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 from event logs and creating process models is one discipline of process mining. Organizations gain valuable insight into 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 of functionality is possible. The open source tool, Prom 6, can be used free of charge and has a wide variety of algorithms to use, at the cost of usability. The open source tool needs professional knowledge to be used. An 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 the heuristic and fuzzy miner, allows easy extensions 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 increases the usability, and ensures longer compatibility in the future. The updated UI improves the appearance of the tool, but also adds more details about the events in the graph. The inductive miner adds new valuable insights into processes 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 modular approach to ensure a cleaner code structure. The architecture has been reworked, by using the MVC pattern across all pages, to separate logic and UI clearly. These changes made the code better reusable and also reduced the lines of code required to add new algorithms. The performance improvements reduced loading times inside the application and also allowed the mining of larger datasets in a reasonable time. Error handling has been improved to prevent crashes of the application. Due to these changes, this tool's stability 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 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

Being able to understand complex processes, or verifying that the process is done correctly, are valuable information for small and large companies. These are the main challenges of process mining and show how important this data is. Process mining tools can gain these valuable insights regarding a process, by transforming logs into a process model. Users can see a visual representation of their processes and understand complex processes easier. This technique is called process discovery. These tools are important for every sector to better understand their processes or to find error in the execution of their processes

There do exist a few of these tools, like PMTK[17], Disco[5], Celonis[2], and ProM 6[18], but most of them do need a license. All the previously mentioned tools except ProM 6 are not open source tools and a paid license is needed to use them. This can be financial challenging, especially for smaller companies or academic institutions, and limit access to such important tools and its features. Most of the commercial tool come with additional 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 mostly used for academic purposes. Extending proprietary software is also most of the time not possible. More about these tools will be mentioned in section 2.1.

Due to these limitations of existing tools, the Process Mining Visualization Tool[3, 10] has been created. The project tries to fix the problems encountered in current process mining tools. It is easy to use, open source, easily extensible and users can choose the used algorithms. The current main problems are the user interface, the lack of currently supported algorithms, the lack of features, bugs, and performance issues. More about this tool will be mentioned in section 2.2.

The motivation of this thesis is to improve the Process Mining Visualization Tool further, by tackling a few of these limitations. First, the user interface will be reworked, by changing the outdated desktop UI library to a newer web UI library. Afterward, the inductive miner will be added to the tool to enhance the analytic capabilities of it. Additionally, the project structure will be reworked, to make adding new algorithms easier. The last changes will include bug fixes and performance improvements.

The goal of this project is to enhance the Process Mining Visualization Tool, to increase the usability and the analytic capabilities. This open source tool should convince, with its ease of use, a modern UI, and easy extensibility, the open source process mining community.

2 Related Work

Process mining is an important discipline in data mining, with three main challenges, process discovery, conformance checking, and enhancement. It aims to give stakeholders a better insight into complex processes.[11, 12, 23]

Process discovery aims to create a process model from event logs. It visualizes the process to create a better understanding about the workflow, by creating Petri nets, BPMN models or other graphs. Many different process discovery algorithms exist, each with a different goal. There do exist algorithms, 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, 23]. Mined models and the visualization of processes depend on the algorithms used. Some only create directly-follows models and others create more complex BPMN models.[11, 12, 23]

Conformance checking tries to find anomalies between a log and a process model, or between two process models. For this discipline, a process model and a log of the process or two process models are needed. Evaluating discovered process models is one of the tasks 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, 23]

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, 23]

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[17], and ProM 6[18]. 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. Logs are stored in data pools. Celonis gets data by extracting data directly from sources, like SAP. The application provides a wide variety of features for business customers. Log

statistics can be shown, processes can be monitored and evaluated, and the conformance of processes can be checked. The main feature of Celonis is process discovery, the creation of process models from event logs. Celonis supports a wider variety of process discovery algorithms[22], e.g. heuristic miner and inductive miner, object-centric process mining[4], and the Business Miner, a proprietary mining algorithm that utilizes AI. Figure 2.1 shows a process model using their object-centric process mining algorithm.[2]

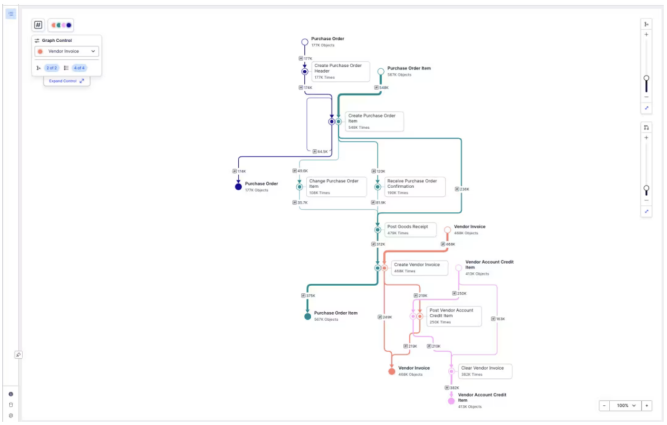

Figure 2.1: Celonis Process Model Graph[2]

