Bachelor thesis

A Python Desktop App for Business Process Mining and Visualization

Author

Anton Chen

angestrebter akademischer Grad

Bachelor of Science (BSc)

Wien, 2023

Studienkennzahl lt. Studienblatt:	A 033 521
Fachrichtung:	Informatik - Allgemeine Informatik
Betreuer:	Dipl.-Ing. Marian LUX

Abstract

This Bachelor thesis presents the development of an open-source desktop application for generating interactive business process graphs from CSV files. Inspired by the Fluxicon DISCO app, the project focuses on creating a user-friendly tool capable of parsing CSV data and applying algorithms to visualize process flow and dependencies. The project is written entirely in Python, ensuring cross platform compatibility and easy customization.

The application emphasizes ease of use, providing an intuitive interface for importing CSV files and generating informative graphs. Various algorithms, including process discovery and process mining techniques, are incorporated to transform the data into visually appealing representations. Users can interact with the graphs, explore process variations, and identify inefficiencies.

By being open-source, the project aims to contribute to the business process analysis community, empowering users to customize and enhance the tool based on their specific needs.

This open-source desktop application democratizes access to powerful visualization tools and promotes collaboration among practitioners, researchers, and developers. Future work may involve integrating additional algorithms and expanding upon the apps features.

Additionally, this thesis will also discuss the use of advanced AI coding assistance, namely OpenAI's ChatGPT, in a project like this one.

Acknowledgements
Dipl.-Ing. Marian Lux - Project Overseer and author of the DDCAL clustering library

Github-Repository
https://github.com/ShiningVision/Process-Mining-Visualization

Contents

1 Motivation **4**

2 Algorithms **5**
2.1 Heuristic Mining Algorithm 5
2.2 DDCAL 6

3 Design and Implementation **7**
3.1 Desktop Application library choices 7
3.2 Project Implementation 8
3.2.1 The File Structure 8
3.2.2 A typical session example 9
3.2.3 Column Selection View 11
3.2.4 Heuristic Graph Viewer 12
3.2.5 MVC Pattern 13
3.2.6 Extensibility for new algorithms 14
3.2.7 Extensibility for new features 14
3.3 Usage Instruction 15
3.4 Unit Testing 15

4 Evaluation and Discussion **16**
4.1 Implementation Afterthoughts 16
4.2 The role of ChatGPT 16
4.2.1 Writing a Python Desk-Application Quickstart 17
4.2.2 Autocomplete Code 18
4.2.3 Introduction to new libraries 21
4.2.4 Using Python without learning Python 22
4.2.5 Letting ChatGPT write your test cases 24
4.2.6 Limits of ChatGPT 26

5 Conclusion and Future Work **26**

1 Motivation

In today's dynamic and intricate business landscape, understanding the inner workings of organizations and pinpointing workflow inefficiencies can be a challenging task. Businesses, particularly large ones, often operate complex processes involving numerous activities and stakeholders. Identifying areas for improvement and optimizing these processes requires a comprehensive analysis of the underlying data. Process mining[3] has emerged as a valuable technique for extracting insights from log files and uncovering hidden patterns and bottlenecks in business processes.

Process mining allows organizations to analyze event logs and reconstruct the sequence of activities performed during process execution. By applying various algorithms and techniques to these event logs, valuable insights can be derived, providing organizations with a clearer understanding of their processes. However, all insights are of limited use, if they can't be effectively communicated to the stakeholders.

This is where visualization plays a crucial role. It bridges the gap between raw data and actionable insights. By presenting process mining results in visually appealing and intuitive graphs, stakeholders can easily comprehend and interpret complex process information. Visualizations enable users, including outsiders, to quickly identify workflow inefficiencies, bottlenecks, and areas for improvement. They facilitate a shared understanding among stakeholders and enable effective communication and collaboration.

