BACHELORARBEIT / BACHELOR'S THESIS

Titel der Bachelorarbeit / Title of the Bachelor's Thesis

„Process Mining Visualization Tool in Python"

verfasst von / submitted by

Argjend Rustemi

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:	Bachelorstudium Informatik
Betreut von / Supervisor:	Marian Lux, Dipl.-Ing. Dr.techn.

Acknowledgements

With the appropriate people's support and belief in you, I firmly think that great things in life are simply achievable. I consider myself very fortunate to have people of that caliber, and I would like to dedicate this section of my thesis to thanking them for being a part of this journey.

I want to start by expressing my profound gratitude to all of my professors at the university, specially Dipl.-Ing. Dr.techn. Marian Lux for giving me the chance to participate in this innovative and exciting project as well as for always being there for me when I needed him, supporting me throughout the completion of my Bachelor thesis, and continuously reviewing my progress.

I must express my gratitude to my fantastic colleagues who never shied away from discussing and offering comments on studies and projects that required more time and effort to complete. I firmly feel they were an integral part of our trip, and I will always be grateful to them. However, I also want to thank them for making the last four years a lot more enjoyable and for keeping me calm throughout the entire process.

Finally, but the most importantly, I want to thank my lovely family most of all for their unwavering support during this entire road, for always being there to offer me the proper mental support, and for giving me the time to accomplish my goals. Not only that, but they also gave me priceless counsel that would help me throughout the rest of my life. Without them at my side, I would not have been able to complete this adventure. Aside from being mine, this accomplishment also belongs to my entire family, who, in my opinion, contributed just as much to it as I did.

Thank you!

Abstract

The study of extracting and evaluating business processes from event logs is the focus of the developing field of process mining in computer science. Process discovery, which finds the real processes from logs; conformance verification, which assesses the found processes against pre-established models; and process enhancement, which seeks to enhance current processes based on log insights, are its three primary components. This bachelor's thesis focuses on creating and improving an open-source Python desktop process mining tool by utilising a variety of libraries to process event logs and utilise Graphviz to visualise the processes. With our programme, users can start process mining operations by importing event logs in CSV format. Process graphs are then generated, giving users a clear visual depiction of the processes that have been mined. Process discovery is the particular focus of our project. The tool initially supported the heuristic miner and fuzzy miner algorithms. One of my contributions was putting the Alpha miner algorithm (a key technique in process discovery) into practice. Furthermore, we improved the Heuristic miner by adding a new filter that was based on the search process model (spm) metric, as explained in Lux, Rinderle-Ma, and Preda's (2018) study [6]. We addressed important concerns pertaining to the tool's stability and usability in addition to algorithmic improvements. This involved troubleshooting and resolving issues that had previously caused problems when importing both large and small CSV datasets. Additionally, we included a feature that greatly increases the tool's versatility in handling various data types by letting users choose the delimiter when importing CSV files. By offering enhanced functionality for finding and evaluating process models from event logs, this study advances the practical use of process mining tools. The tool is now even more algorithmically capable and guarantees more flexible and dependable data handling, which makes it an invaluable tool to users trying to extract insights from their event logs and streamline their workflows. These enhancements let the tool better facilitate the investigation and comprehension of intricate processes, which helps users make wise decisions and promotes process improvements.

Contents

Contents

1 Motivation

It's truly motivating to work on an open-source tool that helps businesses mine their workflows. There is no denying the relevance of process mining in the business sector because it offers priceless insights into areas for improvement and operational efficiencies. Process mining's ability to improve business processes and results by turning huge amounts of event log data into actionable intelligence is what is driving its growth.

In his work[8], Professor Wil van der Aalst outlined the goal of process mining as "to discover, monitor, and improve real processes by extracting knowledge from event logs readily available in today's information systems."[8] Three main activities are included in this field: process enhancement, process discovery, and conformance checking.

Process discovery is the process of using an event log to create a process model that faithfully captures the behaviour seen in the log. In essence, it maps event logs onto process models that represent the actual processes recorded in the data using discovery algorithms. Understanding the real workflows inside a business requires taking this step.[1] [9]

In contrast, conformance checking contrasts the actions described in a predefined process model with the events documented in the event log. Finding similarities and differences between the modelled behaviour and the real behaviour is the aim here. This comparison aids in confirming the accuracy of process models and locating deviations that might point to problems with compliance or parts in need of improvement.[1]

The goal of process enhancement is to expand or enhance an already-existing process model by adding new data obtained from event logs. Using data from the real world to guide these improvements, this step concentrates on streamlining the process to improve its efficiency or modify it to meet evolving business requirements.[1]

The chance to work on the actual application of algorithms, more especially, the Alpha miner algorithm, was a major source of drive for me. This algorithm is the foundation of my practical work and is a key component of process discovery. By putting the Alpha algorithm into practice, I was able to gain a deeper understanding of the principles behind process discovery and improve my proficiency in this essential field of process mining.

The combination of theoretical understanding and real-world implementation highlights the significance and fun of developing an open-source process mining tool. My goal in enhancing the tool's process discovery capabilities was to give users more effective tools for examining and refining their business processes. In addition to making a valuable contribution to the field of process mining, this work has the potential to significantly advance the way businesses function and develop.

1.1 Problem Statement

Open-source tools have an important part in the process mining space by offering easily navigable and adaptable solutions that support companies in process analysis and improvement. However, there exist prospects to enhance the functionality and broaden the capabilities of the existing tool. The following areas in particular have been determined to need improvement:

1.1.1 Alpha Miner Algorithm Support

The tool does not presently have the Alpha miner algorithm, although it does support the Heuristic[4] and Fuzzy[3] miner algorithms. This algorithm will increase the tool's ability to provide numerous options for process discovery.

1.1.2 Improvement of the Heuristic Miner

The search process model (spm) metric filter is an essential element for enhancing the accuracy and size of the process models that are mined, but the heuristic miner is not equipped with it.