Disco, developed by Fluxicon, is a process mining tool that requires a commercial license to use. Logs are imported directly into the tool as a CSV file. Disco uses a custom, closed source, implementation of the fuzzy miner and supports no other mining algorithms. Figure 2.2 shows a process model graph found by Disco. The tool has log analysis and case analysis functionalities and operates locally as a desktop application. However, it does not support custom algorithms or extensions, as this tool is closed source.[5]

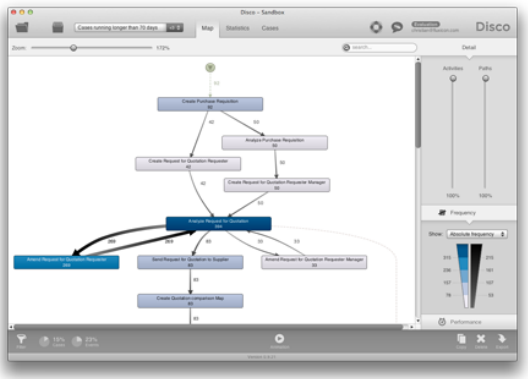

Figure 2.2: Disco Process Model Graph[5]

PMTK is a newer process mining tool, but also needs a paid license to use it. Logs are

uploaded to a workspace, as CSV files. The columns needed for the algorithms need to be chosen directly after the upload. 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.[17] Figure 2.3 shows the PMTK tool, with their directly-follows miner.

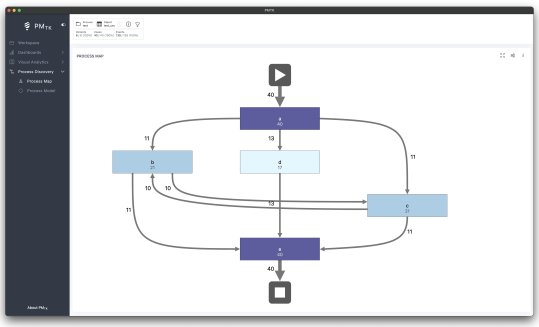

Figure 2.3: PMTK Process Mapping Graph

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, and many tasks can be performed on these log files, like log analysis, log filtering, mining algorithms, or conformance checking. Log files have to be added to a workspace, by importing them as XES or CSV file. Actions that are performed on a log file, 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.[18]

Figure 2.4: Prom 6 Action View [18]

As this section showed, there do exist many different process mining tools, but each of these have their own problems. Proprietary tools are widely used, but lack support for customization and variety of different mining algorithms. Users often have no influence on the used algorithms. These tools can also become a financial burden for smaller

organizations. The most known open source process mining tool, ProM 6, fixes the main challenges of commercial tools, but at the cost of usability and ease of use.

2.2 Process Mining Visualization Tool

The Process Mining Visualization Tool wants to fix these challenges. It was created and extended in two theses[3, 10]. It aims to be an open source tool, that convinces with its ease of use, a broader range of algorithms and easy extensibility. Currently, only process discovery and enhancement of the process models, with frequency or other metrics, are supported. Logs are imported as a CSV file and the process discovery algorithm can be chosen by the user. Two algorithms are currently supported, the heuristic miner[3] and the fuzzy miner[10], and further algorithms can be added. Additionally, exported process models can be imported again. The tool uses the outdated PyQt5 UI library to create a local desktop application. Figure 2.5 shows the heuristic graph view of the application.[3, 10]

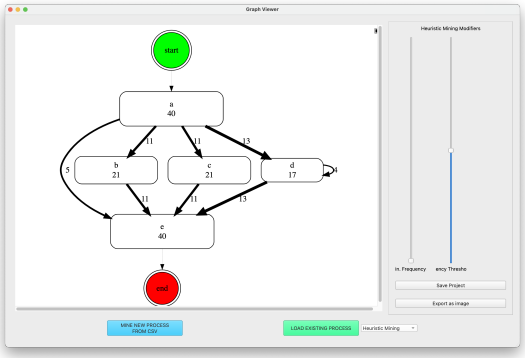

Figure 2.5: Heuristic Graph View

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 could also further be improved to ensure the future of this open source project and modernize the user interface.

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 between two nodes A and B exists, if the event A is directly followed by event B at least once in the event log. The start nodes are a collection of all nodes, that are the first event of a trace, while the end nodes are all the events 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, represented by tau, and all other nodes are operators. These operators may vary depending on the application. The inductive miner has a sequence, exclusive, parallel and loop operator. The tree shows the operations 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.

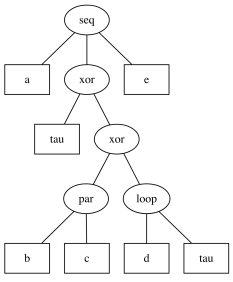

Figure 3.1: Process tree

3.2 Heuristic Miner

The heuristic miner is a simple, and effective process mining algorithm. It considers the frequencies of events and directly-follows relations during the creation of 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, 23]

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, which are then discarded.[3]

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.[3]

The dependency metric determines, how one-sided a relationship is between two edges. It compares how dependent an edge is by the formula shown in Figure 3.2. S(a $>$ b) represents the frequency of the edge (a, b) in the succession matrix.[3]

$if a \neq b:$

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

$if a = b:$

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

Figure 3.2: Formula for calculating the dependency metric[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 the 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 the threshold.

