universität wien

BACHELORARBEIT / BACHELOR’S THESIS

Titel der Bachelorarbeit / Title of the Bachelor’s Thesis

„Enhancing the Process Mining Visualization Tool by adding the Inductive Miner and improving the UI“

verfasst von / submitted by

Fabian Frauenberger

angestrebter akademischer Grad / in partial fulfilment of the requirements for the degree of

Bachelor of Science (BSc)

Wien, 2024 / Vienna, 2024

Studienkennzahl lt. Studienblatt / degree programme code as it appears on the student record sheet:	UA 033 521
Studienrichtung lt. Studienblatt / degree programme as it appears on the student record sheet:	Informatik - Data Science
Betreut von / Supervisor:	Ing. Dipl.-Ing. Dr.techn. Marian Lux

Abstract

Extracting information for event logs and creating process models is one discipline of process mining. Organization gain valuable insight in their complex processes through a visual representation of their workflows. There do exist many commercial and one open source process mining tool, that allows companies to analyze their workflows from event logs. Commercial tools require a paid license to use their applications, which can be a financial challenge for smaller enterprises or academic research groups. The analytic capabilities are often limited, due to a small range of algorithms, and no extension is possible. The open source tool can be used free of charge and has a wide variety of algorithms to use, at the cost of usability. The tool need a lot of professional knowledge to be used. One open source process mining tool has been created in two theses addressing these problems. The Process Mining Visualization Tool is an open source process mining desktop application written in python. It supports multiple heuristic and fuzzy miner, allows extension of the tool, and convinces with its ease of use. Users can import CSV files and create process graphs to visualize their processes. The goal of this bachelor's thesis is to contribute to the open source project and further enhance the tool. Bugs were fixed and the performance of the tool has been increased. The main part of the thesis was to update the user interface, and to incorporate the inductive miner into the tool. Changing the UI increased the usability, and ensure longer compatibility in the future. The inductive miner added a new valuable inside into process to the tool, by displaying more complex relationships, not only directly-follows relations, between events or group of events. Due to the change in the UI library, the project structure has been reworked to a more modular approach to ensure a clean code structure. The architecture has been reworked, by using the MVC pattern across all pages, to separate logic and UI clearly. These changes made code better reusable and also reduced the lines of code to add a new algorithm. The performance improvements reduced loading times inside the application and also allow mining larger datasets in a reasonable time. Error handling has been improved to stop crashes of the application. Due to these changes, this tool has drastically improved. The UI is responsive and up to date, the analytic capabilities have increased, the performance further improved and the tool's stability also increased. Furthermore, the structure has been reworked to ensure easy extensions of the tool in the future. These changes enhanced the tool significantly and creates an open source alternative to commercial process mining tool, to give valuable insights about processes to the users.

Contents

1 Motivation

Understanding processes is an important task for small and big companies. This can be achieved by using process mining tools. These tool transform logs into a process model, that visualizes the process. Users can see a visual representation of their processes and understand complex processes easier. This process is called process discovery. These tools are important for every sector to better understand their process or to find error in the execution of their processes.

There do exist a few of these tools, like PMTK[16], Disco[5], Celonis[2], and ProM 6[17], but most of them do need a license. All the previously mentioned tools except ProM 6 are not open source tools and a license is needed to use them. Most of the commercial tool come with limitations, like a limited range of process mining algorithms, or users can not choose the appropriate algorithm for their use cases. An exception to this is ProM 6 again, where users can choose the algorithm directly, but the major disadvantage is the usability of the tool, as this tool is rather complex to use and more for research purposes. Extending proprietary software is also most of the time not possible, except ProM 6. More about these tools will be mentioned in section 2.1

Due to these challenges with existing tools, the Process Mining Visualization Tool[3, 10] has been created. This should be an open source process mining tool, that tackled the previously mentioned challenges. It is easy to use, open source, easily extensible and users can choose the used algorithms. The main problems are the user interface and the lack of currently supported algorithms. More about these tools will be mentioned in section 2.2

These Limitations to overcome is the goal of this thesis. The goal is to update the current Process Mining Visualization Tool. First, the user interface will be reworked, by changing the Desktop UI library to a Web UI library. Afterward, the Inductive Miner will be added to the tool to enhance the analytic capabilities of the tool. As a last step, the existing tool's structure will be reworked, to make it easier to add new algorithms to the tool.

At the end of this project, a new open source tool will exist, that is easy to use, with a modern UI, better extensibility and more analytic capabilities.

2 Related Work

Process mining is an important discipline in Data Mining for smaller and bigger businesses. Process mining extracts information from process logs and can build process models to visualize the process, check the correctness of the process execution or find and improve the process execution. These are the three main challenges in process mining process discovery, conformance checking, and enhancement. Process mining aims to gave stakeholders a better insight in complex processes[11, 12, 21]

Process discovery aims to create a process model from event logs. It visualized the process to create a better understanding about the workflow, by creating Petri nets, BPMN Models or other graphs to visualize the event execution of a process. Many different process discovery algorithms exist, each with a different goal. There exist algorithm that focus on the soundness of the model, like the inductive miner, and others do create unsound models to only visualize significant process behavior, like the fuzzy miner[8, 10] and the heuristic miner[3, 21]. Mined models and the visualization of processes depend on the algorithms. Some only create directly-follows models and others create more complex BPMN models.[11, 12, 21]

Conformance checking tries to find anomalies between a log and a process model. For this discipline, a process model and a log of the process is needed. Evaluating discovered process model is one of the task that can be performed. Conformance checking techniques can also verify that company policies or legal regulations are obeyed. The system's correctness can also be proven by checking the logs with the process model, and changes in the model can be found easily with conformance checking techniques.[11, 21]

The last challenge is the enhancement. In this step, the process model is enhanced with additional information obtained by the log. This enhanced process model, can hold information, about the frequencies, time or duration of an event or other resources. This information adds valuable knowledge to the model, that can be used to further improve processes.[11, 21]

2.1 Existing Tools

Process mining tools have the goal to tackle these challenges and help businesses to analyze and visualize their workflows. there do exist a few process mining tools, but the most important ones are Celonis[2], Disco[5], PMTK[16], and ProM 6[17]. This section will give an overview of these tools and explain the main problems of these applications.

Celonis is a well known commercial process mining tool, that is used by many larger businesses. A paid license is mandatory to gain access to the tool. Celonis runs primarily in the cloud and has a lot of useful features for larger companies. Among these are process discovery, conformance checking, and the enhancement of process models. The application provides log statistics, extract data from sources directly and also monitor processes and check if they are run according to the model. Celonis uses a proprietary mining algorithm and AI for its analytic features, and also supports object-centric process mining[4]. Figure 2.1 shows a process model obtained by Celonis. Custom extensions are not possible.[2]

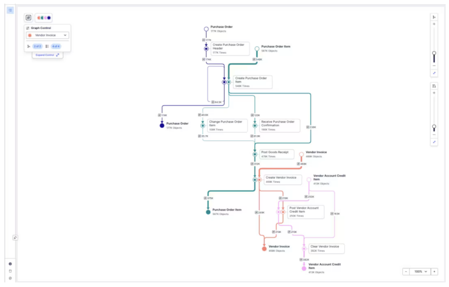


Figure 2.1: Celonis Process Model Graph[2]

Disco, developed by Fluxicon, is a commercial process mining tool that requires a license for use. Disco uses a custom, closed source, implementation of the fuzzy miner and supports no other mining algorithms. Figure 2.2 shows the graph found by Disco. Logs need to be imported in a supported format. The tool has log analysis and case analysis functionalities and operates locally as a desktop application. However, it does not support custom algorithms.[5]

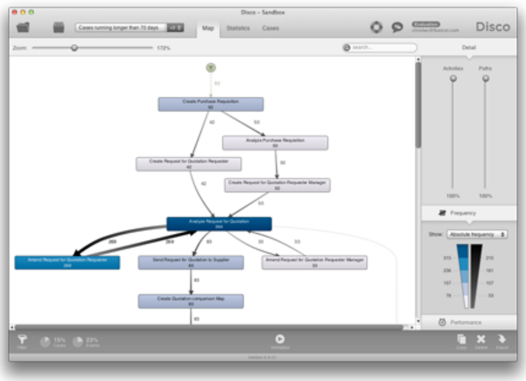


Figure 2.2: Disco Process Model Graph[5]

PMTK is a newer process mining tool, but also needs a paid license to use it. The tool uses two different mining algorithms, an inductive miner and a directly-follows miner, that allows filtering of events and edges. This tool also shows statistics for logs and cases and also supports conformance checking and is run locally as a desktop application. The application does not allow any custom extensions.[16] Figure 2.3 shows the PMTK tool, with their directly-follows miner.