1.1.3 Issues with Data Import and Handling

Users encountered some challenges when importing event logs in CSV format, especially when handling very small and very large datasets. These challenges could lead to errors and disruptions, which could impair the overall user experience and dependability of process mining. Additionally, because users are unable to specify delimiters when importing CSV files, the tool's flexibility in supporting various data formats and user needs is restricted. To ensure a dependable and smooth process, the data import features of the tool need to be improved and debugged.

1.2 Goals of the work

This thesis was focused mainly with improving an open-source process mining tool's usability, dependability, and functionality. The following were the precise goals:

1.2.1 Apply the Alpha Miner Algorithm

Including the Alpha Miner Algorithm into the current tool was one of the primary objectives. Including this essential discovery algorithm would give users access to a wider range of tools for event log analysis.

1.2.2 Put the Heuristic Miner's SPM Metric Filter into Practice

The idea was to add the spm (search process model) metric filter to improve the heuristic miner's accuracy and resilience. This enhancement attempted to provide more accurate process models and was based on research by.[6]

1.2.3 Enhance Data Import and Handling

Addressing and resolving problems with importing event logs in CSV format, especially for big and small datasets, was another important goal. This required debugging and improving the tool's capacity to manage various data sizes error-free. Allowing users to specify delimiters when importing CSV files was another way to increase data handling flexibility. The goal of these upgrades was to guarantee a more dependable, effective, and user-friendly import procedure.

1.3 Process of the work and methods

Understanding the project's current status and its past accomplishments was necessary when I first started working on it. I carefully went through my colleagues' work in order to understand this. This involved running the application while navigating through the code in a code editor, reading their documentation, and carefully reviewing the codebase. I was able to get a clear picture of the project thanks to this thorough review, which I then documented and visualised.
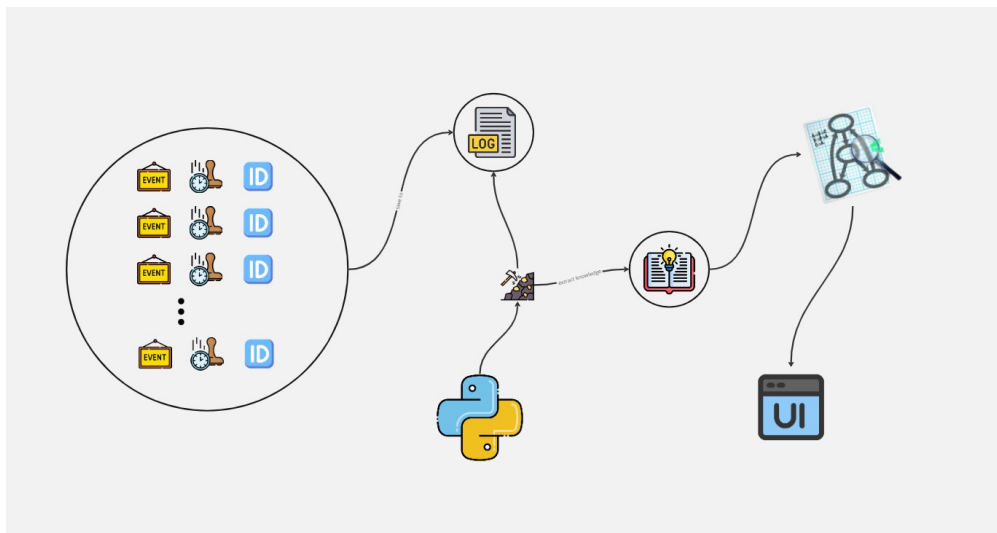

Figure 1.1: Diagram illustrating the project's scope and tasks, based on the initial review and understanding.

A thorough understanding of process mining was necessary for my work. After doing a tonne of research, I found this basic definition of process mining: "Process mining involves the extraction of knowledge from event logs." I was able to better understand the fundamentals of process mining thanks to this explanation, which also made the ensuing tasks easier.

I identified my three primary goals, which are detailed in the "Goals of the Work" section, after having a firm grasp of process mining. I then moved on to the most important aspect of the project, which was putting the Alpha miner algorithm into practice. I started by reading and researching the Alpha miner in great detail, with a focus on learning the eight key steps involved in putting the algorithm into practice. To confirm my understanding, I by hand solved a number of examples.

I then proceeded to improve the Heuristic miner. I began implementing this improvement after carefully reviewing relevant literature and evaluating the spm metric. My work ended with debugging and enhancing the data import features. Despite its apparent straightforwardness, this task took the longest because there were so many ways to optimise it and so many different approaches to choose from. On the other hand, the first two goals had objectives and steps that were more clearly defined.

I used PyCharm as my code editor and worked in Python for all of the implementation. I used GitHub for version control, following in the footsteps of my predecessors. In order to maintain the repository tidy and well-organized, I preferred working on different feature branches. It was simple to keep track of what was being added and what was being worked on with this method. It made it possible for my supervisor to effectively test implementations and keep an eye on progress. I integrated the modifications into the development branch after finishing each task.

This methodical approach made sure that every component was well tested and documented and that the project moved forward without hiccups.

2 Related Work

2.1 Analysis of State of the Art Literature

Through the analysis of event logs, process mining has grown as an important discipline for understanding and enhancing business processes. Leading researchers have made major contributions that have laid the foundation for many of the methods and tools that are used today.

- Wil van der Aalst created the Alpha algorithm, a fundamental method for process discovery that builds process models from event logs. It is described in detail in the work of van der Aalst and van Dongen (2013)[10], which focusses on finding Petri nets from event logs. It captures the fundamental behaviour of business processes.

- The Alpha algorithm has been re-examined in light of recent developments, such as the study by Küsters and van der Aalst (2023)[5], to increase its suitability for practical process discovery applications and guarantee that it is still effective and relevant in contemporary settings.

- The practical applications of the Alpha miner algorithm, as examined in[7], demonstrate the algorithm's adaptability to real-world situations. They specifically used Moodle, a popular learning management system (LMS), and the Alpha miner algorithm. All online learning activities are recorded by Moodle in event logs, which are stored in the database and can be used for process analysis to improve the quality of learning. However, Moodle event log data requires some preprocessing steps before applying process mining algorithms. In order to solve this, Nafasa and associates created a program that combines Moodle with tools for process mining, making the preprocessing necessary for successful mining of processes easier.