3.3 Fuzzy Miner

The fuzzy miner is a mining algorithm, that 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.[8, 10]

The algorithm 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.[8, 10]

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, or edge significance, is calculated by taking the source node's significance.

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.[10]

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

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

The edge cutoff is calculated by normalizing the utility value for edges with the same target node. The Min-Max normalization is used.[10]

3.3.2 Filtering

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

The significance value is 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 creates a directly-follows graph from it. By detecting cuts in the graph, exclusive executions, parallel executions, sequential executions, and loops can be found. The algorithm produces sound process models, meaning every trace of the input log is represented in the model.[12]

First, the algorithm checks for base cases, then it tries to find partitions in the directly-follows graphs. If a partition is 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 or an empty trace is in the log, 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.

```
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, tau, 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.[11, 12]

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 into 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, a 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.[11, 12]. This cut is found by calculating the connected components of the directly follows graph. Figure 3.5 shows an example of the cut. An exclusive cut has been found with two partitions. One partition contains the events a, b, c and the other contains the events d, e, f.

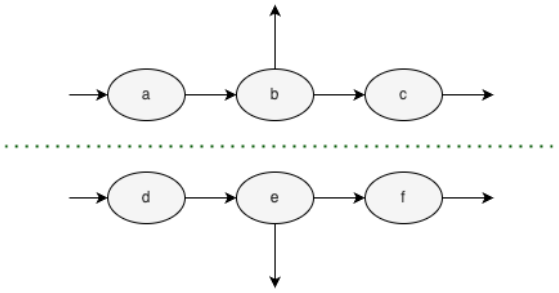

Figure 3.5: Exclusive Cut of a directly-follows graph

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 to the lower one. If only one partition was found, the algorithm checks for a parallel cut.[11, 12] 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 by their reachability. Figure 3.6 shows an example of the cut. There do exist 3 partitions, the first partition contains only the event a, the second partition contains the events b and c, and the third partition contains event d.

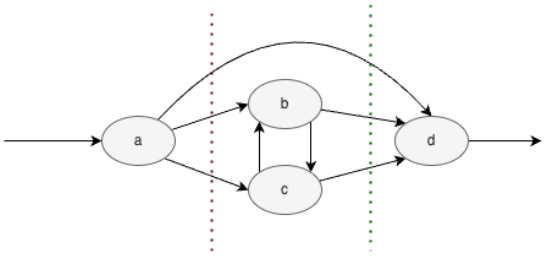

Figure 3.6: Sequence Cut of a directly-follows graph

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.[11, 12]. 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. Two partitions are found, one contains only event b and the second only contains event c.

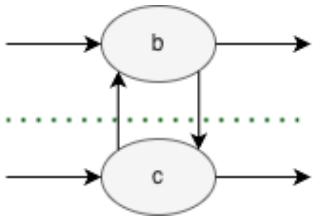

Figure 3.7: Parallel Cut of a directly-follows graph

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.[11, 12]. This cut finds its first partition, by adding all start and end activities to the partition. Then these activities are temporarily 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 containing the events a and b, and the other two partitions are redo parts, containing the events c and d, e.

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 multiple subtraces and each subtrace will be

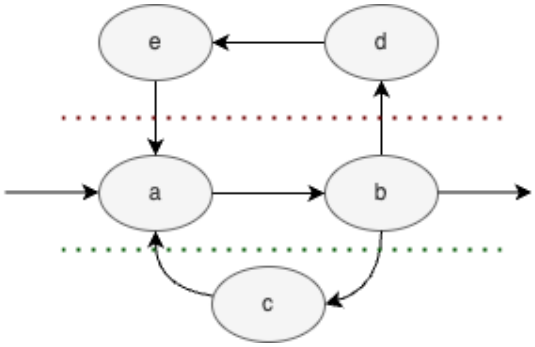

Figure 3.8: Loop Cut of a directly-follows graph

assigned to a sublog. The algorithm starts with the first partition and an empty subtrace. While an event is part of the partition, the event will be added to the subtrace and the next element of the trace is checked. Should the element not be part of the partition, then the subtrace will be added to the sublog of the partition and a new empty subtrace will be created. The algorithm advances to the next partition and checks the events as previously described. This is done, until the full trace has been traversed. Should a trace not contain any events of a partition, then an empty trace is added to the sublog of the partition.[12]

The parallel split does project each trace, for each partition. The projection 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 multiple subtraces and each subtrace will be assigned to a sublog. The algorithm starts with the first element and an empty subtrace. First, the partition of the current event needs to be found, and the event is added to the subtrace. Then the next event is checked. As long as the events are in the same partition, they are added to the subtrace. If an event is not in the partition, the subtrace will be added to the sublog of the partition, an empty subtrace 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 contains only one event, but it is executed more than once. 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 all 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. First, the UI library choice will be discussed. 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[21] library was one requirement of the thesis, but this UI library has many advantages over other UI libraries. Streamlit is a web UI library that is easy to use for python developers, mostly used to create prototypes, chatbots or dashboard. No JavaScript/frontend knowledge is needed to create modern, responsive, and interactive pages. Streamlit is used by a lot of developers, has great community support, and many custom components written by the community. Another well known web UI library is dash[16], but dash needs HTML and CSS knowledge to create an application.