While there are existing process mining tools available on the market, such as Fluxicon DISCO [1] and ProM Uitopia [2], they often come with limitations. For instance, DISCO is a commercial tool that requires a financial investment, which may not be feasible for all organizations, particularly smaller ones or research projects with limited budgets. Additionally, it does not provide the flexibility to choose the desired algorithms or easily extend their functionalities to suit specific organizational needs. The ProM tool on the other hand is open source and extendable, but is very technical and ill suited for non-professional users.

Therefore, the aim of this research project is to address these limitations by developing an open-source, user-friendly, and extensible application framework for process mining visualization in Python. By creating an open source tool, this project seeks to empower users to analyze log files, generate visually appealing interactive graphs, and gain valuable insights into their business processes. Furthermore, by providing flexibility in algorithm selection and extensibility, the tool will enable users to adapt and customize the visualization approach to meet their specific requirements. And by using DISCO as a reference for user interface and features, this project will also create an intuitive tool for non-professional users.

[1] https://fluxicon.com/disco/, accessed 2023-07-06
[2] https://promtools.org/, accessed 2023-07-06

2 Algorithms

2.1 Heuristic Mining Algorithm

In this project, the Heuristic Mining Algorithm was chosen as the first implemented algorithm for business process mining due to its simplicity, relevance and effectiveness. Unlike the Alpha algorithm, the heuristic algorithm takes into account the frequencies of events and sequences when constructing the process model. Its goal is to focus on the most common paths in the process while disregarding infrequent ones. [3]

In a nutshell, the heuristic algorithm takes analyzes all the cases and generates a succession matrix of all the activities. Then it generates a dependency matrix based on the succession matrix measuring the one-sidedness of the relationship between 2 activities. It determines the significance of the succession by comparing the relative frequencies of one activity being followed by another compared to the reverse case. And by setting a dependency threshold, we can ensure that events that likely happened in parallel are not mistakenly considered as a sequence of activities.

Consider the sequence of events in 3 very simple cases:

1. a b c d
2. a b c d
3. a c b d

The succession matrix, describing the succession order of the cases would then look like this:

	a	b	c	d
a	0	2	1	0
b	0	0	2	1
c	0	1	0	2
d	0	0	0	0

Table 1: Succession Matrix

The dependency matrix that results from the succession matrix is then calculated using the following formula:

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

Which leads to the following dependency matrix:

Now if all the successions in the dependency matrix that have a value higher than a threshold of 0 are drawn, you get the graph in Figure1.

	a	b	c	d
a	0	2/3	1/2	0
b	-2/3	0	1/4	1/2
c	-1/2	-1/4	0	1/3
d	0	-1/2	-1/3	0

Table 2: Dependency Matrix

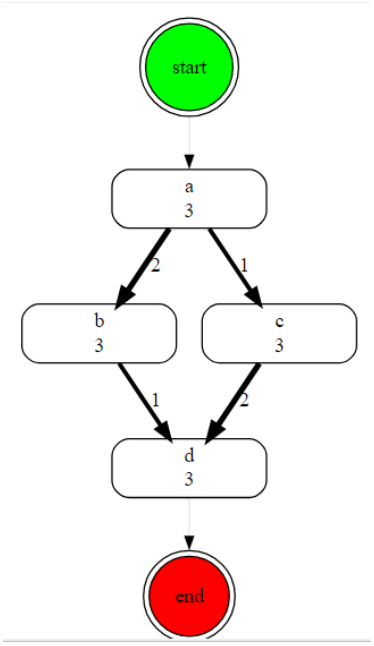

Figure 1: Example Graph

2.2 DDCAL

In order to visualize the data in a pleasant way, we could change the size of the nodes depending on its appearance frequency in the log files. However this method of visualization results in outliers being ridiculously large or small and nodes with similar, but not equal appearance frequency to appear almost equal, which puts a strain on the human brain having to identify the subtle difference.

This is where Clustering algorithms come into play. Clustering algorithms deal with data structure partition in an unknown area. Instances in the same cluster must be as similar as possible. And instances in different clusters must be as different as possible. [1] By clustering the appearance frequency of nodes and edges, those nodes and edges can be shown in a limited number of sizes, which is more pleasant for the human brain to process.

