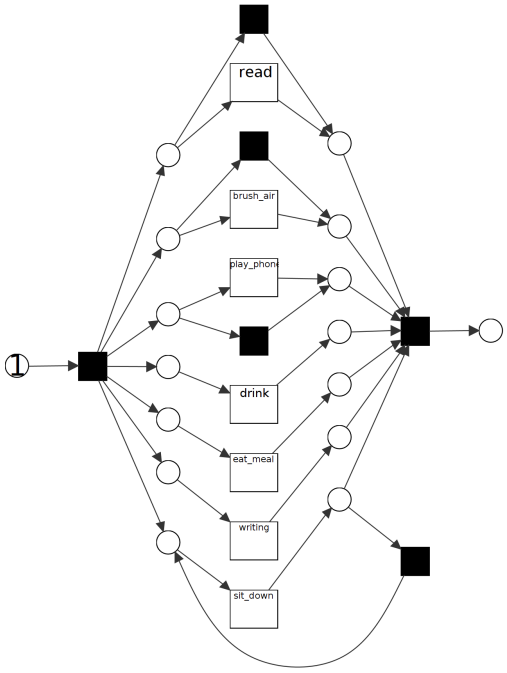
Therefore, the inductive miner and its variants are mislead by the start and complete events in the original event log and benefit from removing the start events.

Filtered event log without start events and infrequent events

The results obtained with this event log are a further improvement with respect to the previous logs, especially on the precision.

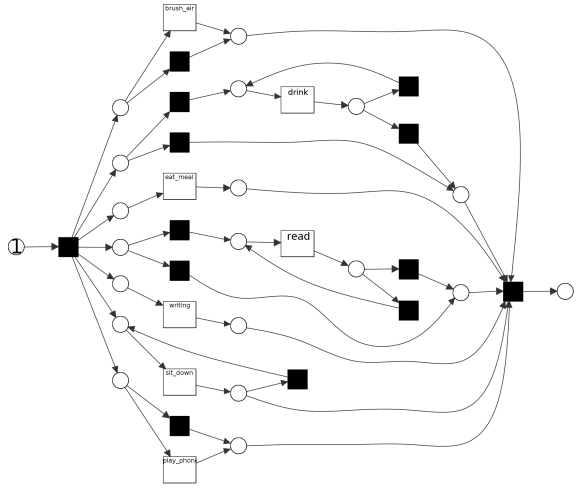


(a) IMlc

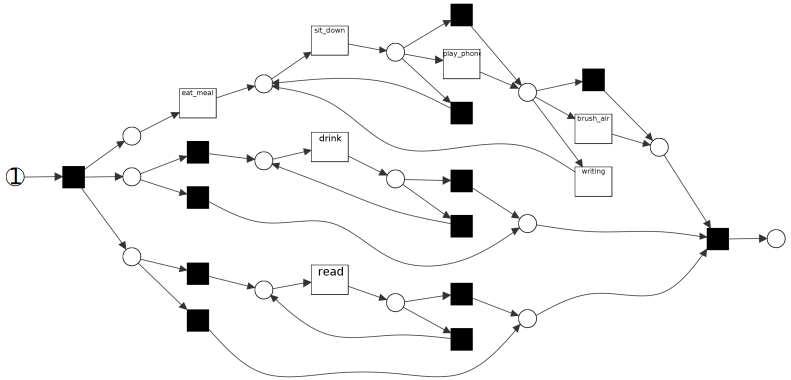


(b) IMflc

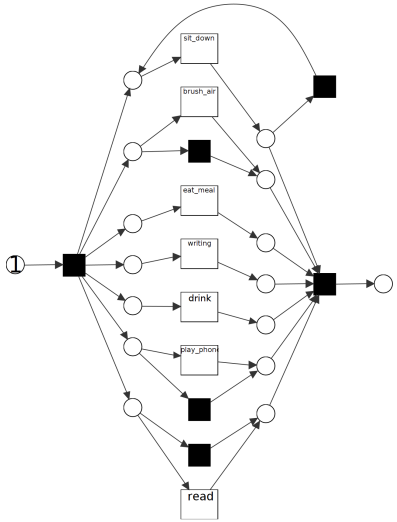
Figure 14: Petri nets discovered on the event log (original) by the inductive miner with event type information