2.2 Comparison of existing similar approaches

Many of the tools and techniques used in the field of process mining are exclusive and commercial in nature. The goal of this project is to provide a strong, open-source substitute that uses advanced techniques like the Alpha algorithm and improvements to heuristic mining.

- Open-Source vs. Commercial Tools: Although commercial process mining tools like Disco offer extensive features, their high cost often prevents smaller organisations or individual researchers from utilising them. Delivering an open-source tool that

opens up access to sophisticated process mining capabilities is the main goal of this project. The project guarantees users can perform advanced process analysis at no expense by integrating the SPM metric into the heuristic miner and implementing the Alpha miner algorithm.

This initiative advances the field of process mining and increases the accessibility of these potent tools for a wider audience by merging these innovative techniques with an open-source approach.

3 Major Idea

3.1 Alpha Miner Algorithm

A key element of process mining, specifically in the context of process discovery, is the Alpha miner algorithm. According to Van der Aalst, one of the trickiest things about process mining is process discovery. It involves creating a process model from an event log by capturing the behaviour seen in the log. The Alpha miner algorithm is therefore essential to process mining because it offers a basic understanding of how process models can be found from unprocessed data.[1]

The Alpha miner algorithm is significant because it can be used as a process discovery introductory tool. Although this algorithm is the most basic in this field, it effectively demonstrates the fundamental concepts shared by numerous process mining algorithms[1]. It facilitates comprehension of the idea of process discovery and serves as a springboard for talking about the difficulties in this field. The Alpha miner algorithm's simplicity enables it to draw attention to important ideas that are hidden within more complex and reliable methods that are applied in real-world scenarios.

In other words, given a basic event log, the Alpha miner algorithm creates a Petri net that presumably can replay the log[1]. This is an example of a function, also referred to as the Play-in technique. The algorithm was among the initial ones that could effectively manage concurrency in process discovery. The Alpha miner algorithm's limitations with noise, irregular or incomplete behaviour, and complicated routing constructs make it difficult for use in real-world mining, it is important to remember. Nevertheless, because of its ease of use, it's a great starting point for talking about the difficulties and developments in process discovery algorithms.

There are four essential steps in the Alpha miner algorithm: choice, parallel, causality, and direct succession.(1,7) By following these procedures, you can build a process model that accurately depicts the behaviour you've seen and determine the connections between the events in the log.

We will examine each of these four steps in more detail in the following subsections, illustrating each one with an example that shows how it fits into the overall process discovery. An broad understanding of the Alpha miner algorithm and its function in process mining will be possible through this thorough investigation. Although we will talk about these steps here, the implementation section will go into more detail about how the algorithm is actually implemented.

Using the same log that was used to test the application while implementing the Alpha miner, we will use it as a test case for the next section, in which we will go over all four

steps.

$$L = [(a, b, c, d, e, f), (a, c, b, d, f, e), (a, d, e), (a, b, b, d)]$$

is the representation of the log. This log is brief for the sake of clarity.

3.1.1 Direct Succession

The term "direct succession," represented as "x>y," refers to the situation where event x in the event log is immediately followed by event y. The process model's activities' sequential dependencies can be discovered due to this relationship.

```python
def direct_succession (self):

    direct_succession = []
    for case in self.cases:
        for i in range(len(case) - 1):
            x = case[i]
            y = case[i + 1]
            direct_succession.append((x, y))

    return set(direct_succession)
```

Listing 3.1: Python implementation of direct succession

Following the application of the Direct Succession analysis to the event log L, we were able to determine the following relationships, in which an event within the same case follows another directly:

>L
(a,b)
(b,b)
(b,d)
(a,b)
(b,c)
(c,d)
(d,e)
(e,f)
(a,c)
(c,b)
(b,d)
(d,f)
(f,e)
(a,d)
(d,e)

Table 3.1: Set of events in which an event within the same case follows another directly

3.1.2 Causality

Causality, represented as x→y, is present when x comes before y (that is, x>y) and when y comes before x (that is, not y>x). This shows a directional dependency in which the occurrence of event x causes event y to occur.

```python
def causality(direct_succession):

    causality = []
    for pair in direct_succession:
        pair_reversed = (pair[1], pair[0])
        if pair_reversed not in direct_succession:
            pair_not_reversed = (pair[0], pair[1])
            causality.append(pair_not_reversed)

    return set(causality)
```

Listing 3.2: Python implementation of causality

The following causal relationships, in which one event directly causes another, were found after the Causality analysis was applied to the event log L:

→L
(d,f)
(a,c)
(b,d)
(a,b)
(d,e)
(c,d)
(a,d)

Table 3.2: Set of events in which an event directly causes another

3.1.3 Parallel

Parallel: Parallel, represented as x||y, happens when x comes before y (that is, x>y) and y comes before x (that is, y>x). This suggests that there is no set order in which events x and y must occur; they can happen simultaneously.

```
1
2  def parallel(direct_succession):
3
4      parallel = []
5      for pair in direct_succession:
6          pair_reversed = (pair[1], pair[0])
7          if pair_reversed in direct_succession:
8              pair_not_reversed = (pair[0], pair[1])
9              parallel.append(pair_not_reversed)
10
11     return set(parallel)
```

Listing 3.3: Python implementation of parallel

We discovered the following parallel relationships, where where two events can occur simultaneously, after applying the Parallel analysis to the event log L:

||L
(e,f)
(c,b)
(f,e)
(b,c)
(b,b)

Table 3.3: Set of events in which two events can occur simultaneously

3.1.4 Choice

Choice, represented as x#y, is present if neither x nor y come before x (that is, if x does not come before y or if y does not come before x). This shows that within the same process instance, events x and y are mutually exclusive and do not occur in order.

```
1
2  def choice(unique_events, causality, parallel):
3
4      choice = []
5      for event1 in unique_events:
6          for event2 in unique_events:
7              if (event1 != event2) and ((event1, event2) not in causality)
     and (
8                      (event2, event1) not in causality) and ((event1,
     event2) not in parallel):
9                  choice.append((event1, event2))
10
11     return set(choice)
```