The clustering algorithm for this purpose must meet the following require-

ments:

1. Handle one dimensional data set (appearance frequency)
2. Work on real-world data sets, e.g., data sets containing gaps between data points
3. Reproduce the same clusters after each execution on the same data set
4. Define a targeted number of clusters (Only a given number of cluster sizes)
5. Avoid overlapping cluster ranges
6. Handle non-unique data points in data sets (multiple nodes with same appearance frequency)
7. Result in an even distribution of data points into clusters (to better handle outliers)

DDCAL (1D Distribution Cluster Algorithm) is a clustering algorithm based on iterative feature scaling specifically designed with process mining graphs in mind.[2] And it meets all the above requirements. The python library by the same author used in this project is a pypi project[3]

3 Design and Implementation

3.1 Desktop Application library choices

As in any big programming project, external libraries were also used in this project. This section tries to explain why certain libraries were chosen.

Firstly, PyQt5 was selected as the library for the Desktop Application UI. This decision was based on the fact that PyQt5 has a well-established presence and a large community of users, making bug fixing and support readily available. In contrast, PyQt6 was released in 2021, and given the limited knowledge available about it within the current context, it was deemed more advantageous to rely on the stability and extensive resources of PyQt5. Given the projects intention to fully harness the power of AI to speed up development, the PyQt6 release date was a major discouraging factor. Since as of date, ChatGPT has very limited knowledge about libraries that released after 2021 and likely incomplete knowledge about libraries from 2021.

Next, graphviz was chosen as the primary graph drawing library. While networkx was still required for unittesting. The reasoning for this choice is explained in Section 3.2.4.

One of the requirements of this project was a 'Save project' feature, which allows uncomplicated saving and loading of process mining projects. Initially it was decided to save the project as a .txt file. The .txt file would contain the preprocessed cases line by line. It allowed any chosen algorithm view to load

[3]https://pypi.org/project/ddcal/, accessed 2023-07-07

the file and show a corresponding graph, while skipping the column selection process. The downside of this approach was that parameters such as the min frequency slider value are not saved and the user has to set them again. However, this approach was later abandoned in favour of pickle. A python library that allows saving python class instances as a bytestream and later load it again. This new approach allowed the saving of the parameters as well, in an uncomplicated manner. And since pickle instances are a way to temporarily save a project and not intended to be shared across devices, the security risk that pickle poses does not play a role in this case.

Finally, there is the DDCAL library. The use of this library is explained in Section 2.2

3.2 Project Implementation

This section explains in detail how the project is organized and how the program works.

3.2.1 The File Structure

Figure 2 shows the overall structure of the project. All the view control methods are gathered in the Qt5 MainWindow class. It holds the mainWidget, which is a QStackedWidget that holds all the views and can change its active view with the various switch-to methods. The mine-new-process method switches to the columnSelectionView and the mine-existing-process method prompts the user to select a pickle file in the designated folder before switching to the chosen algorithmView. Additionally the MainWindow also has a method to show a quick pop up notification whenever needed.

The algorithms and business logic are outsourced to the respective views that use them. This way new views with new features can be implemented without knowledge of the other views. Ensuring separation of concerns.

The custom-wigdet.py file contains many classes that inherit from QWidget and are used across views to enable a uniform look and less duplicate code.

The Graph Viewers all inherit from AlgorithmViewInterface, as there are methods every Graph Viewer must support. Implementation instructions are written in the AlgorthmViewInterface class as comments.

The HTMLWidget uses QWebengine to show a HTML view running on a HTMLServer instance inside the Desktop Application and reacts to user interaction with the HTML view by sending/receiving signals from and to the HTMLServer. Since the callbacks of the HTMLServer must always match the react methods of the HTMLWidget, both classes are written in a single file to enable easy replacement if needed. For example the d3-html-widget.py.