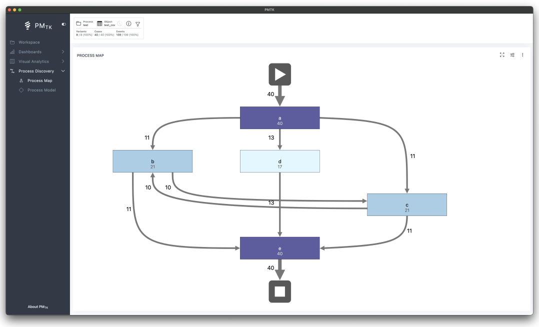


Figure 2.3: PMTK Process Mapping Graph

Prom 6 is an open source process mining tool written in Java. Due to its open source nature, the tool can be extended with custom algorithm implementations. This application does have a wide variety of mining algorithms to choose from. Many tasks can be performed on these log files. Log analysis, log filtering, mining algorithms, or conformance checking can be performed. All of these actions need to be chosen from a list, as Figure 2.4 shows. The tool is technical, not easily useable for nonprofessional users and mostly used in the academic field.[17] Figure shows

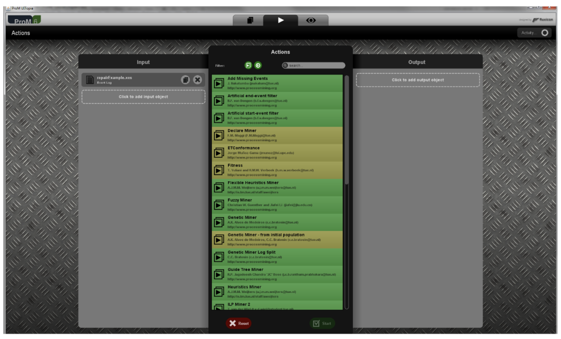


Figure 2.4: Prom 6 Action View [17]

As this section showed, there are a lot of different existing tool, but each of these have their own problems. Proprietary tool are widely used, but lack support for customization. Commercial tools also support a limited amount of process mining algorithms, and users often have no influence on the used algorithms. These tools can also become a financial burden for smaller organizations and effect the insight about their processes. The most known open source process mining tool tackled these challenges, at the cost of usability.

2.2 Process Mining Visualization Tool

The Process Mining Visualization Tool has the goal to be the best of these two groups. It was created in two theses[3, 10]. It aims to be an open source tool, that convinces with its ease of use and a broader range of algorithms. Two algorithms are currently supported, the heuristic miner[3] and the fuzzy miner[10], and further can be easily added. The tool uses the outdated PyQt5 UI library to create the desktop application. Figure 2.5 shows the heuristic graph view of the application. [3, 10]

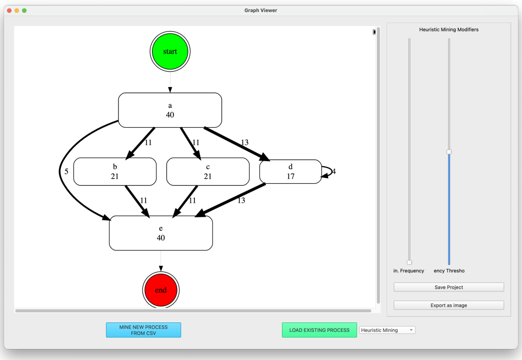


Figure 2.5: Heuristic Graph View

The Process Mining Visualization Tool is a step in the right direction, but still has only a limited range of mining algorithms. Larger datasets can not be used in the tool and the performance could also be improved. The UI library could also further be improved to ensure the future of this open source project and modernize the user interface. This thesis will improve the performance, update the UI and add the inductive miner to the tool.

3 Algorithms

This section will explain the mining algorithms. First, we will start with a few preliminary terms for process mining. Then the three algorithms will be explained. The heuristic miner and fuzzy miner will only have a short description, because there already exist theses about them, and they were not part of this thesis. The inductive miner will get a more in-depth description.

3.1 Preliminaries

This part explains a few preliminary definitions that are important in process mining.

The event log is a collection of event traces. One event trace represents one execution of a process. An event trace is an ordered sequence of events. This order is important, because it also shows the execution order of events.[12]

The directly-follows graph is a graph that shows directly-follows relations of an event log. The graph contains a list of all edges, nodes, start and end nodes. An edge from one node to another node exists, if the other node is directly follows the one node in an event trace. The start nodes are a collection of all nodes, that are at the start of a trace, while the end nodes are all the nodes at the end of a trace. [12]

A Process tree is an abstract representation of a workflow net, like Petri-nets or BPMN. The tree is a rooted tree, where all leafs contain an event or a silent activity and all other nodes are operation. these operations may vary depending on the application. The inductive miner has a sequence, exclusive, parallel and loop operator. The tree shows the operation performed on the directly-follows graph in case of the inductive miner.[12] Figure 3.1 shows a process tree obtained by the inductive miner

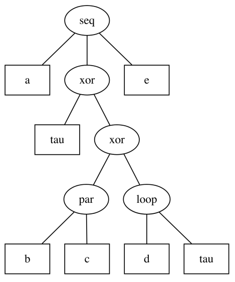


Figure 3.1: Process tree

3.2 Heuristic Miner

The heuristic miner is a simple, and effective process mining algorithm. It considers the frequencies of events and sequences when constructing the process model and focuses on the most common paths in the process while removing infrequent ones. Different metrics are used to remove connections and events from the process model.[3, 21]

The algorithm creates a directly-follows graph and stores it as a succession matrix. The different metrics are used to find infrequent nodes and edges and these are discarded.

3.2.1 Metrics

Currently, two metrics are used to simplify the graph, the frequency metric and the dependency metric. Both metrics concentrate on the connections of the graph.

The frequency metric is calculated for edges and nodes/events. The event frequency counts the occurrences of an event in the log. The edge frequency counts the number of times one event is directly followed by another event.

The dependency metric determines, how one-sided a relationship is between two edges. It compared how significant an edge is by the relative frequency of this relationship, compared to all other connections from the node. Figure 3.2 shows the formula for this metric.

$ifa \neq b:$

$$D(a > b) = \frac{S(a > b) - S(b > a)}{S(a > b) + S(b > a) + 1} \tag{1}$$

$ifa = b:$

$$D(a > a) = \frac{S(a > a)}{S(a > a) + 1} \tag{2}$$

Figure 3.2: Formula for calculating the dependency[3]

3.2.2 Filtering

There are two filtering parameters in the current implementation of the heuristic miner. The minimum frequency and the dependency threshold

The minimum frequency filters out edges, that have a lower frequency than that threshold. The range of this threshold is from 1 to the maximum edge frequency. Additionally, nodes with a lower frequency are also removed.

The dependency threshold is in the range of 0.0 to 1.0. It removed edges, that have a lower dependency score than that threshold.

3.3 Fuzzy Miner

The fuzzy miner is a mining algorithm is best used for less structured processes. The algorithm uses the significance and the correlation metric to abstract the process model. Nodes can be collapsed to clusters and edges can be merged for an easier view of the process model.[10, 8]

Building clusters follows three rules. Significant nodes are kept and will not be merged, less significant nodes and highly correlated nodes will be merged into a cluster, and less significant and lowly correlated nodes will be removed. Edge filtering uses the edge cutoff metric, that will be described later.[10, 8]

3.3.1 Metrics

The fuzzy miner uses the following metrics, unary significance, binary significance, correlation, utility ratio, utility value and the edge cutoff value.[10]

The unary significance describes the relative importance of an event. A frequency significance is used, that counts the frequency of all the events and divides it by the maximum event frequency.[10]

The binary significance is rather simply calculated. It uses the source node frequency.