Using streamlit, an application can be written like a python script. Streamlit renders pages by running a python file from top to bottom. The file can be a simple python script or a more complex application that uses classes. Input widgets return their input value directly, or in the session state. If users interact with an input widget, the whole page is rerun and the returned value of the input element is updated. Writing a web application in streamlit is easy for python developers and data scientist, which is the main advantage of this library.

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 user interactions. Events from users, e.g. button presses, input field changes, are passed to the controller. The controller processes events from the view, passes the data to the correct model, and sends the updated data to the view. Figure 4.1 shows this relationship between these components.[7, 19]

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 to demonstrate the relationships between the components.

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

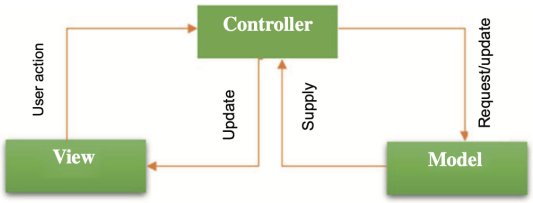

Figure 4.1: Model View Controller Pattern[19]

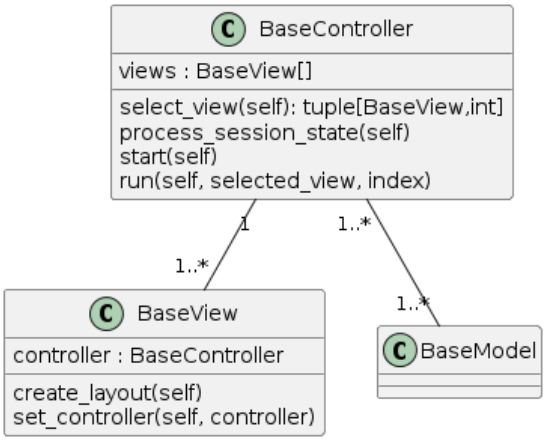

Figure 4.2: UML Diagram MVC Pattern

views, to support different representations in the future, but currently each page has only one view. The views only display the data and pass events to the controller. The data is received by the controller. Controllers can have multiple views and models. The controller is the entry point for each page. It reads data from the session state, events are stored in the session state in streamlit[21], 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, and error handling is a responsibility of 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 detailed description of the MVC pattern shows, that the application uses the pattern as shown in Figure 4.1

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 independent 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.[7, 19]

4.3 Custom Graphs

The project uses a custom graph class. This class stores the calculated process graph, and all node information. Graphviz[6] is used to visualize the graph, because 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 visualize the graphs and to easily export them to PNGs or SVGs. The graphviz graph will only be used to visualize the graphs. The custom graph class will store nodes and edges independently of graphviz, to make testing easier. Through the custom class, connections can be checked and the existence of nodes can be verified. The nodes can also store additional node information, that can later be used in the interactive graph to show this information again. In the current state of the application this feature may not be necessary, but for future algorithms additional node data may be useful. Figure 4.3 shows a simple UML diagram of the BaseGraph class.

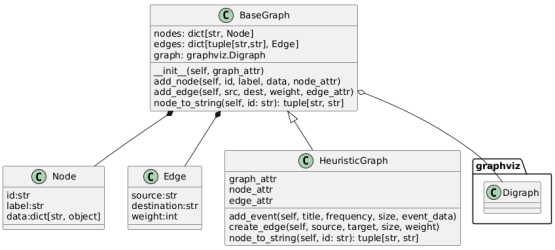

Figure 4.3: Custom Graph UML Diagram

The BaseGraph will be the base class for all future graph visualization classes. The graph stores nodes and edges inside of dictionaries. Nodes have an id, a label, and can hold additional information in the data dictionary. Edges only have a source node, a destination node and a weight. A custom graph can have global styling for nodes, edges or the whole graph. This ensures a consistent styling across the graph and abstracts the creation of new nodes and edges, as the styling is hidden from the outside. The class uses a graphviz.Digraph instance to visualize the graph. Each graph should have a custom method to generate a node information title and node information details from the node id. These representations should be changed for each mining algorithm. New custom graph classes can be created, by inheriting from the base class. These classes can define a classwide style and add simpler methods to add nodes and edges. Internally, they still should use the BaseGraph methods to add nodes and edges.

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 them 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[20] can be made using the web framework React or Typescript.