Listing 3.4: Python implementation of choice

The choice analysis was applied to the event log L, and the resulting mutually exclusive relationships, where one event takes the place of another were as follows

#L
(c,f)
(c,e)
(f,c)
(f,b)
(f,a)
(b,f)
(b,e)
(a,f)
(a,e)
(e,c)
(e,b)
(e,a)

Table 3.4: Set of events which represents mutually exclusive relationships

3.2 The spm Metric

3.2.1 Methodology

In the field of process mining, the Heuristic Miner is a well-known algorithm that is known for its capacity to manage noise and quantify activity frequency, which helps in demonstrating dependencies among complex events. In our current application, information is extracted from event logs that document actual user activities as part of the process mining discovery task using the Heuristic Miner. The objective of this study is to improve the quality of the generated process models by enhancing the Heuristic Miner by adding the spm (Search Process Model) metric as an extra filter.[6]

Background of the Heuristic Miner: The Heuristic Miner finds process models that precisely represent the real behaviour recorded by an information system by examining event logs. It is appropriate for complex event logs because it excels at handling noisy data and highlighting recurring patterns. Three primary categories of process mining tasks are supported by the Heuristic Miner: enhancement, conformance checking, and discovery. Many examples have demonstrated this algorithm's capacity to manage various event log formats and quantify dependencies, making it a dependable option for process discovery.

Improvement with spm Metric: By tackling particular issues with the complexity and quality of search process models, the spm metric is presented as a means of enhancing the Heuristic Miner even more. A number of important considerations led to the decision to incorporate the spm metric into the Heuristic Miner:

- Relevance to Search Processes: The individual variability and complexity of search processes are extremely common. By analysing the complexity and level of shared behaviour in the models, the spm metric is intended to evaluate these processes.

Because of this, it is the perfect addition to the Heuristic Miner for handling search-related event logs.

- Reducing Complexity: Process logs frequently have a high degree of complexity, also known as the "spaghetti degree," which may hide important information. The spm metric simplifies the process models, making them easier to understand and evaluate by incorporating aggregation and abstraction principles.

- Quality Assurance: By taking into account the frequency of activity execution, the degree of node connections, and the overall diversity of the process, the spm metric offers an organised method for assessing the quality of search process models. This guarantees the high quality and representation of the models that are generated.

- Integration with Heuristic Miner: The Heuristic Miner can concentrate on high-quality nodes by integrating the spm metric as a filter, which lowers complexity and increases the precision of the process models that are found. This improvement makes it possible to evaluate the complexity and behavioural similarity of the model in an improved way.

3.2.2 Functionality

The spm metric is designed to improve the Heuristic Miner by offering a numerical assessment of search process model quality. It assesses every node in the process model according to how frequently it executes and how connected it is to the entire log. This method makes it possible to evaluate the complexity and behavioural similarity of the model more precisely.

Here is how the metric is defined:

$$\text{spm}(n) := 1 - \frac{\text{degree}(n) \times |A|}{\text{freq} \times |L|}$$

3.2.3 Overview of the Implementation

An broad explanation of the SPM metric and its theoretical implications for process mining and the Heuristic Miner were given in the preceding sections. But the real implementation of the SPM metric in the process mining tool, including how the filter is created in the user interface (UI) will be covered in-depth in the chapter that follows, "Implementation."

Throughout the "Implementation" chapter, we'll discuss:

- Integration of the spm Metric: A detailed account of the process used to integrate the spm metric into the Heuristic Miner algorithm.

- UI Enhancements: The adjustments made to the UI's layout to make room for the new spm filter, including improvements to its usability and placement as well as how it works with the other filters.

- Use Guidelines: Comprehensive guidelines explaining how users can utilise the tool's spm filter and apply it, along with any new user interface components.

We have made an important impact on the process mining tool's functionality and usability by including the SPM metric as a third filter. A thorough explanation of these modifications will be given in the "Implementation" chapter, guaranteeing that users can make the most of the SPM filter and improve their process analysis.

The UI has been modified to maintain the tool's user-friendliness while adding elegant filtering options. To assist users in comprehending and utilising the SPM filter, this includes layout adjustments, the addition of new interactive components, and improved visual feedback. The next chapter will include a detailed explanation on how to access the filter, navigate the user interface, and interpret the results.

To sum up, this section concentrated on the theoretical background and justifications; the upcoming chapter will explore the practical aspects of applying the spm metric, highlighting its advantages and offering instructions for efficient use.

3.3 Debugging and Bug Fixing

3.3.1 Methodology

Two main areas of focus for debugging and bug-fixing were handling very large and very small datasets, as well as resolving UI issues arising from the addition of the spm filter. At first, the program worked well with medium-sized datasets, but it had serious issues with very large/small datasets. Furthermore, there were difficulties incorporating the new spm filter into the current interface, especially on smaller screens where the filter text would be invisible.

3.3.2 Handling Very Large and Very Small Datasets

Before any fixes were applied, the program had trouble importing both very small and very large datasets. These problems resulted from incorrect encoding and delimiter detection. A mix of error handling and debugging techniques were used to address these issues:

Error Handling Enhancements:

- **Delimiter Selector Implementation:** At first, the application used a sniffer to automatically identify delimiters. Nevertheless, this approach was unstable and frequently wrongly identified delimiters, resulting in program crashes. Users can now choose the delimiter manually before importing the CSV due to the implementation of a manual delimiter selector. This method increased reliability in some cases, especially when it came to CSV files that used the ';' delimiter.
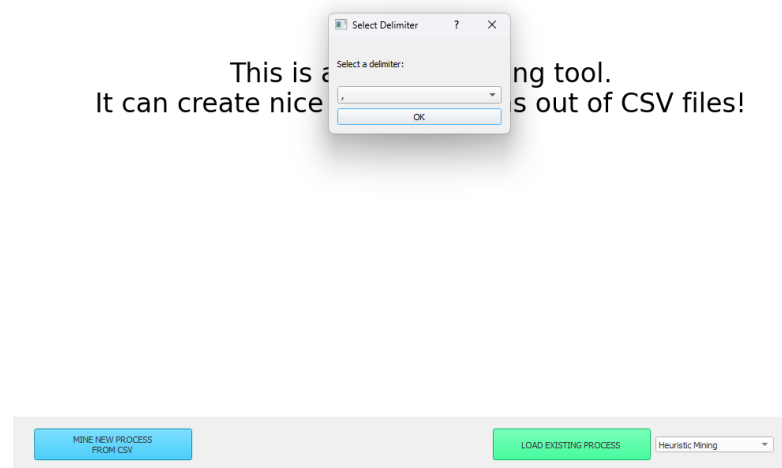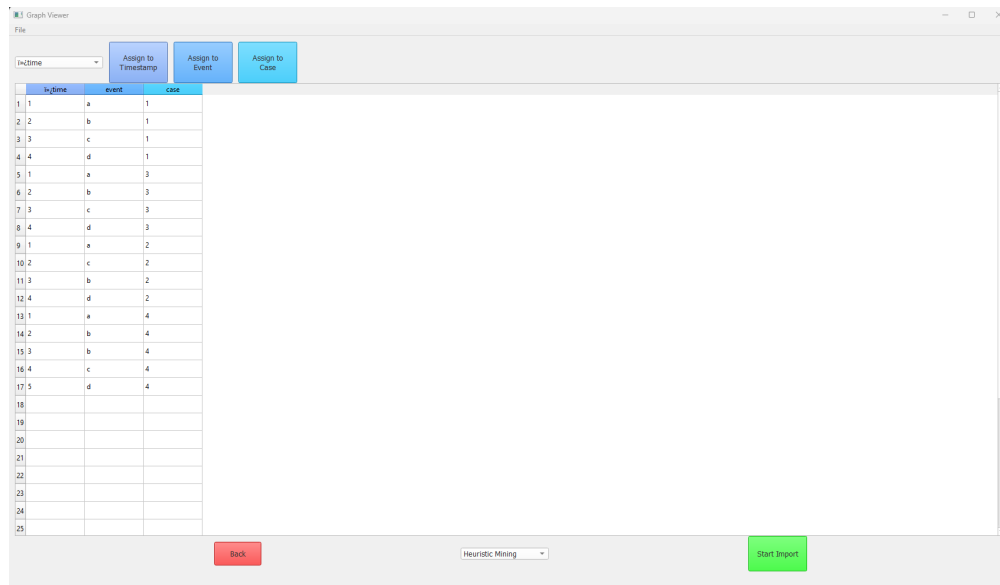

Figure 3.1: Delimiter selector in the UI.

- **UTF-8 Encoding Fixes:** One common problem, brought on by the Byte Order Mark (BOM), was the existence of odd symbols at the start of some CSV files. Errors occurred when the program first read these files using cp1252 encoding rather than UTF-8. Column detection errors were fixed and these symbols were removed by ensuring proper UTF-8 encoding.

Figure 3.2: BOM Issue in CSV File (first row) in time.

Final Delimiter Handling Solution:

Following recognition of the weaknesses of the manual delimiter selection process, a hybrid strategy was implemented:

- **Automatic Delimiter Detection with Sniffer:** Higher accuracy was ensured by returning the sniffer for automatic detection. The sniffer would check again and change the selection if it found a better delimiter, even if users made a mistake in their delimiter selection.

- **Error Messaging:** Users are now prompted to try again with a different dataset or delimiter by the program, which clearly displays an error message if both user selection and automatic detection fail.
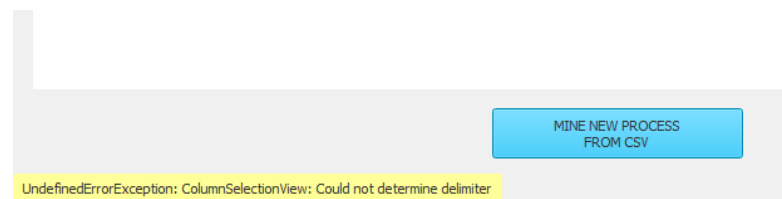
Figure 3.3: Message for Incorrect Delimiter.

Handling Very Small Datasets:

- **Error Handling for Small Datasets:** Using the Alpha Miner, the application was able to import and mine extremely small datasets. Nevertheless, errors would result from attempting to use these datasets with the Heuristic and Fuzzy Miners. Error handling was added to show the user a message when a very small dataset is not compatible with these mining techniques, preventing crashes. By doing this, the program won't crash and the user will be informed of the problem.
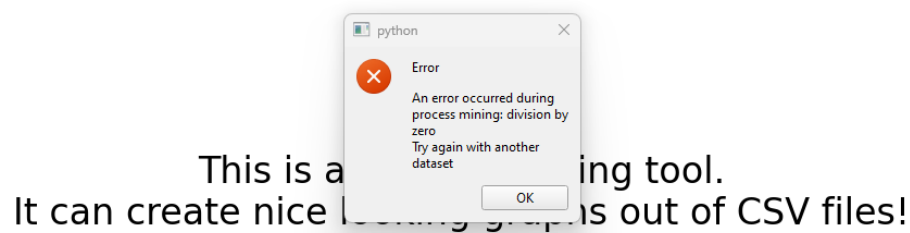

Figure 3.4: Error Message for Small Dataset Compatibility

3.3.3 UI Changes

The user interface faced additional difficulties after the spm filter was added. More specifically, because of limitations in space, the filter text would disappear on smaller screens. In order to address this, various UI enhancements were made:

Horizontal Filter Arrangement:

- **Filter Orientation Change:** The arrangement of the filters was changed from vertical to horizontal. This modification fixed the disappearing text problem on smaller screens in addition to enhancing the UI's visual appeal.