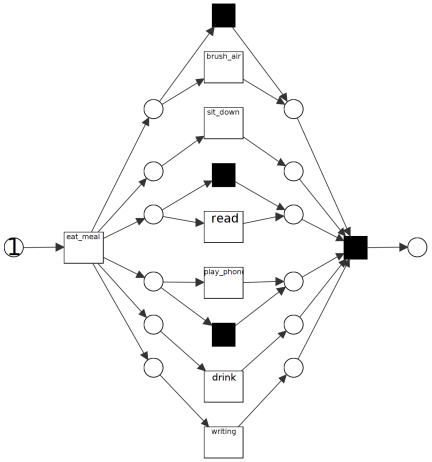
(a) IM



(b) IMc



(c) IMf (threshold 0.2)



(d) IMf (threshold 0.3)

Figure 15: Petri nets discovered on the event log (no start events) by the inductive miner

Before reporting the results, it is important to point out that the evaluation has been performed including the infrequent events. This decision has been made in order to evaluate the models by considering also the infrequent activities that have been not taken into account during the discovery, but that might still happen in a real case. In other words, this allows to check if, despite having removed the infrequent events, the discovered models are a good representation of the daily routine on Mondays.

The model obtained with the IM algorithm is shown in Figure 16a. It has a fitness value of 0.95 (note that the fitness is not perfect because the event log used for the evaluation includes the infrequent events as well), a precision of 0.28, a generalization of 0.63 and a simplicity of 22. Thus, this model is similar to the one discovered with the IM algorithm on the filtered event log without start events, but it improves the precision.

The IMf algorithm with a threshold of 0.2 achieved a precision of 0.41, with a high fitness value of 0.92, a generalization of 0.82 and a simplicity of 13. The discovered model is visible in Figure 16c. It can be seen that this is a very simple model which is also able to have high fitness and generalization values and an average precision value. Using a threshold of 0.1 resulted in a model with essentially the same values obtained by using a threshold of 0.2. Furthermore, by using a threshold of 0.3, the fitness value dropped to 0.78, while the precision increased to 0.6 (the best achieved by the inductive miner), with a generalization of 0.77 and a simplicity of 10 (this is the simplest model discovered by all the algorithms). This model is shown in Figure 16d.

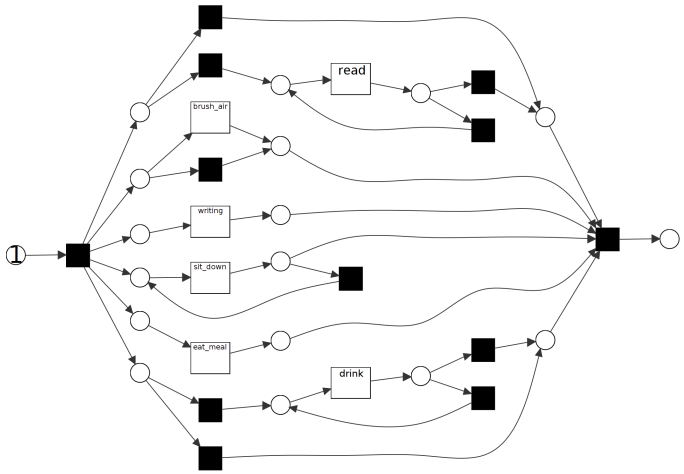
Finally, the result obtained by the IMc algorithm are shown in Figure 16b. This model has a fitness of 0.95, a precision of 0.31 with a generalization of 0.70 and a simplicity of 20. This model is similar to the one obtained by this algorithm on the log without start events, with the exception of the generalization which here is lower.

2.4.3 Genetic miner

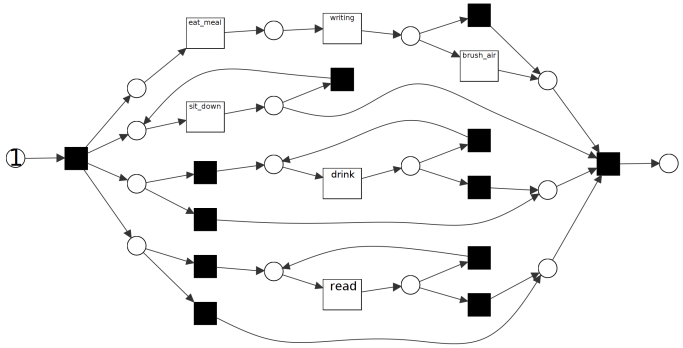
The genetic miner has been the best performing algorithm of the three and produced the models with the highest values of fitness, precision and generalization.