The interactive Graph component uses the Streamlit Components API[20]. Two components need to be written, a python component and a React component. The python component is used inside the application and communicates with the React component.

The python component is a function that takes in a BaseGraph instance, a function that is called when a node is clicked, a key, and a height. The height and a graphviz string of the graph will be passed to the React component. If a node has been clicked, the React component will return the node_id of the clicked node. The component will then use the node_to_string method of the BaseGraph to get a title and a description of the node. The function passed to the python component will be called with the title and node description as arguments.

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 and supports rendering interactive graphviz graphs with d3-graphviz, which was the main factor for choosing these libraries. The graph rendering is done in the background using web workers, to prevent the freezing of the page during the rendering process. After the graph has been rendered, on-Click-Listeners are registered on all the nodes. If a node is clicked, the component return the node_id and a click_id to the python component. The click_id is needed to recognize if a 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 React component will re-render whenever the size of the window or the graphviz string changes.

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 main file is the `streamlit_app.py` file containing all pages and their routes, `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 `components` 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 `templates` module.

The `docs` directory contains all documentation files. These include UML models of the project, current and prior, and all the theses about the Process Mining Visualization Tool. Also, the documentations of the mining algorithms should be stored there in a separate folder. Presentations of the theses can also be stored in their own directory.

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, e.g. models that are responsible for importing and exporting data are stored in the `io_operations` module.

5.2 Requirements

This part will mention the libraries used in the project.

Streamlit[21] 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 found 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.

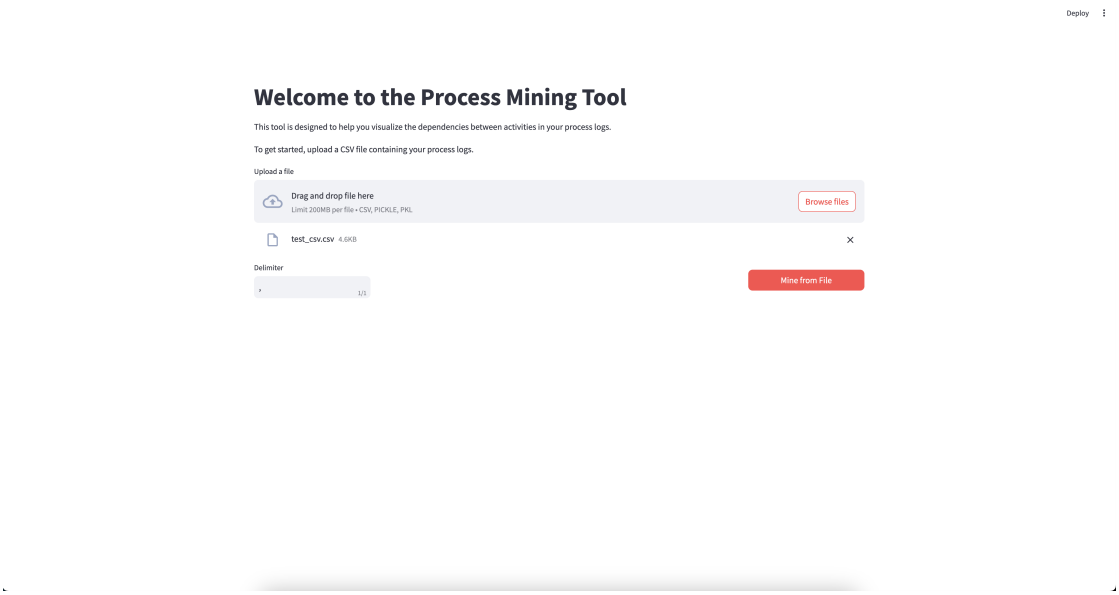

Figure 5.1: Home page

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 field is shown. In this case, or if the detected delimiter is incorrect, the user can manually set the delimiter. By pressing the "Mine from file" button, the CSV file is read, the dataframe is build and passed to the Column Selection page via session state, mentioned in section 5.3.2.

If the user uploads a pickle file, a drop-down 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 Column Selection page shows the dataframe and lets users select the needed columns. The needed columns may depend on the chosen mining algorithm. With the columns, an event log will be built for the mining algorithms. Users have to choose the mining algorithm manually by a drop-down selection. The page is shown in Figure 5.2

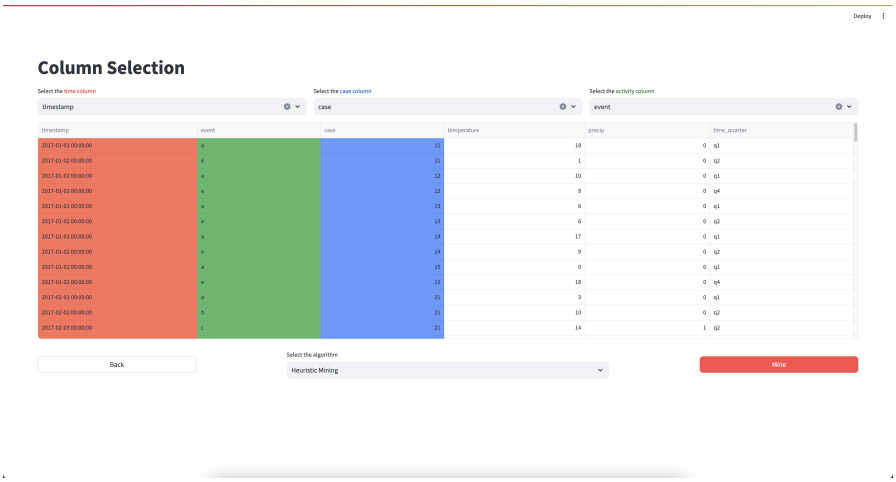

Figure 5.2: Column Selection page

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 drop-down menu. The drop-down menu at the bottom of the page lets users choose the algorithm. 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. Due to the similarities of all Mining Algorithm pages, only one will be shown. By pressing the "Export" button, users will be redirected to the Export page, mentioned in 5.3.4.

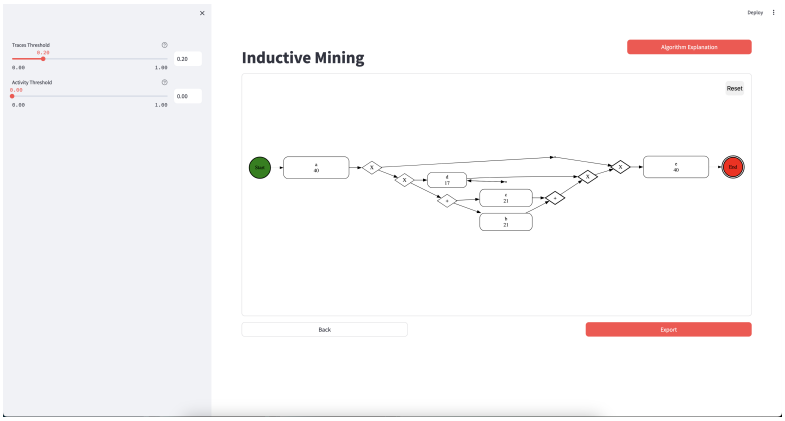

Figure 5.3: Inductive Mining Algorithm page

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, optional 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 node information.

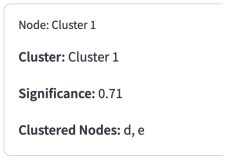

Figure 5.4: Fuzzy Mining Cluster Node Information

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 does exist a drop-down menu to choose an export format, DOT, PNG, and 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.

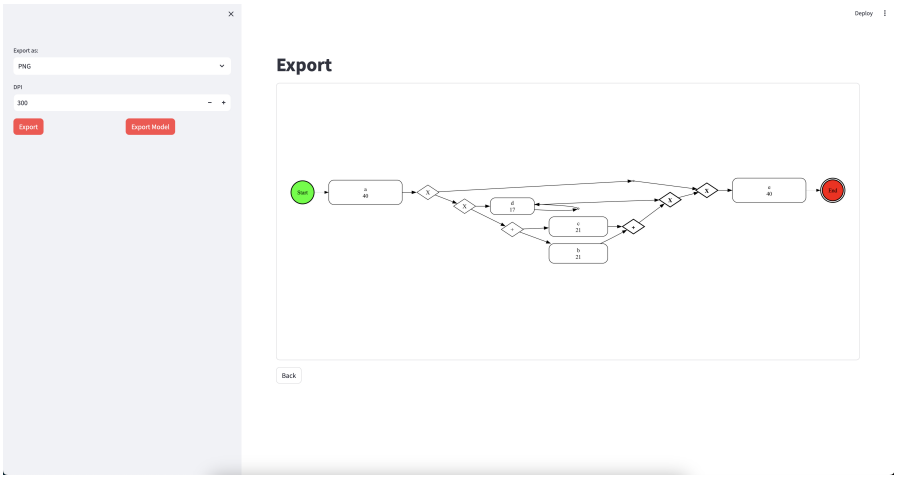

Figure 5.5: Export page

5.4 Testing

Testing is an important step in the software development process, to ensure the correctness of the implementation. To follow 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. All parts of the inductive miner have been tests, the algorithm as a whole and all its components. The dfg data structure, that is used in the algorithm, the cuts of the algorithm, and the log splitting logic have all been fully tested. The algorithm has also been tested as a whole, to ensure, that base cases and fall throughs are applied correctly.