Enhanced Filter Controls:

- **Direct Number Input:** Users can now directly enter numbers for each filter in addition to using the sliders. This dual-input approach improves the accuracy and flexibility of the user.
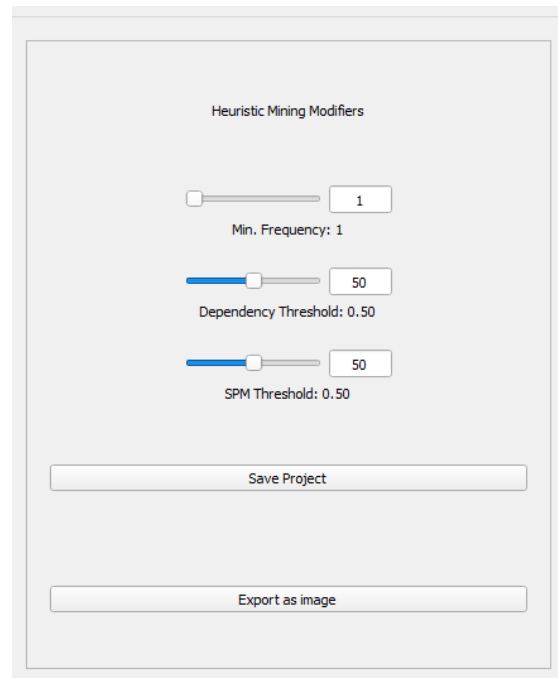

Figure 3.5: New Horizontal Filter Layout and Enhanced Filter Controls with Number Input

The process mining tool's usability and reliability have been significantly enhanced by these upgrades, guaranteeing that it can handle a larger variety of datasets and offer a more user-friendly interface.

3.3.4 Summary

This section described the UI changes required to incorporate the new SPM filter, as well as the debugging and bug-fixing process used to enhance the handling of large and small datasets. The process mining tool's robustness and usability were significantly increased by tackling these problems with a combination of improved error handling and careful UI design.

4 Implementation

4.1 Explanation of the engineered tool

4.1.1 Alpha Miner Algorithm

To improve the process mining tool's ability to identify process models from event logs, the Alpha Miner algorithm was added as a new feature. Additional information regarding the Alpha Miner algorithm and its importance was covered in Chapter 3, Major Idea. This algorithm, which is well-known in the process mining community, examines the sequential relationships between events in the log to determine the underlying process model.

Challenges and Implementation

1. Updating the UI

   - **Drop-down Modification:** Updating the user interface where the algorithms for process mining are chosen was the first step in the implementation process. This required adding the Alpha Miner option to the already-existing dropdown menu.

   - **Process:** To achieve this, the UI code was updated to add "Alpha Miner" as a new selectable item in the dropdown list. This ensures that users can easily choose the Alpha Miner algorithm from the list of available algorithms.
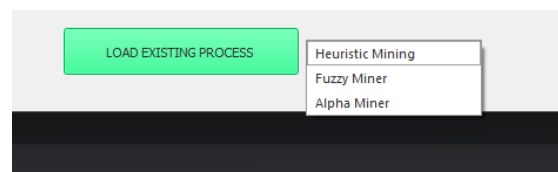

Figure 4.1: Updated Dropdown Menu with Alpha Miner Option

We will then go into the fundamentals of the Alpha Miner algorithm, explaining how it processes event logs in order to extract the relationships required in order to build a process model. We will focus on going further into the execution of the eight steps that make up the Alpha Miner algorithm.

2. **8 Steps for implementing the Alpha Algorithm**[10][5][7]

The most important thing to do now that I had updated the user interface was to put the eight known steps of the Alpha Miner algorithm into practice. The simple steps and the more complicated steps are the two categories into which these steps can be separated according to their level of complexity. In particular, steps 1-3 and 6–8 are simple, but steps 4 and 5 are more complicated. I prefer to think of these eight steps as being as follows:

a) **Define All Events:** Identify all unique events in the log.

```python
def get_unique_events(self):
    unique_events = []

    for case in self.cases:
        for event in case:
            unique_events.append(event)

    return set(unique_events)
```
<div align="center">Listing 4.1: Python implementation of step 1</div>

b) **Define All Possible Start Events:** Identify the starting events of each trace.

```python
def get_start_events(self):
    start_events = []

    for case in self.cases:
        start_events.append(case[0])

    return set(start_events)
```
<div align="center">Listing 4.2: Python implementation of step 2</div>

c) **Define All Possible End Events:** Identify the end events of each trace.

```python
def get_end_events(self):
    end_events = []

    for case in self.cases:
        end_events.append(case[len(case) - 1])

    return set(end_events))
```
<div align="center">Listing 4.3: Python implementation of step 3</div>

d) **Calculate Possible Sets A (Input) and B (Output):** Using the footprint matrix, identify sets where elements in A are in causality with elements in B, elements in A are in a choice set with each other, and the same for elements in B.

```python
def generate_set_xl(self, unique_events, choice, causality):
    xl_set = []

    subsets = itertools.chain.from_iterable(
        itertools.combinations(unique_events, r) for r in range
    (1, len(unique_events) + 1))
    subsets_in_choice = [_set for _set in subsets if self.
    __is_set_in_choice(_set, choice)]
    for a, b in itertools.product(subsets_in_choice,
    subsets_in_choice):
        if self.__is_set_in_causality((a, b), causality):
            xl_set.append((a, b))

    return set(xl_set)
```

Listing 4.4: Python implementation of step 4

e) **Drop Non-Maximum Sets:** Eliminate unnecessary sets by ensuring that removing an event from A or B will still result in elements that do not follow each other (remain in a choice set).

```python
    def generate_set_yl(self, xl_set, parallel):

        yl_set = xl_set
        s_all = itertools.combinations(yl_set, 2)

        for pair in s_all:
            if self.__is_subset(pair[0], pair[1]):
                yl_set.discard(pair[0])
            elif self.__is_subset(pair[1], pair[0]):
                yl_set.discard(pair[1])

        self_loop = set()
        for pair in parallel:
            if pair == pair[::-1]:
                self_loop.add(pair[0])

        to_be_deleted = set()
        for pair in yl_set:
            if self.__contains(pair, self_loop):
                to_be_deleted.add(pair)
        for pair in to_be_deleted:
            yl_set.discard(pair)
        return yl_set
```

Listing 4.5: Python implementation of step 5

f) **Create Places for All Derived Sets (1-4):** Include the final start and end states, and create related input and output transitions.

g) **Draw the Connections:** Using the information from the previous steps, establish links between places and transitions.

h) **Return the Finished Petri Net:** Output the created Petri net model.