In particular, the models are all sound WF-nets (also in this case, this is not surprising as the algorithm produces process trees which are sound by construction) and have high fitness values as well as high precision values. However, the discovered models are, on average, more complex than the ones produced by the other two miners.

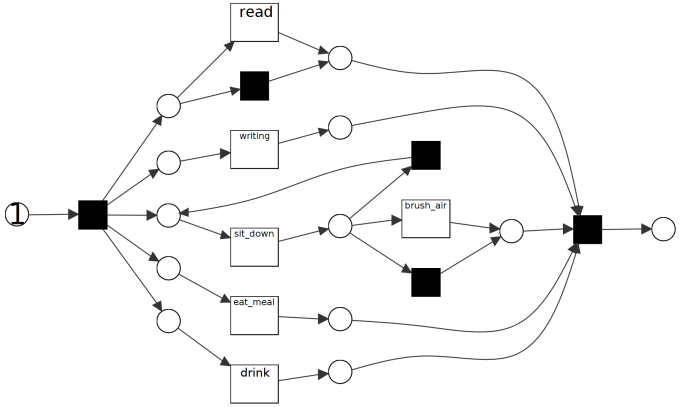
Before showing in more details the results obtained on the three event logs, it is worth mentioning that the process trees discovered with this algorithm have been converted to a Petri net and, then, evaluated. Furthermore, the number of iterations executed for the genetic algorithm has been set to 50 and 500. Hence, the algorithm has been run two times. In the first, it stopped after 50 iterations, while in the latter after 500.



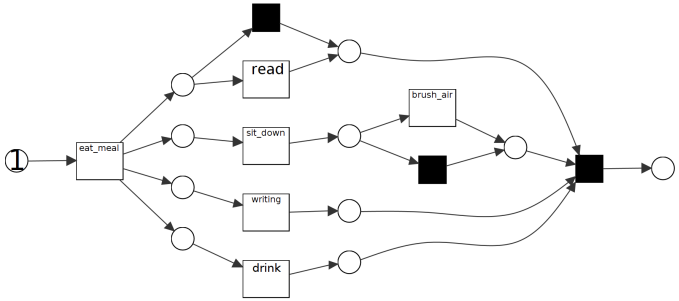
(a) IM



(b) IMc



(c) IMf (threshold 0.2)



(d) IMf (threshold 0.3)

Figure 16: Petri nets discovered on the event log (no start and infrequent events) by the inductive miner

Original event log

The model discovered on the original event log with 50 iterations (Figure 17a) has a fitness value of 0.83. This value is lower than the fitness values of the models discovered with the inductive miner. Nevertheless, the precision value is much higher (0.9) meaning that the model is not underfitted. The generalization value is 0.99 and, thus, the model does not allow for much behavior different than the one in the log, while still being general. However, the simplicity value is 34 which denotes a fairly complex model.

The model discovered with 500 iterations (Figure 18a) has a fitness of 0.85 which is just a minor improvement with respect to the previous model. However, the precision is almost perfect (0.99) as the generalization (0.99). Furthermore, this model is much more simple than the previous one with a simplicity of 18. Thus, not surprisingly, by increasing the number of iterations a better model can be discovered.

In addition, note that both models are mainly based on the sequential execution of the activities in the process. For instance, the model shown in Figure 17a requires that the daily routine starts with the *sit_ down* activity, then the *eat_ meal* activity is performed and, following, the *drink* activity is carried out and so on. However, this is not correct as these activities are usually performed in parallel. For instance, the user typically eats while sitting down. Hence, this algorithm seems to not be able to correctly capture the parallelism among multiple activities.