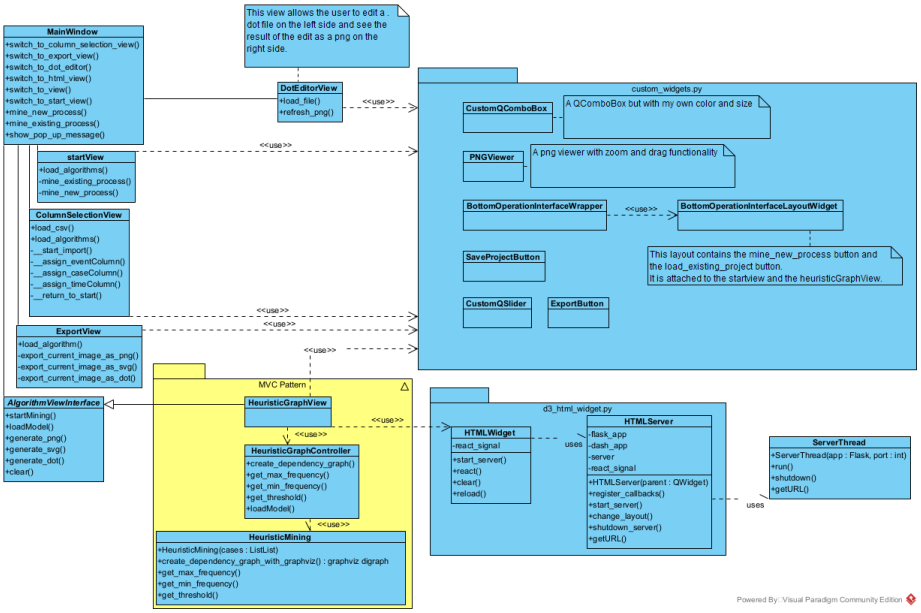

Figure 2: class diagram

3.2.2 A typical session example

Figure 4 shows a sequence diagram representing a typical session, mining a new process.

Upon starting the application the user is greeted by the StartView. If the user clicks on the MineNewProcess button, the StartView will tell the MainWindow to switch to the ColumnSelectionView.

The MainWindow will then proceed and prompt the user with a FileDialog to select a CSV file to mine. When a CSV file is selected, MainWindow will change the canvas to the ColumnSelectionView (indicated with a call to its own function), where the user can manually select 3 columns to assign to timestamp, case and event. Lastly the user can select the algorithm he wishes to use and click the StartImport button. [4]

This will prompt the MainWindow to call the startMining method of the selected algorithmView before switching to said view.

In the selected algorithmView the user can now play around with the sliders and interact with the graph. If implemented, like in the heuristic view (cf. Figure 6), the user can save the project as a pickle file under a custom name, so it can be loaded later. Every algorithm view has its own numbered default save folder. Finally the user can export the graph by either clicking on the Export Button or select 'Export' in the File menu. MainWindow will then switch to the Export View, where the user can see the image on the left side, choose between

[4] More detailed description of the Column Selection View in Section 3.2.3

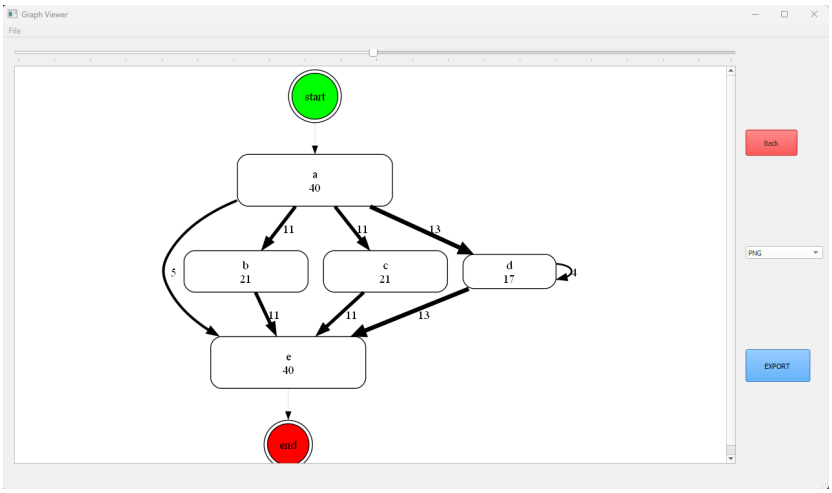

Figure 3: Export View

PNG,SVG and DOT file format on the right side and finally export the image to a selected path/folder. (cf. Figure 3)