Correlation measures how closely related two events are. All edges between two nodes are counted, and divided by the sum of all edges with the same source.[10]

The utility ratio defines a ratio to calculate the utility value by, weighting the correlation and significance of an edge.[10]

The utility value is defined by using the binary significance, correlation, and the utility ratio. Figure 3.3 shows the formula for this calculation. The edge cutoff is calculated by normalizing the utility value for edges with the same target node. The Min-Max normalization is used.[10]

$$\mathrm{util}\,(A, B) = \mathrm{ur} \cdot \mathrm{sig}\,(A, B) + (1 - \mathrm{ur}) \cdot \mathrm{cor}\,(A, B)$$

Figure 3.3: Formula to calculate the utility value[10]

3.3.2 Filtering

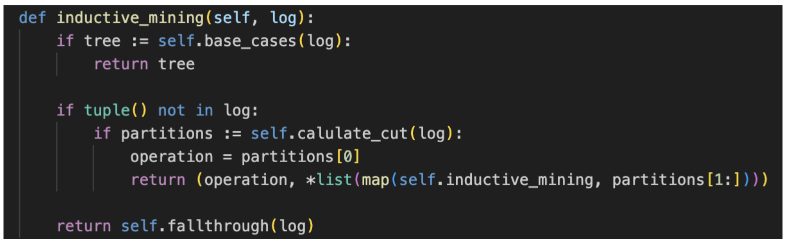
Three metrics are used for filtering, the significance, the correlation and the edge cutoff.

The significance value in the range of 0.0 and 1.0. This value defines the threshold, until a node is considered significant. The correlation value has the same range and defines the threshold, until nodes are considered lowly correlated. The edge cutoff value is also in a range of 0.0 and 1.0. It removes all edges that have a lower edge cutoff value, than this threshold.

3.4 Inductive Miner

The inductive miner is a recursive mining algorithm. It does not consider frequencies, but only looks at directly-follows relations. The algorithm uses an event log and create a directly follows graph from it.Through cuts in the graph, exclusive executions, parallel executions, sequential executions, and loops can be found. The algorithm produces sound process models.[12]

First, the algorithm checks for base cases, then it tries to find partitions in the directly follows graphs. Should a partition be found, the log is split in smaller logs and the algorithm is performed on them again until the algorithm terminates. If no cut is found, then a fall through is applied. In each recursion, a sub process tree is created. These trees are merged together to achieve the tree.[12] Figure 3.4 shows the structure of the algorithm as previously mentioned.


```python
def inductive_mining(self, log):
    if tree := self.base_cases(log):
        return tree

    if tuple() not in log:
        if partitions := self.calulate_cut(log):
            operation = partitions[0]
            return (operation, *list(map(self.inductive_mining, partitions[1:])))

    return self.fallthrough(log)
```

Figure 3.4: Structure of the Inductive Mining Algorithm

This implementation uses a small deviation from the described algorithm. Frequency is taken into account before the algorithm is run. To eliminate small outliers, infrequent events or traces can be removed. This was done to make the model easier readable and to find better cuts.

3.4.1 Base Cases

First, the log is checked for a base case. currently, there only exist two base cases, if the log is empty or only an empty trace is in the log, a silent activity is returned. This silent activity represents a silent transition, without an action. If there only exists one trace with exactly one event, then this event is returned. These are process trees with only one activity and will be the leafs of the final process tree. The base cases are one way to stop the recursive calls.[12, 11]

3.4.2 Cuts

The second step is trying to find a cut in the directly-follows graph. There do exist 4 cuts in this implementation. The exclusive cut, the sequence cut, the parallel cut and the loop cut. If a cut is found, the log is split in multiple sublogs, the splitting depends on the found cut. Should an empty trace be in the log, cut detection is skipped and a fall through is applied.[12] If, for cut detection, a partition of size one is found, the cut could not be detected and none is returned. The next cut will be performed until all cuts were checked.

The exclusive cut finds a partition in the graph, so that there does not exist an edge from one partition to any other partition. When all nodes are connected, no cut could be found and the algorithm checks for a sequence cut.[12, 11]. This cut is found by calculating the connected components of the directly follows graph. Figure 3.5 shows an example of the cut.

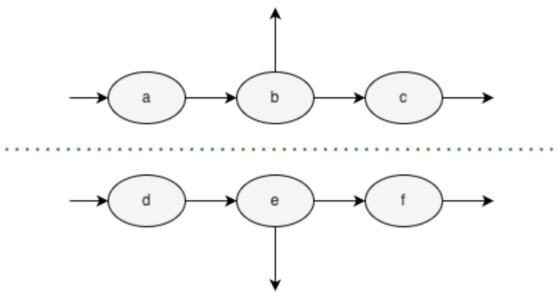


Figure 3.5: Exclusive Cut of a directly-follows graph

The sequence cut is an order cut. It tries to find partitions so that there exists a path from a lower partition to a higher partition, but no path from the higher partition from the lower one. If only one partition was found, the algorithm checks for a parallel cut.[12, 11] This cut is found by a greedy approach, inspired by the approach in [11]. All pairs of nodes are first put in their own partitions, afterward the reachability of all node pairs is checked. If two nodes are pairwise reachable or pairwise unreachable, they have to be in the same partition and the partitions of these two nodes are merged. As last step, the partitions are order after their reachability. Figure 3.6 shows an example of the cut.

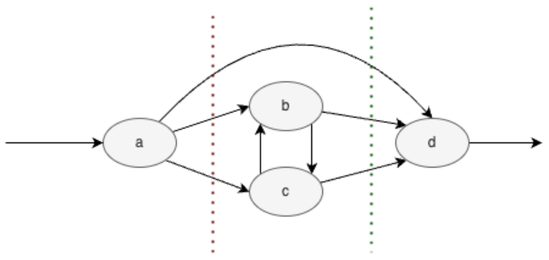


Figure 3.6: Sequence Cut of a directly-follows graph

The parallel cut shows that two partitions can be done concurrently. Every partition needs at least one start and one end event. A parallel cut is found if all nodes from one partition have an edge to all other nodes that are not in the partition. If no cut is found, the algorithm tries to find a loop cut.[12, 11]. This cut is calculated, by first inverting the directly-follows graph. All double edges are removed, and simple edges or no edges between two nodes will be replaced with double edges. Then the connected components are calculated. If a partition does not have a start or end activity, it is merged. First, partitions with only start activities are merges with partitions with only end activities, to try to maximize the cut. Afterward, the other partitions are merged, with already correct partitions.Figure 3.7 shows an example of the cut.

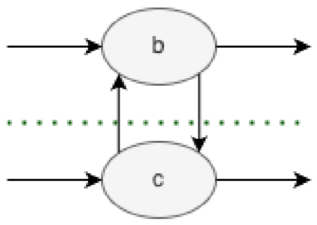


Figure 3.7: Parallel Cut of a directly-follows graph

The last cut is the loop cut. It consists of a do part and one or multiple redo part. The do parts contains all start and end activities and represents a successful execution, the redo part does represent a failed execution. Redo parts always start at end activities and end at start activities. There does not exist an edge between multiple redo parts. Also, there does not exist an edge from a start activity to an activity of any redo partitions and there does not exist an edge from a redo partition to any end activities. Furthermore, if an activity from a redo partition has an edge from an end activity to itself, then all end activities need to have an edge to that activity.The same is true for edges to start activities. If no cut can be found, then a fall through is applied.[12, 11]. This cut finds its first partition, by adding all start and end activities to the partition. Then these activities are removed from the directly-follows graph. To find candidates for the other partitions, the connected components are found of the new graph. The candidates are checked if all conditions are fulfilled. Should one of these conditions not be fulfilled, the partition is merged with the first partition. Figure 3.8 shows an example of the cut. The partition in the middle represent the do part, and the other two partitions are redo parts.

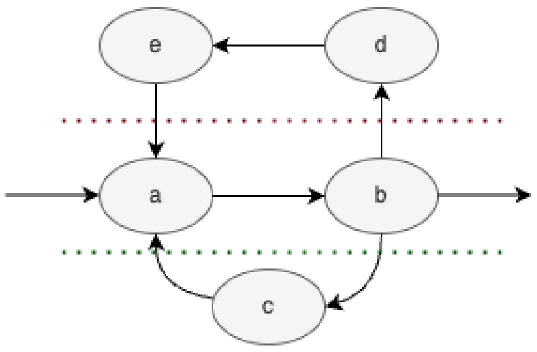


Figure 3.8: Loop Cut of a directly-follows graph

3.4.3 Log Splitting

Each cut has its own rules on how to split the log. Each of the four log splitting procedures will be explained. The log will be split in a number of sublogs, that is equal to the number of partitions.