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 implementation, the graph of the test_csv dataset were compared with the process mining application PMTK[17]. Figure5.6 shows the graph of the implementation, while Figure 5.7 shows the process model from PMTK.

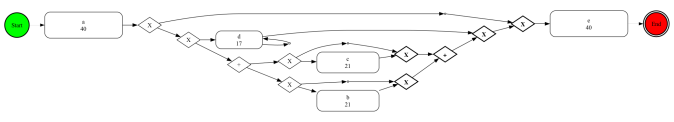

Figure 5.6: Inductive Mining Graph using the test_csv dataset

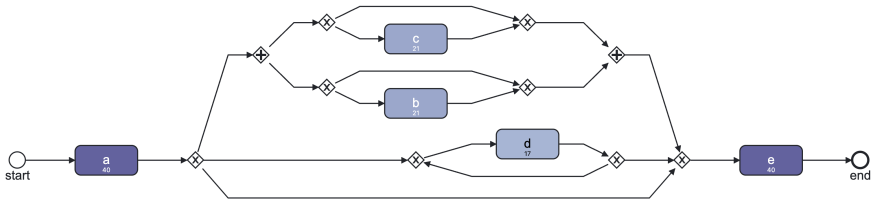

Figure 5.7: PMTK Inductive Mining Graph using the test_csv dataset

These figures prove, that the implementation is correct, as the process models show the same relationships between the events. These models show the same process model, but represent some relations differently. PMTK does not show silent activities, they are represented by an edge. Consecutive exclusive choices are merged to one exclusive choice, and the loop is represented with an exclusive gate.

5.5 Working on the project

This section explains how to extend the project. There are many extension options, but only a few of these will be covered. This section will explain how to add a new page, how to add new views to already existing pages, how to add new column selections, and how to add a new mining algorithm. Other ways to extend the application will be mentioned, but only briefly.

5.5.1 Adding a page

The project uses the MVC pattern to build pages. At least 2 classes need to be written, a view class and a controller class. These classes should be stored in its own folder inside the `ui` package. The logic for a page should be store in models. These need to be created, or existing models can be reused. Views are only responsible for displaying the page. The controller decides what part of the view or what view is displayed, processes input events, e.g. button clicks or value changes, calls the models and passes data to the view class to display it. It is recommended to use the template stored in `templates/ui_template` to create a new page.

The run method inside the controller is run at every reload, the process_session_state method should be used to either read values from the session state or to set values in the session state. select_view is an optional method and only needs to be implemented when more than one view is used for a page, the process_session_state method is called before the run and select_view method. Variables set in this method can be directly used in the select_view and run method.

It is recommended to have a function for each section that should be displayed, inside the view class. The create_layout method can be uses to reserve space for a section and

render these sections out of order.

To add the page to the application, the controller needs to be called in the `streamlit_app.py` file, with a route.

5.5.2 Adding a View to a page

If a new view for a page is added, the view should be stored in the same folder as the page views and the controller. The view should either be a subclass of the BaseView class or directly inherit from the page view class. The switch between views the select_view method has to be implemented. If the two views do not have the same methods, the run methods also need to be changed to support the new view. One use case will be adding a ColumnSelectionView. This case will be mentioned here, due to its importance.

5.5.3 Adding new column selections

Adding new column selections may be necessary in the future, for some algorithms, e.g. a resource column, a person column. To add these columns, there are two templates in the templates/column_selection_template. The BaseColumnSelectionViewTemplate does not contain any columns at all, and all columns need to be added by the new class. The ExtendedColumnSelectionViewTemplate contains the time_column, case_column and event_column/activity_column. The needed_columns and column_styles need to be added with additional data. The method render_column_selections needs to add a selection box with a key equal to the needed_column's name (e.g. person_column), so that the controller can assign the value to the correct field. Additionally, the select_view method of the ColumnSelectionController needs to be updated, to switch to this view, if a specific algorithm has been chosen. To ensure the correct transformation for the mining algorithm, the method transform_df_to_log needs to be overridden, in the AlgorithmController of the new mining algorithms. Otherwise, the additional columns will be ignored, and the default transformation will be performed.

5.5.4 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 has to be 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. 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_column) 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 maps the name of the algorithm to the route, the algorithm_routes maps 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 doc's 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.