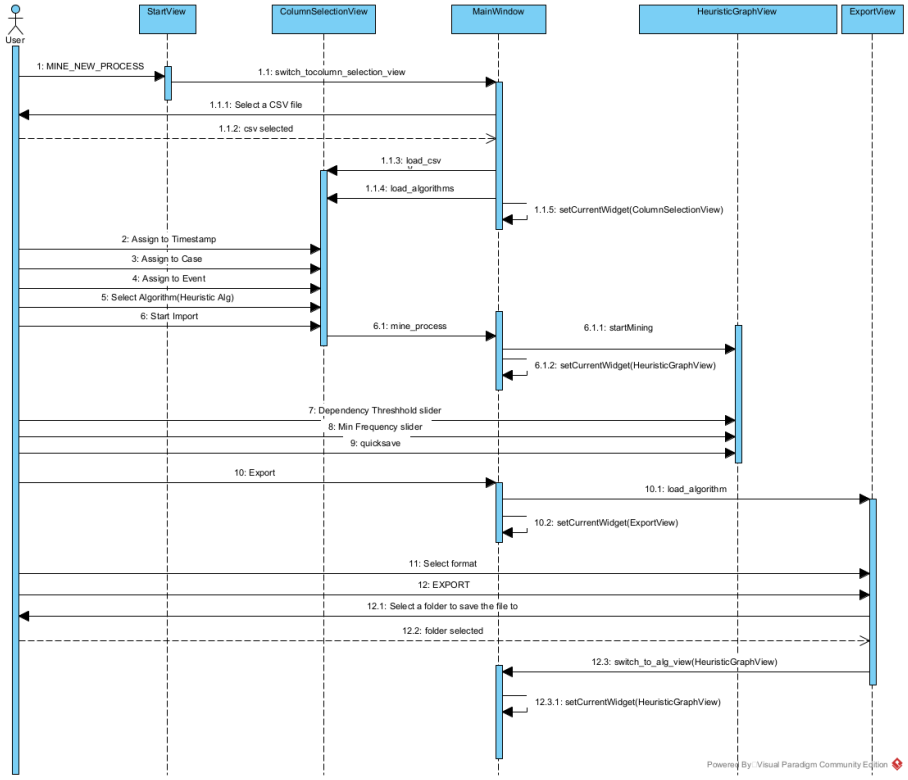

Figure 4: sequence diagram

3.2.3 Column Selection View

The column selection view (cf. Figure 5) serves the purpose to assign 3 necessary columns to the 3 labels 'timestamp', 'case' and 'event'. All other columns are, for now at least, not relevant to the graph. The timestamp column must be in a datetime format. The case column groups individual rows with the same case together and the event column is the name of the event that happens.

The user can assign the columns by clicking on the header of the column and then clicking on one of the 'Assign to' buttons or select a column in the dropdown on the upper left side and click on the 'Assign to' button. The colors of the columns will change according to the assignment. By default the first 3 columns are assigned to the 3 labels.

When the startImport button is triggered, the program will show a pop up message showing the selected columns and respective assignments for the user to verify.