The first split will be the exclusive split. Each trace is checked as a whole and will be assigned to one sublog. If the events of a trace are all in the same partition, then the trace will be added to the sublog of the partition.[12]

The sequence split does split each trace in subtraces and each subtrace will be assigned to a sublog. The algorithm starts with the first partition and an empty trace. While an event is part of the partition, the event will be added to the trace and the next element is checked. Should the element not be part of the trace, then the trace will be added to the sublog of the partition and a new empty trace will be created. The algorithm goes to the next partition and checks the event like described previously. This is done, until the full trace has been traversed.[12]

The parallel split does project each trace, for each partition. The project function only includes the events of the partitions in the projected trace, but keeps the relative order of the events. Should a trace not contain any elements of a partition, then an empty trace is projected.[12]

The loop split does split each trace in subtraces and each subtrace will be assigned to a sublog. The algorithm starts with the first element and an empty trace. First, the partition of the current event needs to be found, and the event is added to the empty trace. Then the next event is checked. As long as the events are in the same partition, they are added to the trace. If an event is not in the partition, the trace will be added to the sublog of the partition, an empty trace will be created, and the new partition will be found. These steps continue, until the end of the trace.[12]

3.4.4 Fall through

The fall through is the last resort of the algorithm, if no cuts can be found, multiple fall through can be applied.

The first fall through checks if an empty trace is present in the log. Should this be the case, a process tree with the xor operation will be returned. The children will be tau, symbolizing a silent activity and the recursive call of the inductive miner on the log without the empty trace.[11]

The second case occurs, when the log has contains only one event, but it is executed more than one. Then a process tree, with a loop operation and the event and tau as children, will be returned. This process tree shows that the event can be executed more than once.[11]

The last fall through is the flower model. This is a process tree, with the loop operator and tau as the first child and all other events in the log as other children. This shows that any order of execution is possible.[11]

3.4.5 Metrics

The algorithm uses two metrics, the activity threshold, and the traces threshold.

The activity threshold describes the frequency of an event in relation to the most frequent event. This metric is calculated, by dividing the frequency of the event by the maximum frequency of the events.

The traces threshold describes the frequency of a trace in relation to the most frequent trace. This metric is calculated, by dividing the frequency of a trace by the maximum frequency of all traces.

3.4.6 Filtering

Two metrics are used for filtering, the activity threshold, and the traces threshold.

The activity threshold is in the range of 0.0 and 1.0. It will remove activities/events from the log, that have a lower threshold than set.

The traces threshold is in the range of 0.0 and 1.0. It will remove traces from the log, that have a lower threshold than this parameter.

4 Design

This section will explain design decisions and explain the need of some components of the application. First, the UI library choice will be explained. Afterward, the MVC Pattern will be described, how it was implemented and its advantages. The last two sections explain the need for custom graph classes and an interactive graph component.

4.1 UI Library Choice

The usage of the streamlit[20] library was one requirement of the thesis. Streamlit is a Web UI library that is easy to use for python developers. No Javascript/frontend knowledge is needed to create modern, responsive, and interactive pages, like in other frameworks. Streamlit is an open source framework with a great community support and many custom components written by the community. Creating these components needs frontend knowledge as described in section 4.4. The main reason for choosing this library is the easy use of it, for python developers.

4.2 Model View Controller Pattern

The Model View Controller pattern (MVC) specifies that an application has to be separated in three components, View, Controller, and Model. The Model does hold application data and contains the business logic, the View is responsible for displaying the data and input elements. Events from users, e.g. button presses, input field changes, are passed to the controller. The controller processes events from the view. This component passes the data to the correct model, and sends the updated data to the view. Figure 4.1 shows this behavior.[18, 7]

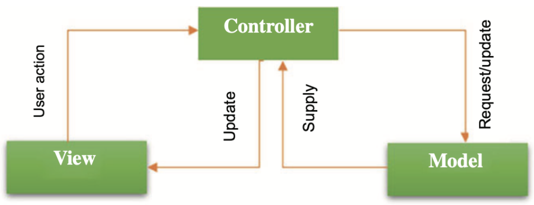


Figure 4.1: Model View Controller Pattern[18]

Figure 4.2 shows an overview of the MVC structure in the project. The BaseModel class does not exist and can be substituted, by multiple different models. For an easier UML model, this class has been used.

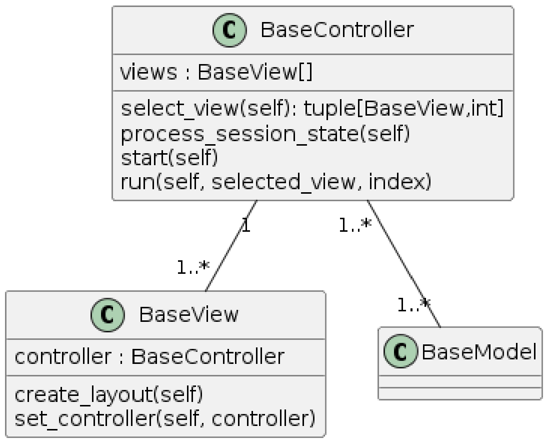

BaseController
views : BaseView[]
select_view(self): tuple[BaseView,int]
process_session_state(self)
start(self)
run(self, selected_view, index)
1
1..*
1..*
1..*
BaseView
controller : BaseController
create_layout(self)
set_controller(self, controller)
BaseModel

Figure 4.2: UML Diagram MVC Pattern

In this project, the MVC pattern is used for each page. Every page has exactly one controller and at least one view, as shown in Figure 4.2. A controller can have multiple views, to support different representation in the future. Currently, each page has only one view. The views only display the data and pass events to the controller. The data is received from the controller. Controllers can have multiple views and models. The controller is the entry point for each page. It reads data from the session storage, events are stored in the session storage in streamlit[20], and passes the data to the correct model. Afterward, the controller gets the updated, processed data from the model and passes the data to the view. The view then displays the data from the controller. The controller can also switch between different views. Error handling is done in the controller. The models contain the application data and the business logic of the application. Each feature has its own model, e.g. export, import, data transformations, mining algorithms etc., and there are no two models that do the same. Models are only called by a controller, never by the views.

This design has a few major advantages. By separating the three components, the code is easier to read and maintain. Models can be reused by different controllers, which reduces code duplication. The modular design allows separate development and testing of the components. Models can easily be extended, or a new implementation can easily be substituted, if the method signatures are the same.[18, 7]

4.3 Custom Graphs

The project uses a custom graph class. This class uses graphviz.Digraph[6] internally to visualize the graph. The graphviz graph will only be uses to visualize the graphs. The custom graph class will store nodes and edges independently of graphviz. The graph elements can store additional node information, that will be displayed in the interactive graph. Figure 4.3 shows a simple UML diagram of the BaseGraph class. New custom graph classes can be created, by extending from the base class. These classes can define a classwide style and add simpler methods to add nodes and edges. Internally, they still use the BaseGraph methods.

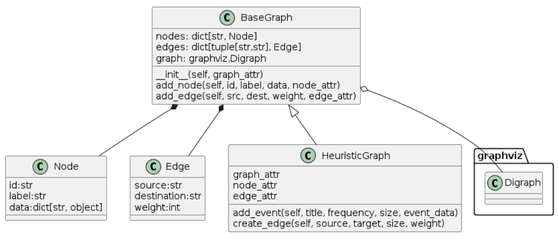

BaseGraph
nodes: dict[str, Node]
edges: dict[tuple[str,str], Edge]
graph: graphviz.Digraph
__init__(self, graph_attr)
add_node(self, id, label, data, node_attr)
add_edge(self, src, dest, weight, edge_attr)
Node
id:str
label:str
data:dict[str, object]
Edge
source:str
destination:str
weight:int
HeuristicGraph
graph_attr
node_attr
edge_attr
add_event(self, title, frequency, size, event_data)
create_edge(self, source, target, size, weight)
graphviz
Digraph

Figure 4.3: Custom Graph UML Diagram

Graphviz is the leading graph visualization library with the capacity to completely customize the graph. Since this feature is an advantage, the graph class has to use the graphviz library internally, to create the graphs. Graphs can also easily be exported to PNGs or SVGs using graphviz. Graphviz does not have getters to get nodes or edges. This makes testing difficult and is one of the reasons for the custom graph class, to make the algorithms easier testable. Another reason for this graph class is the need to add additional data to nodes. In the current state of the application this feature may not be necessary, but for future algorithms additional node data may be useful.