5.5.5 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. The functionality of models can also be extended, new import and export formats can be added, and existing models can also be updated. The prediction model uses a dictionary to perform the prediction tasks. The dictionary could be updated, or the model could use AI to predict the columns of the input dataset.

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 the streamlit UI library will be discussed. Afterward, the encountered problems during the development will be mentioned. The third section will mention the improvements of the application in comparison to the previous implementation. The last part will summarize what has been learned during the development of this project.

6.1 Limitations of Streamlit

Streamlit [21] is an easy-to-use UI framework, that comes with a few limitations, that will be discussed here.

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 and texts 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 order of execution in the script. More of this problem will be mentioned in section 6.2.

The session state also has its restrictions. Session state variables can be bound, by the key parameter, to input widgets. This sets the output of the widget to the current value of the session state. After the variable is bound, only input widgets are allowed to change the value. After the destruction of the widget, the variable only exists for one rerun and will be automatically deleted afterward. The state of some widgets can not be set before creation of the input element, like buttons or file uploaders. The encountered problems due to these limitations will be mentioned in section 6.2.

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 also be rerun, which can decrease the performance of the app.

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. First, challenges encountered because of streamlit will be mentioned, problems with the interactive graph

component will be mentioned afterward, and challenges regarding the performance will be discussed last.

Like previously mentioned, the layout options are rather limited, and the position of an element is defined by the order of execution. There does not exist a way to set the position of an element manually. This increases the difficulty to render back buttons before rendering resource intensive elements, like the interactive graph, which would not allow users to leave the page until the process is finished. To fix this problem, containers are rendered in the preferred order and the elements are attached to the containers. This way, the back button can be loaded, before the graph has finished rendering.

The performance of the page rendering process was also a major challenge to keep the page responsive and prevent unnecessary calculations. To solve this problem, the mining models are cached in the session state and will only generate a new process graph, when input parameters change.

Due to the restrictions of the session state, a few problems were encountered. Bound session state could not be set to a safe value in case of an error and needs to be done by the user. The deletion of bound session state makes it difficult to use on different pages. To prevent this issue, all bound session states, that are needed on other pages, are stored in a new session state variable created before navigating to the next page. An uploaded file can not be manually set for the file uploader, therefore after the home page has been left, the file needs to be re-uploaded again, even when the application directly redirects to the home page.

The most challenging part was the creation of the interactive graph component, or to be more precise, with the libraries d3[1] and d3-graphviz[9]. Rendering small graphs and the interaction with these was easy, but problems arose with bigger, more complex graphs. The rendering of larger graphs took a longer time and was first performed in the main thread, which froze the application. With the help of the documentation, the performance has increased, by changing rendering parameters and moving the calculations to a background thread. Outdated, but not finished, rendering instances were blocking new rendering processes and needed to be deleted manually. All of these solutions could be found in the documentation of d3-graphviz, although some of this information was well hidden. After the destruction of a rendering instance, the contents of the div needed to be cleared. This step was not mentioned in the documentation.