If the user confirms the correctness of the assignments, the ColumnSelectionView will create a list of all cases. Each case holds a list of all the events that are assigned to that case and the events are sorted from earliest to latest

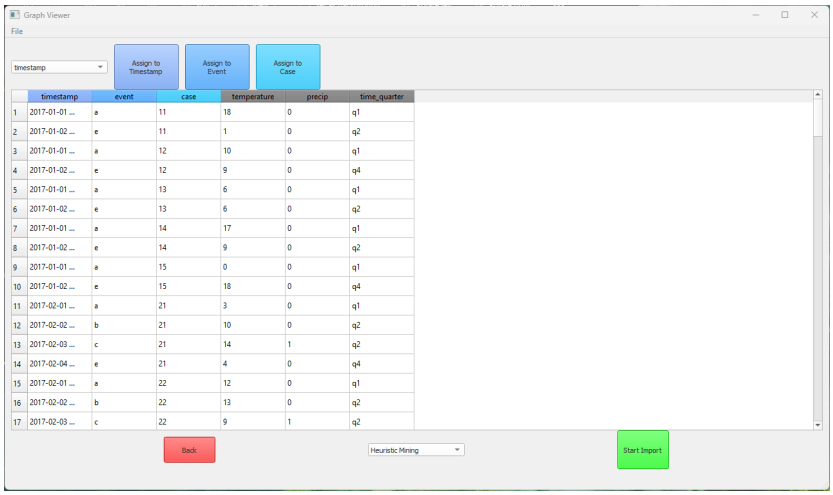

Figure 5: Column Selection View

timestamp. This list of lists is then forwarded to MainWindow which uses it to call the startMining() method of the selected algorithmView.

3.2.4 Heuristic Graph Viewer

There are many ways to visualize business process graphs. Some common graph types used in heuristic-process-mining are causal nets and petri nets. This project however, uses simple directional graphs, as that is the graph type used in the Fluxicon DISCO application this project is inspired by. And to reduce the complexity of the graph, so anyone can interpret what the graph is showing without prior knowledge about causal nets or petri nets.

The graph is furthermore required to be interactive and responsive. It has to return data to the main application, enabling further processing features to be implemented in future extensions.

While there are many libraries out there enabling interactivity. To achieve graph responsivenes in Python, the only method found was the HTML approach, which will influence the library choices discussed later in this section.

During the research for network graph libraries, two major libraries for Python were identified: graphviz and networkx. Networkx stands out for its user-friendly interface, providing a networkx object with convenient methods for retrieving nodes and edges. It also offers an internal draw method and seamless compatibility with popular graph drawing libraries such as Plotly, Bokuh, Pyvis, among others. However, one limitation of networkx is its suitability. It is only suited for relationship networks, such as social media networks and not for streamlined process graphs. Reason being the absence of curved edges, which is a significant drawback.

On the other hand, graphviz allows the creation of visually stunning graphs

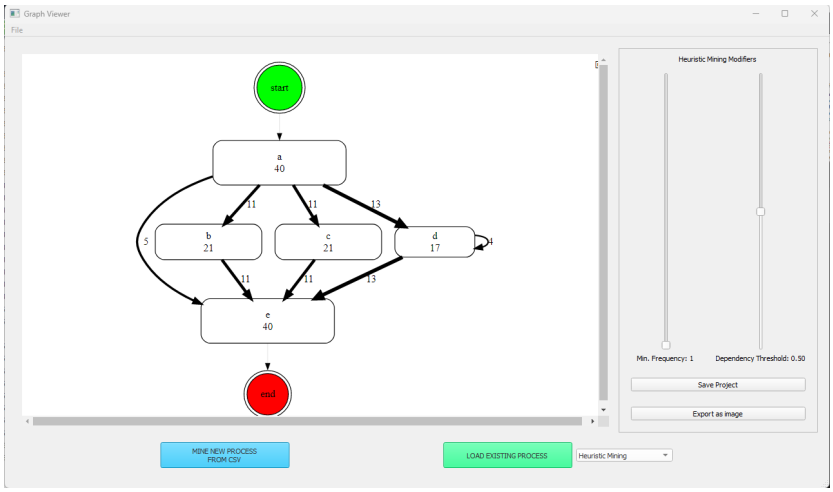

Figure 6: Heuristic Graph View

with curvy edges. Nevertheless, it lacks support for interactivity in any form. A graphviz object also does not provide methods for querying graph nodes and testing generated graphs, nor does it include a built-in draw function for graph visualization out of the box. It solely offers functions to generate graph representations in file formats like PNG, SVG, and DOT. Moreover, there are only a few interactive graph libraries available that support graphviz.