3. **Common patterns of operation and the traces they leave in the event log**

Here, I will talk about the common patterns of behaviour and the traces that the Alpha Miner algorithm finds in the event log. It was difficult to define all the various patterns in Python and to redirect when making the diagram, even though the final step, "Return the Finished Petri Net," seems simple. There are five patterns altogether:[10]
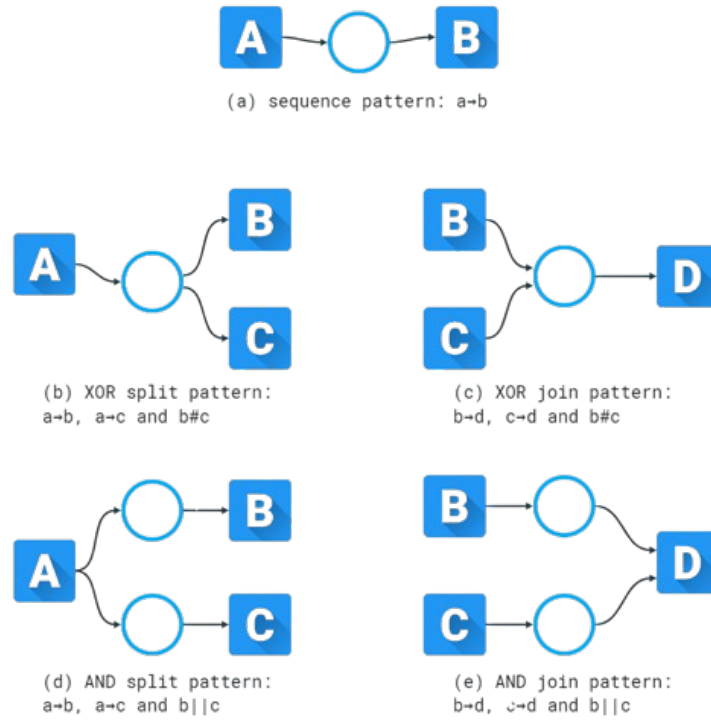

(a) sequence pattern: a→b

(b) XOR split pattern: a→b, a→c and b#c

(c) XOR join pattern: b→d, c→d and b#c

(d) AND split pattern: a→b, a→c and b||c

(e) AND join pattern: b→d, c→d and b||c

Figure 4.2: Common patterns of operation and the traces they leave in the event log

4.1.2 The spm Metric

Using the spm Filter to enhance process mining with the Heuristic Miner

The goal of implementing the spm (Search Process Model) Metric filter was to improve the Heuristic Miner's capacity to evaluate and refine search process model quality. The user interface (UI) and the underlying algorithmic logic both needed to be updated as a result.

1. **UI Changes:**

- **Adding the spm Filter** The spm filter was added to the Heuristic Miner view alongside the other two filters by updating the user interface. Because of this, the arrangement had to be changed to make room for the extra filter.

- **Orientation Adjustment** The filters were initially shown horizontally, which led to problems with text overflow. The filters' orientation was changed to vertical, which made all filter labels and controls visible and easy to use.

- **Number input field:** Each filter control now has a number input field, improving usability by removing the need for the slider and enabling users to enter values directly.

2. **Filter Implementation:**

   Two helper methods were needed for the filter implementation. A node's degree was determined using the first method, and the spm was determined using the degree in the second method.

   - **Degree Calculation:**

```python
def calculate_degree(node, events, succession_matrix):
    index = events.index(node)
    in_degree = sum(succession_matrix[:, index])
    out_degree = sum(succession_matrix[index, :])
    return in_degree + out_degree
```

   Listing 4.6: Python implementation of calculating the degree

   - **spm Calculation:**

```python
def calculate_spm(node, events, node_frequency, A, L,
    succession_matrix):
    degree = calculate_degree(node, events, succession_matrix)
    frequency = node_frequency
    spm = 1 - ((degree * A) / (frequency * L))
    return spm
```

   Listing 4.7: Python implementation of calculating the spm

3. **Handling Edge Cases:**

   A significant problem was that the spm filter eliminated events, which caused the start and end nodes' proper connections to be broken. In order to make sure that the start and end nodes connected properly even after events were filtered out, this required debugging and modifying the diagram-drawing code.

4.2 Explanation of Usage

4.2.1 Using Alpha Miner as a process discovery

It's very easy to use the Alpha Miner for the event of your choice. Once the application has opened, click the 'Load CSV' button and choose the event log file from your device to

load your CSV file. Select the Alpha Miner algorithm from the dropdown menu as soon as the file loads, as shown in the image 4.1. All you have to do to begin mining is click the 'Start Import' button after choosing the Alpha Miner. The Petri Net diagram that results from processing the event log will be shown by the application. The application also offers the ability to download the diagram as an image, which makes it simple for you to store and distribute the outcomes of your process mining analysis.

4.2.2 Using the spm Filter

The spm filter is easy to use and operates similarly to the other filters. Here's a detailed how-to:

1. **Load the CSV File:** Launch the program and import the required CSV file.

2. **Select the Heuristic Miner:** From the dropdown menu, select the Heuristic Miner (see Figure 4.1).

3. **Adjust Filters:** Three filters, including the recently added spm filter, are visible in the filter view (see Figure 3.5). For ease of use, each filter has a number input field and a slider.

4. **Start the Import:** To get the mining process started, click the "Start Import" button.

5. **Filter Adjustments:** You can change the filter values after the first import as necessary. Real-time updates to the graph will show these modifications.

6. **Download the Diagram:** The user can download the process model diagram as an image once they are happy with the outcome.