During the development of the inductive miner, performance issues with larger datasets arose. Cut calculations, especially the sequence cut, needed to improve to fix the problem. Now the sequence cut calculates the reachability of the nodes, stores the results in memory, and uses this information to check for the reachability of nodes. Furthermore, the internal structure of the dfg has been changed to use adjacency lists, instead of an edge list, as these are better suited for graph traversal tasks. These changes improved the runtime significantly.

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 the following link: https://github.com/fabianf00/ProcessMiningVisualization_WS23/tree/Old_Implementation

The UI looks more modern, up to date, and is more responsive due to the changed UI library. 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.

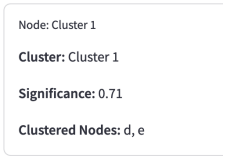

Figure 6.1: Fuzzy Mining Cluster Node Information

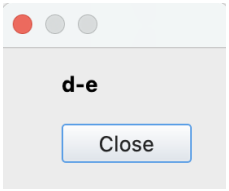

Figure 6.2: Fuzzy Mining Cluster Node Information of the previous implementation

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 it for the mining algorithms. All other pages did not have a controller or models, and the logic was written 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.

Adding new mining algorithms has been drastically improved. Due to the refactoring of the project and the updated architecture, fewer lines of code are needed to add new mining algorithms. Table 6.1 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 it only takes 58 lines. 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 change can be explained, by the increased responsibilities of the controller and usage of a code formatter, that splits longer lines in multiple short ones. The overall lines of

	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

code to add new algorithms has decreased. This improvement was achieved by the usage of the base classes, BaseAlgorithmView and BaseController, that implement common functionality for the views and controllers. The previous implementation also used base classes, but only as interface, without logic inside. The AlgorithmViews needed to be created from scratch, which resulted in a lot of duplicate code.

The performance of the tool was improved, and larger datasets can now be used. The event log is stored inside a dictionary instead of a list, which ensures that each unique trace is only processed once. Furthermore, NumPy's vector and matrix operations are used instead of python for loops. This helps to increase performance drastically for larger datasets. Also, displaying 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 are caught and the application no longer crashes, 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 extension. 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 that will help me in the future.

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, modernizes 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 as the 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, the heuristic miner, the fuzzy miner, and the inductive miner, are all very important process discovery algorithms, and give valuable insights into complex processes.

The new design and updated architecture increase the testability, the maintainability and the extensibility of the tool. 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, e.g. the alpha miner or object-centric process mining algorithms, to enhance the analytic capabilities of the tool. Existing algorithms can also be updated, by adding new metrics or new variants of mining algorithms, e.g. infrequent inductive miner.

Adding or improving features will increase the value of the application. The usage of an AI model to predict the needed columns for algorithms would make it easier to select columns. Analyzing logs and showing statistics would give valuable insights about the event logs and help to reduce the gap between other tools. Another process mining discipline, e.g. conformance checking, 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 more organizations to experience the advantages of 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 intelligence and process mining | celonis. https://www.celonis.com/, 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 van der Aalst, W. M. P. 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] Plotly. Dash documentation & user guide | plotly. `https://dash.plotly.com/`, 2024. Accessed on 2024-08-02.

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

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

[19] 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.

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

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

[22] van der Aalst, W. Process mining: Bridging not only data and processes, but also industry and academia. `https://www.celonis.com/blog/process-mining-bridging-not-only-data-and-processes-but-also-industry-and-academia/`, 2019. Accessed on 2024-07-31.

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