4.4 Interactive Graph Component

Streamlit has no native support for interactive graphs. There are user libraries that support interactive graphs, but these lack functionality for this use case, as most of these do not support the binding of on-Click-Listeners or have limited styling options. Therefore, the interactive graph needs to be implemented from scratch. With streamlit, custom components can be made using the Web-Framework React or Typescript[19].

The interactive Graph component has been implemented using the Streamlit Components API[19]. A Python component has been written, that communicates with a React Component. The React Component uses d3[1] and d3-graphviz[9] to visualize a graphviz graph. D3 is one of the leading data visualization library in JavaScript. The support for visualizing graphviz graphs was also one important factor in this decision

On all nodes, on-Click-Listeners are registered. If a node is clicked, the component return the node_id and a clickId to the python component. The clickId is needed to recognize if a node new node has been clicked or if the returned value is an already processed click. Additionally, a reset button has been included, to reset the zoom.

The Python component needs a custom graph passed as parameter. If a node has been clicked, the node_id will be transformed to a string and displayed. How the nodes are parsed to strings is defined in the custom graph.

5 Implementation

This chapter will be about the application. First, the file structure and requirements will be discussed. Afterward, the project will be shown, and last testing will be discussed and the most important ways to extend from the project will be mentioned. The implementation can be found on GitHub: https://github.com/fabianf00/ProcessMiningVisualization_WS23

5.1 File Structure

The project is separated in different modules/packages. The starting file is the streamlit_app.py file, config.py contains basic configurations, logger.py contains functions to get a logger and the basic configuration of it. UI components can be found in the component module. All classes regarding graphs are stored in the graphs module. This includes the visualization graph classes, the directly-follows graph and the cuts of the graph. All tests are written in the tests module, templates are stored in the template module.

The docs directory contains all documentation files. These include models of the project, current and prior, and theses to this project. Also, the documentations of the mining algorithms should be stored there.

The ui module contains all the pages, separated in a folder for each page. Each page folder contains a controller and at least one view. The mining algorithms are stored in the mining_algorithms package. Each algorithm has its own file. The other models are stored in modules, that describes the feature.

5.2 Requirements

This part will mention the libraries used in the project.

Streamlit[20] is used as the UI library, with version 1.36.0. It is an easy-to-use library for python developers to create modern UIs, without frontend knowledge. Numpy[15] is used for vector and matrix operations. Pandas[14] is used to read CSV files and transform CSV files to event logs.

DDCAL [13] is a clustering library. In this application, it is used for the visualization of the graph. More frequent nodes should be bigger than less frequent nodes, to find an appropriate scale and handle outliers, this library is used to cluster nodes and get a scale factor.

Graphviz[6] is used as the graph visualization library. It is one of the leading graph visualization library, with great options of customization. Graphviz needs to be installed externally to be usable. Further instructions can be fund in the GitHub repository.

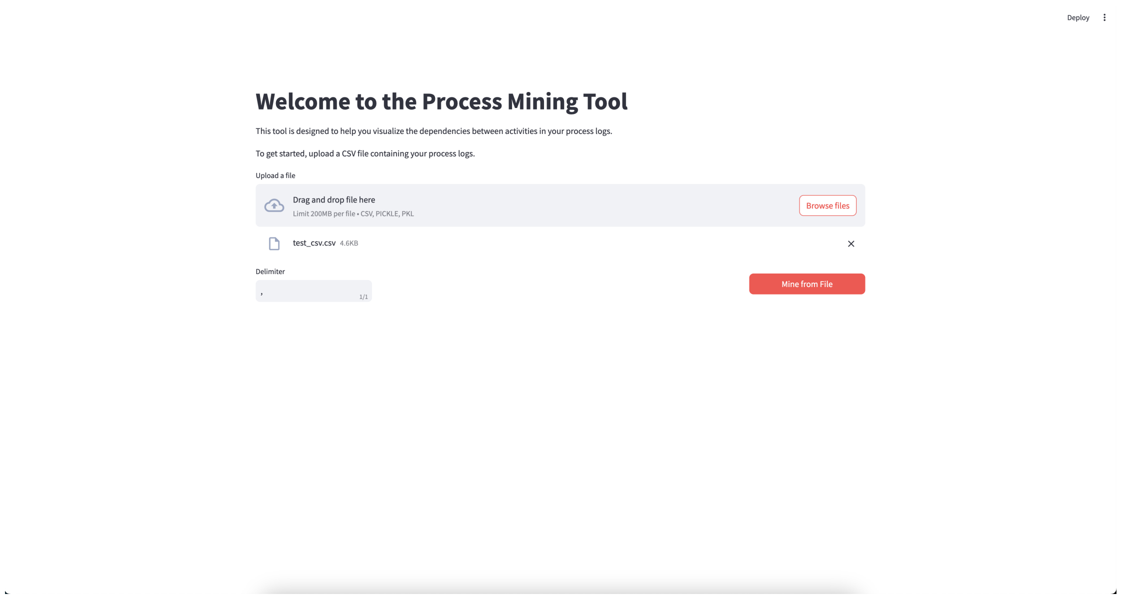
5.3 How to use the Application

The application can be started by using the following command in the project directory, with the requirements, mentioned in section 5.2, installed:

```
streamlit run streamlit_app.py
```

5.3.1 Home page

The first page of the application is the Home page. The page is shown in Figure 5.1. Users can upload CSV or Pickle files to the website, the pickle file has to be an exported model from the page.


Welcome to the Process Mining Tool

This tool is designed to help you visualize the dependencies between activities in your process logs.

To get started, upload a CSV file containing your process logs.

Upload a file

Drag and drop file here
Limit 200MB per file • CSV, PICKLE, PKL

test_csv.csv 4.6KB

Delimiter: ,

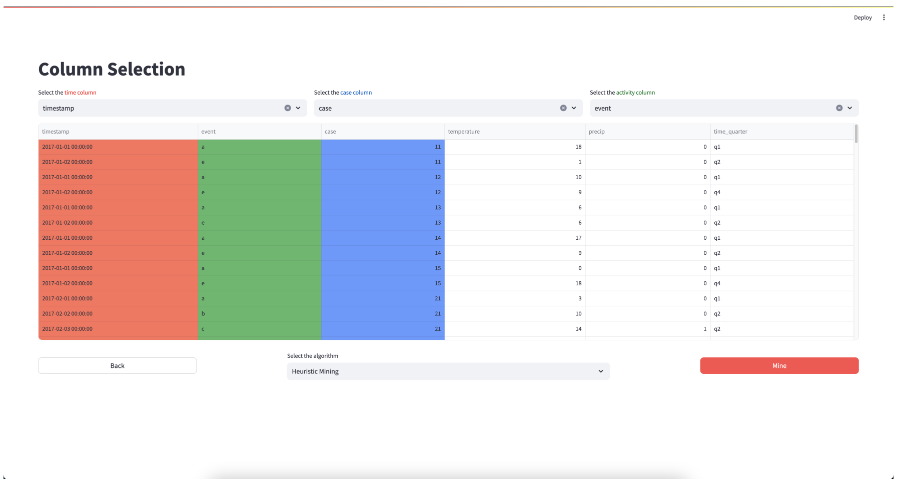
Figure 5.1: Home page

Figure 5.1 shows an uploaded CSV file. The application tries to detect the delimiter of the CSV file, using a method in the DetectionModel. If no delimiter can be detected, an empty filed is shown. In this case, or if the detect delimiter is incorrect, the user can manually set the delimiter. By pressing the "Mine from file" button, the CSV file is read and the dataframe is build and passed to the ColumnSelection page, mentioned in section 5.3.2.