Despite these limitations, an HTML interactive graph library called "dash-interactive-graphviz" [5] was discovered, which supports graphviz. This library enables responsiveness and interactivity for dot files generated by graphviz. It relies on python dash, which is an open-source framework for building data visualization interfaces.

However, as good as it sounds, it is worth mentioning that this library has certain issues that need to be addressed in future work. As of 2023, the library is still experimental, indicated by its version number 0.3.0. Furthermore, there is no comprehensive documentation listing all the available functions to adjust the graph viewer's behavior. Consequently, the clickable area of the nodes is limited to a small rectangle in the middle, and there is no provision for changing the cursor when hovering over nodes or disabling animations, among other limitations. Additionally, the library has not received any updates since January 2021. Despite these shortcomings, it remains the best-suited library found for this project's requirements.

3.2.5 MVC Pattern

An effort was made to implement a variation of the MVC pattern to improve code quality and make unittesting easier as described in Section 3.4. The MVC

[5] https://pypi.org/project/dash-interactive-graphviz/, accessed 2023-06-21

pattern is a design pattern that specifies that an application is divided into data model, presentation information and control information[6].

The model contains the pure application data and no presentation information. In this case the model is the HeuristicMining class. (cf. Figure 2)

The view is responsible for presenting the data and usually does not know the meaning behind the data or how to manipulate it. It only displays the graph given by the controller.

And the controller processes the events triggered by the view, manipulates the model and gives the view the correct information.

The main benefit of explicitly implementing the pattern, instead of putting the controller logic in the view class, is to maintain a cleaner, more readable code base and setting an example for future extensions.

3.2.6 Extensibility for new algorithms

One of the key requirements of this project is the easiness with which new algorithms can be added to the project, meaning adding the name of the algorithm to the selection and creating a new view with its own sliders and other manipulators, as well as the algorithm itself that needs to generate the graph.

For this purpose an interface class called AlgorithmViewInterface, which all the new algorithm views need to inherit from, was created. (cf. Figure 2). It includes a startMining function that receives a preprocessed list of cases and generates its algorithm class with it. A loadModel function that prompts the user to select a saved algorithm class instance and loads it as its own algorithm class. 3 file generation functions for png,svg and dot files. And last but not least a clear function that gets called when the view is not needed anymore.

Any new view class is responsible for providing buttons for saving and exporting the project, parameter sliders and graph display widget. In the case of the HeuristicGraphView the HTMLWidget class is used for the graph display and the HeuristicMining class is used as the algorithm class.

The algorithm class is responsible for creating the graph instance with the given parameters and a List(List(cases)). In this case a graphviz digraph. This separation of View and Model makes it possible to save the model (the algorithm class) as a pickle instance, so that it can be loaded again later.

After creating the view and model classes, adding them to the project is fairly simple: Add the name of the algorithm and the algorithm view instance to the respective arrays in main.py shown in Figure 7.

3.2.7 Extensibility for new features

The app can also easily be extended upon with new features. The best example is the dot editor view, a feature that is entirely separate from the rest of the program. To use a newly created view, create an instance in mainWidget. Add a button to the File menu and link it to a new switch-to method that changes the mainWidgets current Widget to the new view.