Moreover, these models contain multiple transitions with the same names. An explanation for this phenomenon might be related to the fact that the algorithm considers only the names of the events and, hence, it is not able to distinguish between start and complete events. More specifically, it seems to be able to understand that the events are distinguished, but it is not able to give two different names to the events. An alternative explanation might be related to the conversion of the process tree discovered by the algorithm to a Petri net during which some structures of the tree have been converted to nodes with the same name.

Finally, the model discovered with 50 iterations include a overly-complicated structure at the end which behavior is not very clear. This explains why this model has a higher value for the simplicity metric with respect to the models discovered with other algorithms. Again, this might be in part related to the conversion of the process tree to a Petri net.

Thus, if on one hand these models have high precision and generalization, they are complex and they are not able to score a perfect fitness, meaning that some cases cannot be replayed.

Filtered event log without start events

On this event log the genetic miner with 50 iterations managed to discover a model (Figure 17b) with a fitness value of 0.86, a precision value of 0.81, a generalization value of 0.99 and a simplicity value of 38. Hence, in this model, the fitness is slightly higher than the one obtained on the original event log, but the precision is lower. However, overall, the values of the metrics obtained of this model are similar to the values of the metrics of the model discovered on the original event log. This implies that, for this miner, there is no meaningful difference in removing the start events.

Finally, in this case increasing the number of iterations to 500 did not produce a better model. More specifically, the metrics values for this model (Figure 18b) are almost equal to the ones of the the model obtained with 50 iterations with minor improvements in fitness.

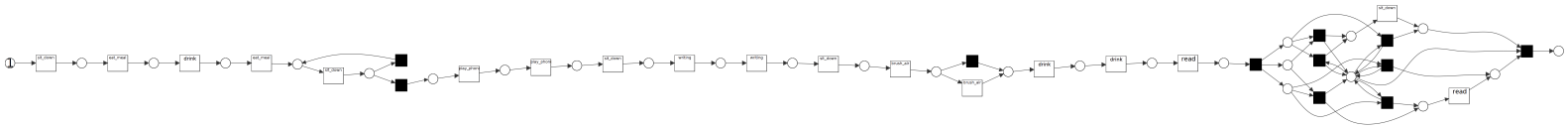
Note that, also in this case, there are two complex structures at the start and end of the model and the part between them follows a sequential structure. Furthermore, there are multiple transitions associated to the *sit_ down* activity. Hence, the considerations given for the previous event log also apply here.

Filtered event log without start events and infrequent events

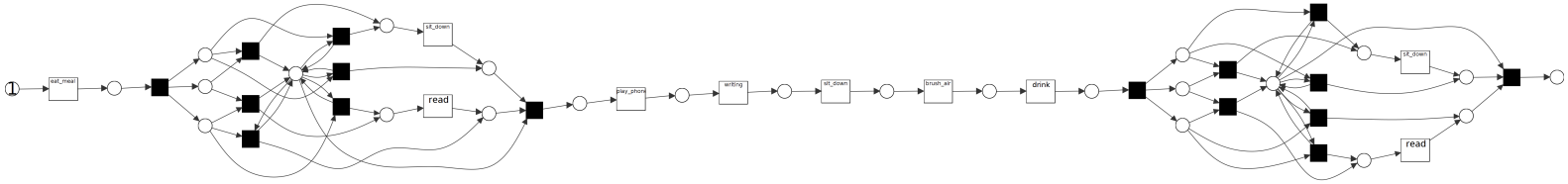
The model obtained on this log with 50 iterations, shown in Figure 17c, have a fitness value of 0.89, a precision value of 0.83, a generalization of 0.99 and a simplicity of 23. Note that this model is simpler than the ones discovered on the original event log and on the event log without start events, but it manages to keep similar values for fitness, precision and generalization. Therefore, the removal of start events and infrequent activities allowed this miner to discover an overall better model.

Finally, the model obtained by running 500 iterations is shown in Figure 18c. It has a high fitness of 0.93 (the highest obtained by this miner), but the precision (0.75) and the generalization (0.91) are the lowest among the results achieved by this miner. Furthermore, the simplicity is 41 and, from the figure, two complex structures can be observed at the beginning and end of the model. Hence, also in this case, increasing the number of the iterations did not allow the algorithm to discover a better model. In fact, this is the most complex model discovered by all the algorithms on all the event logs.