Uploads the user a pickle file, a dropdown menu is shown. The user has to manually select the correct algorithm of the model. After the algorithm is chosen and the "Import Model" button has been pressed, the user is redirected to the Mining Algorithm Page, mentioned in section 5.3.3. If the selected model type is not the same as the uploaded one, an error message is shown.

5.3.2 Column Selection page

The ColumnSelection page shows the dataframe and lets users select columns, from which an event log will later be build, and choose a mining algorithm. The page is shown in Figure 5.2



Select the time column: timestamp

Select the case column: case

Select the activity column: event

timestamp	event	case	temperature	precip	time_quarter
2017-01-01 00:00:00	a	11	18	0	q1
2017-01-02 00:00:00	e	11	1	0	q2
2017-01-01 00:00:00	a	12	10	0	q1
2017-01-02 00:00:00	e	12	9	0	q4
2017-01-01 00:00:00	a	13	6	0	q1
2017-01-02 00:00:00	e	13	6	0	q2
2017-01-01 00:00:00	a	14	17	0	q1
2017-01-02 00:00:00	e	14	9	0	q2
2017-01-01 00:00:00	a	15	0	0	q1
2017-01-02 00:00:00	e	15	18	0	q4
2017-02-01 00:00:00	a	21	3	0	q1
2017-02-02 00:00:00	b	21	10	0	q2
2017-02-03 00:00:00	c	21	14	1	q2

Select the algorithm: Heuristic Mining

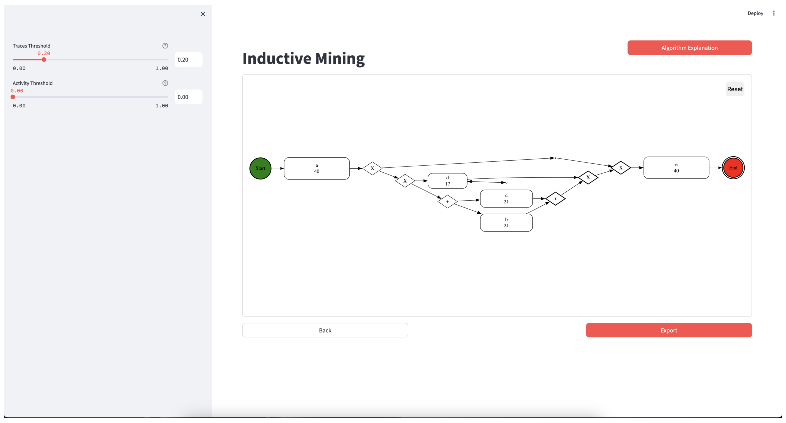
Figure 5.2: Column Selection page

Currently, the user needs to choose 3 columns, a time column, a case column and an activity/event column. The case column groups events from the same process/case together, the activity column contains the event, and the time column is used to order events in a case. The PredictionModel tries to assign the columns automatically, but wrong predictions can happen. Users can manually set these columns through the dropdown menu. The Dropdown Menu on the button lets users choose the algorithms. Each algorithm could need different columns, and therefore another view could be shown. After pressing the "Mine" button, the User is redirected to one of the Mining Algorithm pages, mentioned in section 5.3.3.

5.3.3 Mining Algorithm page

The Mining Algorithm pages perform a mining algorithm and show the generated graph to the user. Figure 5.3 shows the Mining Algorithm page for the Inductive Miner. Because all Algorithm Mining pages look similar, only one will be described. By pressing the "Export" button, users will be redirected to the Export page, mentioned in 5.3.4.

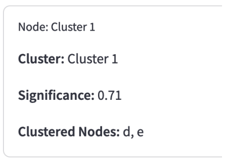
Figure 5.3 shows the structure of the pages. The page has a title, the interactive graph components, design mentioned in section 4.4, navigation buttons, node information under the buttons, and on the sidebar input elements, most of the time number_input_sliders, to adjust algorithm parameters. The graph can be explored by zooming or moving around, if a node is clicked additional information will be displayed under the buttons. Figure 5.4 shows an example from the Fuzzy Miner. For this example, the Fuzzy Miner was chosen, as cluster nodes contain the most additional information.


Inductive Mining

Traces Threshold 0.20

Activity Threshold 0.00

Figure 5.3: Inductive Mining Algorithm page


Node: Cluster 1

Cluster: Cluster 1

Significance: 0.71

Clustered Nodes: d, e

Figure 5.4: Fuzzy Mining Cluster Node Information

5.3.4 Export page

The last Page is the Export page. Figure 5.5 displays the page. The main area of the page shows a graph previews as PNG. This display is not an interactive graph, but a static PNG file. On the sidebar, there exists a dropdown menu to choose an export format, DOT, PNG, SVG are supported. If PNG is the chosen format, users can also set the DPI, for a better image quality. To export the file, the "Export" button needs to be clicked, and the file will be downloaded. Clicking the "Export Model" button exports the mining model in the pickle file format. This file can be imported on the Home page, section 5.3.1.

5.4 Testing

Testing is an important step in the software development process, to ensure the correctness of the implementation. To follows this best practice, unit tests were used to ensure the correctness of the logic. On the UI of the project, views and controllers, no automated tests were performed. These part were only tested manually. How to run the tests is mentioned in the GitHub repository.

Existing tests were retained, the heuristic mining tests were adapted to use the new graph class, no tests for the fuzzy miner were added. The inductive miner has been fully tested, the algorithm as a whole, but also the underlying datastucture, the dfg, the cuts of the graph, and log splitting.

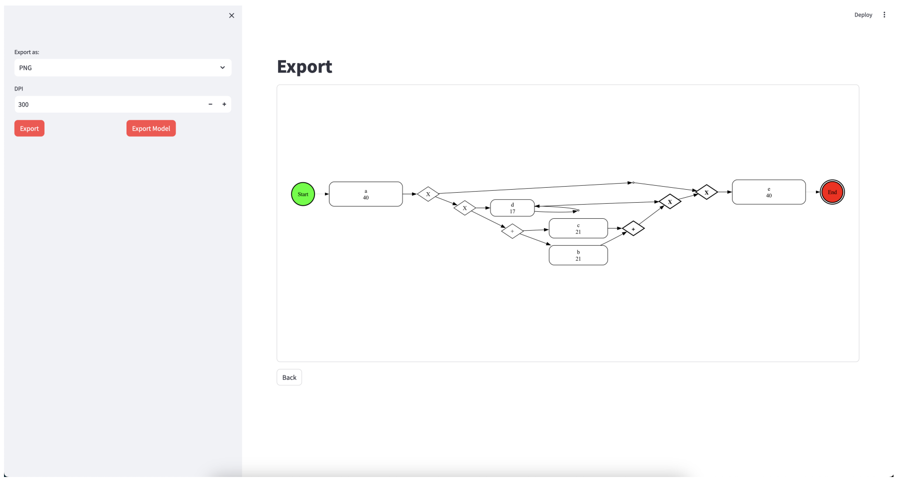


Figure 5.5: Export page

Most of the models have been tested, with common cases, but also edge cases. The IO Models were not tested, as they use already tested library calls to read the data, the dataframe styling was also not tested for the same reasons.

To ensure the correctness of the Inductive Miner, the PMTK[16] tool was used with the same dataset, the test_csv dataset. Figure5.6 shows the graph of the implementation, while Figure 5.7 shows the mined graph from the PMTK tool. As it can be seen, both graph show the same model, they just represent some parts, e.g. loops or two consecutive exclusive cuts, differently. This example shows, that the algorithm works correctly.

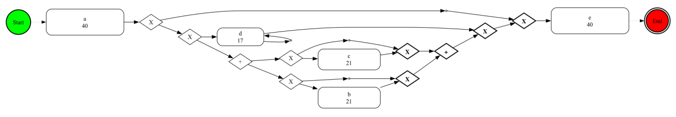


Figure 5.6: Inductive Mining Graph using the test_csv dataset

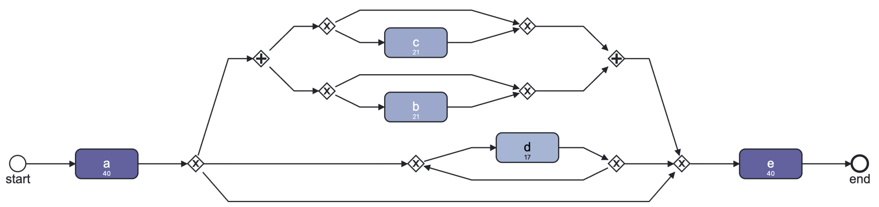


Figure 5.7: PMTK Inductive Mining Graph using the test_csv dataset

5.5 Working on the project

This section explains how to extend the project. There are many extension options, but only adding a new mining algorithm will be covered in this section. Other ways are mentioned in the GitHub repository.