These procedures demonstrate how users can efficiently refine their process models and extract more insightful information from their event logs by utilising the enhanced Heuristic Miner with the SPM filter.

5 Evaluation and discussion

Understanding the previous work, the underlying technologies, and the implementations of the algorithms and user interface (UI) was essential at the outset of this project. The development of AI tools has increased the efficiency of existing code analysis and evaluation. I was able to understand complicated code sections with ChatGPT, though it took several attempts to get it right and fully understood. This initial stage was important in locating anomalies and deciding on the scope of my study. The challenge wasn't limited to becoming familiar with the current codebase. Understanding the algorithms that I planned to use was equally important. This required manually solving algorithmic problems with paper and pen before converting them to a computer language.

Through this project, important improvements were made, such as the application of the Alpha Miner algorithm, which serves as the basis for process mining; the addition of a new filter to the Heuristic Miner, increasing its functionality; and the improvement of error handling, which makes the tool more reliable and user-friendly. These improvements improved the process mining tool's stability and usability in addition to increasing its functionality.

Test event logs and real-world data were used to evaluate the features that were put into place. This two-pronged strategy made sure the improvements and algorithms were reliable and useful in real-world situations. The test logs allowed for controlled testing of particular functionalities and edge cases, while the real-world data gave an in-depth understanding of the tool's performance in real-world scenarios. The evaluation produced positive results: the enhanced Heuristic Miner, with the newly implemented filter, demonstrated improved accuracy in process model discovery; the improved error handling effectively managed exceptions and gave users clear feedback, preventing crashes and improving the overall user experience; and the Alpha Miner, which successfully discovered process models from a range of event logs, confirmed its implementation.

One of the surprising discoveries was the realisation that even small adjustments to the user interface could result in serious problems. This experience made it clear that a comprehensive approach to tool development is required, taking into account both the functional and visual aspects of the tool. To sum up, this project succeeded in achieving its main objectives, adding new features and algorithms to the process mining tool. The difficulties faced and the lessons discovered offer insightful information for upcoming studies, ensuring that process mining tools will continue to advance and innovate.

6 Conclusion and future work

In conclusion, this thesis has successfully improved a process mining tool by adding a new SPM filter to the Heuristic Miner and implementing the Alpha Miner algorithm. Through the use of AI tools, the journey required a thorough understanding of the existing code and an in-depth analysis of anomalies to pinpoint areas in need of improvement. By adding these algorithms, the tool's functionality was increased, enabling the extraction of more precise and informative process models from event logs. Despite difficulties, mainly with regard to UI design and error management, the project produced a more reliable and user-friendly tool.

But there are drawbacks to the strategy. As the original process discovery algorithm, the Alpha Miner has built-in limitations. With formal guarantees, it can find concurrent process models based on imperfect event data. This limitation limits its use in more complex real-world situations where processes tend to be disorganised and have a large number of exceptions.

There are various interesting possibilities for further research in the future. First off, there is a lot of room for improvement in the user interface. The user experience can be further enhanced by adding more interactive and customisable features to the UI.

Furthermore, investigating and putting into practice more sophisticated algorithms is an essential next step given the limitations of the Alpha Miner. Many of the problems of the Alpha Miner are addressed, and an effective replacement is provided by the Alpha++ described in [2] and Alpha+++ algorithm, which was described in the work "Revisiting the Alpha Algorithm To Enable Real-Life Process Discovery Applications" by Aaron Küsters and Wil M.P. van der Aalst [5]. The tool's capacity to manage real-world process discovery applications efficiently can be improved by integrating Alpha+++ or other advanced process mining algorithms.

In conclusion, even though a lot has been accomplished, there is still a lot of room for future research to expand on these achievements. Through addressing the weaknesses of the present methodology and investigating new methods and algorithms, the process mining tool can further develop, providing even more potent and intuitive solutions for process analysis and discovery.

List of Tables

List of Figures

Listings

Bibliography

[1] Van der Aalst and WMP Process Mining. Discovery, conformance and enhancement of business processes. *Media; Springer: Berlin/Heidelberg, Germany*, 136, 2011.

[2] Yutika Amelia Effendi and Riyanarto Sarno. Conformance checking evaluation of process discovery using modified alpha++ miner algorithm. In *2018 International Seminar on Application for Technology of Information and Communication*, pages 435–440. IEEE, 2018.

[3] Christian W Günther and Wil MP Van Der Aalst. Fuzzy mining–adaptive process simplification based on multi-perspective metrics. In *International conference on business process management*, pages 328–343. Springer, 2007.

[4] Angelina Prima Kurniati, Guntur Kusuma, and Gede Wisudiawan. Implementing heuristic miner for different types of event logs. *vol*, 11:5523–5529, 2016.

[5] Aaron Küsters and Wil MP van der Aalst. Revisiting the alpha algorithm to enable real-life process discovery applications. In *ATAED/PN4TT@ Petri Nets*, 2023.

[6] Marian Lux, Stefanie Rinderle-Ma, and Andrei Preda. Assessing the quality of search process models. In *Business Process Management: 16th International Conference, BPM 2018, Sydney, NSW, Australia, September 9–14, 2018, Proceedings 16*, pages 445–461. Springer, 2018.

[7] Phyllalintang Nafasa, Indra Waspada, Nurdin Bahtiar, and Adi Wibowo. Implementation of alpha miner algorithm in process mining application development for online learning activities based on moodle event log data. In *2019 3rd International Conference on Informatics and Computational Sciences (ICICoS)*, pages 1–6. IEEE, 2019.

[8] Wil Van Der Aalst. Process mining: Overview and opportunities. *ACM Transactions on Management Information Systems (TMIS)*, 3(2):1–17, 2012.

[9] Wil MP Van der Aalst and Wil MP van der Aalst. Process discovery: An introduction. *Process mining: Discovery, conformance and enhancement of business processes*, pages 125–156, 2011.

[10] Wil MP Van Der Aalst and Boudewijn F Van Dongen. Discovering petri nets from event logs. In *Transactions on Petri nets and other models of concurrency vii*, pages 372–422. Springer, 2013.