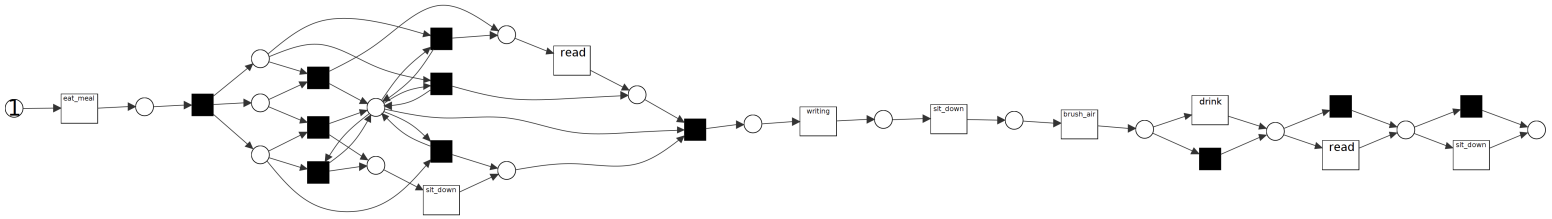
Note that, also for this algorithm, the evaluation of the results obtained on this event log has been carried out including the infrequent events.



(a) Original event log

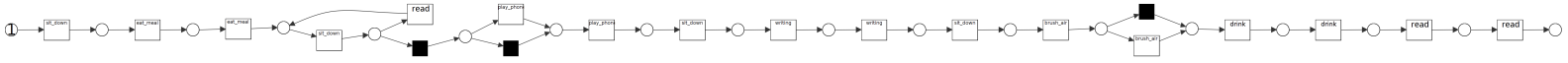


(b) Event log without start events

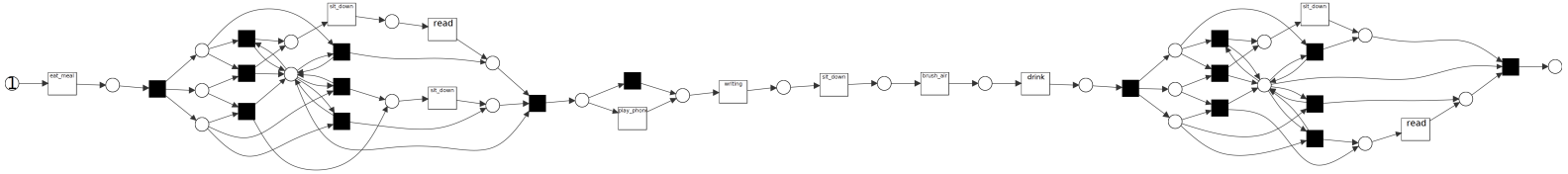


(c) Event log without start and infrequent events

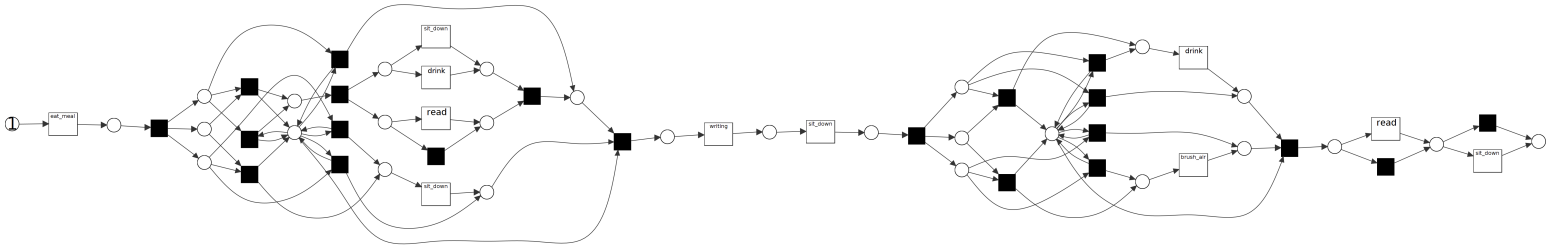
Figure 17: Petri nets discovered by the genetic miner with 50 iterations



(a) Original event log



(b) Event log without start events



(c) Event log without start and infrequent events

Figure 18: Petri nets discovered by the genetic miner with 500 iterations

2.4.4 Discussion

Considering the results presented in the previous section, some general considerations on the models discovered are given.