5.5.1 Adding a Mining Algorithm

The project uses the MVC pattern to build pages. To add a mining Algorithm at least 3 classes are needed, an AlgorithmView, an AlgorithmController and a MiningModel. The View(s) and the Controller should be in its own folder in the ui directory. It is recommended to use the template provided in templates/algorithm_ui_template. The MiningModel should be stored in the mining_algorithms folder, and the model should inherit from either the MiningInterface or the BaseMining class. Developers need to choose which class suits their algorithm best. Additionally, it is recommended to create a custom Graph class, to define styles of nodes and edges directly in it and to make the code more readable. The Graph class should be stored in the graphs/visualization directory, and the BaseGraph is the parent class. It is also possible to use the BaseGraph directly.

To create a View, the class has to inherit from the BaseAlgorithmView class. Inside this base class, most of the code is already defined. For the child class, only the sliders need to be defined. This is done in the render_sidebar method. The keys of the sliders have to be equal to keys of the session state, which are read and set in the controller. It is recommended to use the same keys in the sidebar_values dictionary.

The controller has to inherit from the BaseAlgorithmController class. Inside the constructor the mining model class has to be defined (class not instance!). Furthermore, the following methods need to be overridden: process_algorithm_parameters, perform_mining, have_parameters_changed, get_sidebar_values. process_algorithm_parameters reads the parameters for the mining algorithm from the session state, or sets them with default values if they are not set. perform_mining calls the model to perform the mining algorithm with the algorithm parameters. have_parameters_changed checks if a parameter of the algorithm has changed. If this is not the case, the algorithm does not rerun, as the result stays the same. get_sidebar_values is used to set minimum and maximum values for the slider.

All dataframes are transformed to algorithm data, by default a log dictionary. If an algorithm needs another data format or additional data from the dataframe, developers can override the transform_df_to_log(self, df, **selected_columns) method. This function takes in the pandas dataframe and the selected columns, where a key is the name of the selected column ('time_columns') and the value is the selected column of the dateframe (e.g. 'time'). It is important, that the output values need to be compatible with the constructor of the mining model. The transformation logic should be written in the DataframeTransformations class.

The Mining model has to either inherit from the MiningInterface or the BaseMining class. The Models need to have getters for all the parameters, and the method to perform the mining has to store a graph of type BaseGraph in the self.graph variable.

To add the algorithm to the page, it needs to be added in the ‘config.py‘file. The algorithm_mappings map the name of the algorithm to the route, the algorithm_routes map the route to the controller class. Both dictionaries need to add the new data, to make the algorithm usable. Optionally, a documentation page can be added to the mining algorithm. This should be written in a markdown file and stored in the docs/algorithms directory. To add the documentation, the docs path needs to be added to the docs_path_mappings dictionary, the key is the route of the algorithms and the value the path to the file.

5.5.2 Extending the Application

There are more ways to extend the application. These are mentioned in the GitHub repository in the Contributing.md file in great detail. This section will only mention other extension options that were considered. Future contributors can add pages or only adapt the Views of existing pages. Extending the ColumnSelection view is one use case, to allow more input data from logs. Also extending or updating models will be easily possible due to the MVC pattern.

6 Evaluation and Discussion

This section will discuss challenges encountered during the project, improvements to the Process Mining Visualization Tool and learned experiences. First, the limitations of streamlit UI library will be discussed. Afterward, the encountered problems will be mentioned, directly followed by the project improvements. the last part will summarize what has been learned during the implementation process.

6.1 Limitations of Streamlit

Streamlit [20] is an easy-to-use UI framework. No frontend knowledge is needed to build web apps in python. Streamlit renders a page, by running the script from top to bottom, if an event is triggered the page is completely re-rendered, with an updated value for the input widget.

Streamlit does only have limited styling options. The color theme can be changed in the streamlit config, but no further styling is possible. Colors of elements can not be changed, the only exception are buttons, there do exist two button types, primary and secondary. Text can use a few predefined colors. Natively, the styling is limited. The layout options for pages are also sparse and rather simple. Sections, Columns, and the sidebar are supported as layout options, but more complex options, like grids, are not supported. The position of an element is also determined by the position in the script.

Performance can be an issue for streamlit applications. As previously described, the script is completely rerun at every user interaction. Smaller tasks will not influence the performance significantly, but resource intensive computations will.

6.2 Challenges

This section will outline some of the challenges encountered during the project implementation. More problem can be found in the issues of the GitHub repository.

The main challenge was the interactive graph component. It needed to be written in React using typescript. The d3-graphviz[9] library was used to create the interactive graph. Creating the graph and making it interactive, with node click listeners, was easy. This basic use-case is well documented. Larger graphs took significantly longer to render and were blocking the website. This made the application unusable for larger datasets. The documentation had a few tips to increase the performance, by disabling graph rendering options. These changes helped slightly, but the calculations did still block the website. By using Web workers, the graph was rendered in the background and the main website was no longer frozen. This made the application usable again. This solution could be found in the documentation, but was well hidden, under a different section. Another problem was destroying an outdated renderer. The documentation did explain it, but the Typescript interface did not support deleting the instance, and unsafe type conversions needed to be made.

Another challenge was already mentioned in section 6.1. The position of a UI element is equal to the order of execution. Out-of-order rendering seems rather unimportant, but it has a few use-cases. An example is the Export page, reading larger pngs can take time, and would block the rendering of the "back" button. Users would be forced to wait until the image is generated, to go back. Elements can be rendered out of order, if containers are rendered first in the correct order. The elements need to be attached to the container.

Performance is also a limitation of streamlit, as mentioned in section6.1. To improve the performance was also one of the main problems encountered. This was achieved by caching the model in the session state and only recompute process models, when parameter have changed.

A few problems with the session state were also encountered. Should the session state be bound to a widget, by the key argument, the session state can no longer be changed manually. This can be a problem, when an error occurs and the session state value should be reset to a default value. Also, some elements, like the file uploader, do not allow setting a value before the widget is created. Due to this, files need to be re-uploaded after leaving the page.

While implementing the inductive miner, the runtime was also a challenge with larger datasets. Cut calculations needed to be improved, the sequence cut the most. Instead of calculating for each pair of nodes if they are pairwise reachable, reachable nodes are only calculated once for each node. The results are stored and can be reused to check the pairwise reachability. Also improving the graph datastructure for the specific use-case improved the runtime, by switching from a node and edge list approach to an adjacency list approach, to achieve better runtimes for graph traversal.

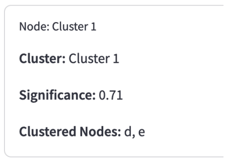
6.3 Project Improvements

This section will explain the advantages of the updated Process Mining tool, with respect to the previous implementation. The previous project can be found at this link ht-tps://github.com/fabianf00/ProcessMiningVisualization_WS23/tree/Old_Implementation

Firstly, a newer, more moderns, and not outdated UI library was chosen. The previous implementation used the PyQt5 library, that already has a newer version. The UI looks up-to-date and is responsive. Because a Web UI library is used, there is no need to run a web server in the background, to display an interactive graph. Also, the node information display has drastically improved, and more information is shown, previously only the node was shown. Figure 6.1 shown the current node information display, and Figure 6.2 shows the previous version.

As it can be seen in these figures, the display of node information has been improved. Now a lot more data about a node is shown.