To better visualize the values of the four metrics measured on the discovered models, Figure 19 shows four bar charts (one for each metric) with the values obtained on the three event logs by the inductive and the genetic miner (the results obtained with the alpha miner are not visualized since, as already mentioned, no metrics have been measured for these models).

The models produced by the alpha miner are very simple, but they are not WF-nets and, thus, they completely fail to capture the routine as most of them contain unconnected nodes and/or unreachable transitions. As a consequence, all the models discovered by this algorithm and its more sophisticated variants cannot be actually used in a real environment as a model of the daily routine on Mondays. In other words, it is impossible to use one of these models either to predict the next action of the user in a specific situation or to detect dangerous situations by identifying deviations from the routine. Perhaps, this is caused by the presence of incompleteness and noise in the event log that are not being handled properly by the algorithm.

Overall, the models discovered by the inductive miner and its variants have a perfect fitness (0.87, on average). Hence, they are generally able to replay the events in the log. However, the most important flaw of these models is the low precision (0.31, on average). As a consequence, they are underfitted and too general meaning that they allow a lot of sequences of activities which are different than the ones in the event log. This means that they cannot be effectively used to detect deviations from the routine since a deviation would most likely be allowed in the model.

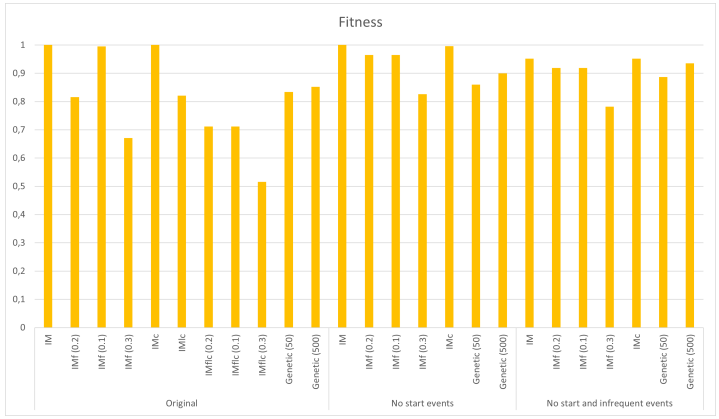
Note that this is true for all the models produced by this miner. Thus, using the alternative variants and filtering the event log improved the results, but not significantly. However, an exception is provided by the model discovered on the event log without start events by the IMf algorithm with a threshold of 0.2. In particular, this model has good values of fitness and precision, while being one of the simplest models discovered.

Furthermore, it is surprising noting how the IMlc and IMflc algorithms, which take into account the start and complete events, produced models which, overall, are worse than the ones produced by algorithms which do not consider these information. In addition, the average simplicity is 20, but some of the models are too complex and, yet, still unable to properly capture the routine of the user on Monday.

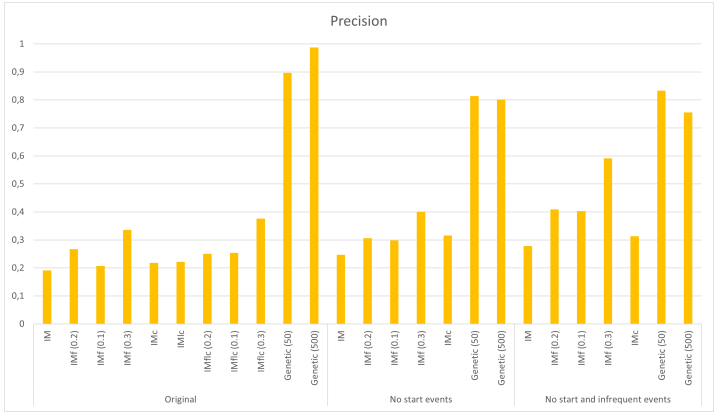
Regarding the models obtained with the genetic miner, they managed to better balance between fitness and precision. In particular, if on one hand these models have a similar fitness with respect to the models produced by the inductive miner (0.88, on average), on the other hand, they have a much higher precision (0.85, on average). Thus, they are not able to replay all the events in the log but, at the same time, they do not overgeneralize the routine meaning that the sequences of events that can be produced are more similar to the ones in the log.