Both Implementations use the MVC pattern, but the previous project did only utilize


Node: Cluster 1

Cluster: Cluster 1

Significance: 0.71

Clustered Nodes: d, e

Figure 6.1: Fuzzy Mining Cluster Node Information

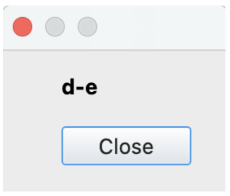

d-e

Figure 6.2: Fuzzy Mining Cluster Node Information of the previous implementation

it for the mining algorithms, all other views did not have a controller or models and the logic was in the view class. This is a major improvement in this project, as described in section 4.2. Every page uses the MVC pattern and models are reused. This decreases code duplication and make the code easier extensible and maintainable.

The lines of code to add new algorithms has drastically reduced. Table **??** shows the lines of code for the controllers and views in the previous and current implementation. To add the UI for the heuristic mining algorithm, 162 lines were needed, now only 58 lines are needed. For the fuzzy miner, 219 lines were needed, and now only 84. It can be seen, that the implementation now needs more line of code to create the controller. This could have multiple reasons. First, the controller now has more responsibilities, as some of the code has been moved from the views to the controllers. Secondly, a formatter was used during development, that splits a long line in multiple short ones.

	Previous Implementation	Current Implementation
Heuristic View	134	21
Heuristic Controller	28	37
Fuzzy View	187	37
Fuzzy Controller	32	47

Table 6.1: Line of Codes in class files

The overall lines of code to add new algorithms has decreased, This is most likely due to the usage of base classes for the views and controllers. BaseAlgorithmView and BaseController classes contain common functionality. The previous implementation also had a base class, that only was an interface, without logic inside. The old Algorithm Views needed to create all components of the window, while the new Algorithm Views only need to modify the sidebar. Other elements of the page are implemented in the base class.

Changes were made to improve the performance, mostly for larger datasets. First, the event log structure has been changed. Logs no longer use a list as datastructure, where each trace is stored as a list. Now they are stored in a dictionary with the trace as key and the frequency as value. This change makes sure that each unique trace is only parsed once, instead of multiple times. Furthermore, NumPy's vector and matrix operations are used instead of python for loops. This helps to increase performance drastically for larger datasets. Also, display these larger graphs is now possible.

Error handling has significantly improved. Multiple error cases were taken in consideration, and custom exceptions and error messages were added to the application. All errors caught and the application no longer crashed, when an error occurs, which drastically increased the stability of the tool.

The application is now more future-proof and extensible. A newer UI library is used to create a more modern looking UI. The MVC pattern is used to make the project more extensible and maintainable and because of the modular structure and the separation of concerns, components can be easily replaced or updated. As section 5.5 mentioned, there are more ways to extend the project in the future.

6.4 Lessons Learned

This section will summarize what I learned during the development of this project. Due to this project, I learned a lot regarding design decisions, project architectures, and project structures. Design decisions should never be final and always be reevaluated to fit the use case. This process may increase the work, but is important to create qualitatively good projects. The architecture of a project should be carefully evaluated to fit current and future needs and adapted if necessary. Also, the project structure is important. A modular project structure increases the readability and discoverability of code. This makes code easier reusable and open for extensions. I also learned the importance of testing to find errors in the code more easily and make the code more secure for internal changes. This project also showed me the importance of use-case specific data structures to improve the performance. Overall, due to this project, I gained a lot of important experiences.

7 Conclusion and Future Work

In conclusion, this thesis brought a lot of important improvements to the open source Process Mining Visualization Tool. The tool can be found on in the following GitHub repository: https://github.com/fabianf00/ProcessMiningVisualization_WS23. The new UI library, streamlit, modernized the UI and the new mining algorithm adds an important process model view to the tool, by showing exclusive, sequence, parallel and loop relationships. Streamlit was the right choice of UI library, as it is easy to use and can be used by developers without frontend experience. The library works great even for bigger applications and is performant, when caching is used. The currently supported algorithms, heuristic miner, fuzzy miner, and inductive miner, are all very important for process discovery algorithms. Furthermore, the design and architecture make the tool more future-proof for extension, by providing many different interfaces.

Due to the open source nature of the project, there are many possible extensions for future work possible. Only a few will be mentioned here.

The most important extension is adding new algorithms to the tool. Existing algorithms can be updated, by adding new metrics or new variants of mining algorithms, e.g. infrequent inductive miner. This can further enhance the analytic capabilities of the tool.

Adding or improving features will increase the value of the application. The usage of an AI model could also be used to improve the prediction of columns. Analyzing logs and showing statistics, like other tools already do, could also be added. Another Process Mining technique, e.g. Conformance checking or finding bottlenecks, could also broaden the capabilities of the tool.

Graph interactions could be improved, by further enhancing node information or by adding edge interactivity to the graph. Updating and standardizing the process models could also improve the consistency across the tool. Improving the rendering performance of larger interactive graphs would also make the tool more usable.

This project has laid a new foundation for the future of the Process Mining Visualization Tool in the open source process mining community and will allow smaller companies to use process mining, without financial burdens.

Bibliography

[1] Bostock, M., and Observable, I. The javascript library for bespoke data visualization. `https://d3js.org/`, 2024. Accessed on 2024-06-26.

[2] Celonis. Process mining und execution management software | celonis. `https://www.celonis.com/de/`, 2024. Accessed on 2024-07-13.

[3] Chen, A. *A Python Desktop App for Business. Process Mining and Visualization.* Bachelor thesis, University of Vienna, 2023.

[4] Detwiler, B. What is object-centric process mining? `https://www.celonis.com/blog/what-is-object-centric-process-mining-ocpm/`, 2022. Accessed on 2024-07-20.

[5] Fluxicon. Process mining and automated process discovery software for professionals - fluxicon disco. `https://fluxicon.com/disco/`, 2024. Accessed on 2024-07-13.

[6] Graphviz. Graphviz. `https://graphviz.org/`, 2021. Accessed on 2024-06-26.

[7] Grove, R. F., and Ozkan, E. The mvc-web design pattern. In *Proceedings of the 7th International Conference on Web Information Systems and Technologies - Volume 1: WEBIST,*. SciTePress, 2011, pp. 127–130.

[8] Günther, C., and Aalst, W. Fuzzy mining – adaptive process simplification based on multi-perspective metrics. In *Business Process Management.* Springer Berlin Heidelberg, 2007, pp. 328–343.

[9] Jacobsson, M. d3-graphviz. `https://www.npmjs.com/package/d3-graphviz`, 2024. Accessed on 2024-06-26.

[10] Krasniqi, A. *Fuzzy Miner as an additional Process Mining Algorithm for a Process Mining Visualization Tool in Python.* Bachelor thesis, University of Vienna, 2024.

[11] Leemans, S. *Robust process mining with guarantees.* PhD thesis, Technische Universiteit Eindhoven, 2017.

[12] Leemans, S. J. J., Fahland, D., and van der Aalst, W. M. P. Discovering block-structured process models from event logs - a constructive approach. In *Application and Theory of Petri Nets and Concurrency*, J.-M. Colom and J. Desel, Eds. Springer Berlin Heidelberg, Berlin, Heidelberg, 2013, pp. 311–329.

[13] Lux, M. ddcal. `https://pypi.org/project/ddcal/`, 2023. Accessed on 2024-06-27.

[14] NumFOCUS-Inc. Pandas. `https://pandas.pydata.org/`, 2024. Accessed on 2024-06-27.

[15] NumPy-team. Numpy. `https://numpy.org/`, 2024. Accessed on 2024-06-27.

[16] ProcessIntelligenceSolutions. Process intelligence solutions. `https://processintelligence.solutions/`, 2024. Accessed on 2024-07-13.

[17] ProMTools. Prom tools. `https://promtools.org/`, 2024. Accessed on 2024-07-13.

[18] Rana, M. E., and Saleh, O. S. Chapter 15 - high assurance software architecture and design. In *System Assurances*, P. Johri, A. Anand, J. Vain, J. Singh, and M. Quasim, Eds., Emerging Methodologies and Applications in Modelling. Academic Press, 2022, pp. 271–285.

[19] Streamlit. Custom components. `https://docs.streamlit.io/develop/concepts/custom-components`, 2024. Accessed on 2024-06-26.

[20] Streamlit. Streamlit. `https://streamlit.io/`, 2024. Accessed on 2024-06-26.

[21] van der Aalst, W. M. P. *Process Mining Discovery, Conformance and Enhancement of Business Processes.* Springer Verlag Berlin Heidelberg, 2011.