However, these models are more complex (32, on average) than the ones discovered by the inductive miner (20, on average) and, as mentioned in the previous section, they are mainly made up of sequential structures falling between complex structures with silent transitions. In addition, note how increasing the number of iterations did not allow to improve a lot the models. In fact, it often resulted in lower quality models.

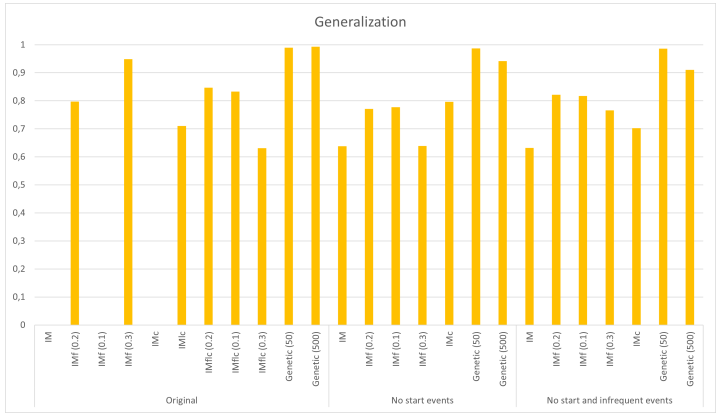
Nevertheless, these models might be used in a real environment to predict the next action of the user or to detect deviations as they do not allow much more behavior than the one in the event log. In other words, it is easy to detect a deviation from the typical routine by using one of these models.



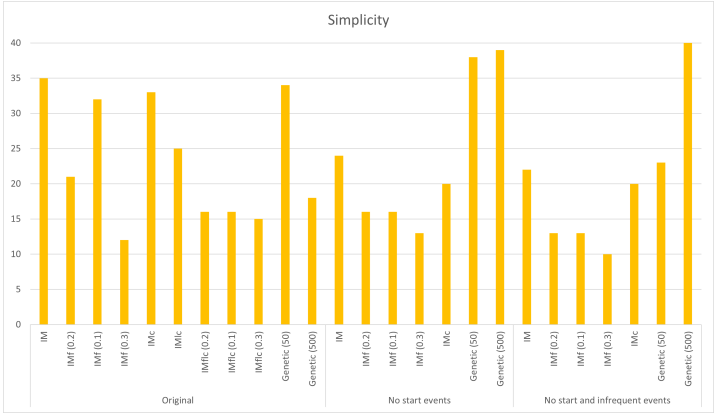
(a) Fitness



(b) Precision



(c) Generalization



(d) Simplicity

Figure 19: Bar charts with the values of the four quality metrics measured on the discovered models

Conclusions

In this case study a practical application of process mining techniques has been carried out. In particular, a labeled event log containing the events related to the daily routine on Mondays of a user has been analyzed and, then, three process discovery algorithms (namely, alpha miner, inductive miner and genetic miner) have been applied to discover, evaluate and compare a collection of process models discovered on three versions of the event log and expressed with a subclass of Petri nets, that is sound WF-nets.

It has been observed that the alpha miner has not been able to produce adequate models as all of them are not valid WF-nets. This is probably caused by the presence of noise and incompleteness in the event log.

On the other hand, the inductive miner and the genetic miner managed to discover WF-nets with varying levels of fitness, precision, generalization and simplicity. More specifically, the inductive miner produced model with high fitness, but low precision. Hence, these models are able to replay the events in the log, but they allow a lot of behavior different from the one in log. As a consequence, they cannot be effectively used to detect deviations from the daily routine.

The genetic miner instead, discovered models with a high level of complexity, but with high fitness, precision and generalization values. Thus, these models might actually be used in a real environment to detect emergency situations.

In the future, it would be interesting to run additional process mining algorithms (such as the heuristic miner or the region-based miner) and compare the results with the ones obtained in this case study. Furthermore, it would be also possible to record new events from the daily routine in order to obtain an event log with a higher number of cases in addition to the ones already in the log. This would allow the algorithms to have more samples of the user behavior and to hopefully discover better models. Finally, additional features might be added to the events (such as the timestamp) in order to take into account this information as well in the routine model.