[6] https://www.geeksforgeeks.org/mvc-design-pattern/, accessed 2023-07-11

```
# !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!ADD YOUR ALGORITHM HERE!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
# ADD NEW ALGORITHM NAME IN THIS algorithms ARRAY
# Create your algorithm view page like the heuristicGraphView below.
# AND THEN append() YOUR ALGORITHMVIEW TO THE algorithmViews ARRAY
# MAKE SURE THE INDEXING of both arrays match.
self.algorithms = ["Heuristic Mining",
                   "Heuristic Mining 2 (There is only 1 algorithm)"]
self.algorithmViews = []

# The BottomOperationInterfaceWrapper adds a bottom layout with 2 buttons for mining/loading models.
self.heuristicGraphView = BottomOperationInterfaceWrapper(
    self, HeuristicGraphView(self, 'saves/0/'), self.algorithms)
self.algorithmViews.append(self.heuristicGraphView)

# This is a placeholder for future algorithm views. DELETE OR REPLACE IT
self.heuristicGraphView2 = BottomOperationInterfaceWrapper(
    self, HeuristicGraphView(self, 'saves/1/'), self.algorithms)
self.algorithmViews.append(self.heuristicGraphView2)
```

Figure 7: Adding new algorithm

3.3 Usage Instruction

A detailed installation instruction can be found in README.md on Github [7]

3.4 Unit Testing

Unit testing is a crucial aspect of software development that plays a significant role in ensuring the quality and reliability of code. In this project, unit testing has been implemented for the HeuristicController class, which was specifically designed to facilitate the Model View Controller (MVC) pattern, enabling unit testing of the loadModel() function. The purpose of the loadModel() function is to load a pickle file in order to recreate the HeuristicMining class, which serves as the Model in this pattern.

Four types of tests have been implemented for the HeuristicController. The first test utilizes different preprocessed cases stored in text files to verify the integrity of the generated graphs [8]. The second test examines the integrity of the graph under various slider parameters. The third test assesses the program's capacity by utilizing a large sample data set with over 7000 entries. Lastly, the fourth and final test focuses on the loadModel() function. It involves loading a prepared pickle test file and verifying the integrity of the loaded HeuristicMining class by invoking its methods and asserting the validity of their outputs.

How to run the unit test is also described in README.md on Github.

Additionally, there is a small main_widget_prototype.py file in the same folder as main.py that serves as a quick ui testing file. It can be used to test new widgets before adding them to the main project.

[7] https://github.com/ShiningVision/Process-Mining-Visualization, accessed 2023-07-07
[8] https://github.com/ShiningVision/Process-Mining-Visualization/tree/master/tests/testlogs

4 Evaluation and Discussion

In this section we will discuss the problems encountered during the project, dependency issues, what can be done better and the usage of AI assistance throughout the project.

4.1 Implementation Afterthoughts

The dash interactive graphviz library has some severe problems. As already mentioned in section 3.2.4, the clickable node area is limited to the rectangles of the text in the middle. It might be possible to reverse engineer the library to make it suit the project use case. But doing so is out of the scope of this Bachelor project.

The Graphviz software installation is not exactly easy. Users might have to manually add it to their systems PATH depending on the operating system. The python graphviz library relies on the Graphviz software being preinstalled. But to date, there is no suitable substitute for Graphviz.

The approach to make an interactive responsive graph using HTML Callbacks comes with its own set of potential issues. Firstly, a http port must be selected to host the HTML page. If there are multiple threads running, each thread needs to reserve its own port. If the chosen port is not available, or for some reason the QWebengine used to import websites into the app can't access the site, the program might, at worst, show a blank page.

4.2 The role of ChatGPT

This Bachelor project is among the first ones to have the opportunity to use ChatGPT [9]. In the following sections I will showcase some of the prompts I used to make my life a lot easier.

[9] https://openai.com/blog/chatgpt, accessed 2023-07-07

4.2.1 Writing a Python Desk-Application Quickstart

At the beginning of the project, I had little to no experience or knowledge about how to create a Destop App in Python. Reading through tutorials and documentation takes a lot of time. If only their was a way to quickly generate boilerplate code tailored to my needs with easy to understand comments at every function to help me understand the syntax much faster! So I asked ChatGPT exactly that as seen in Figure 8).

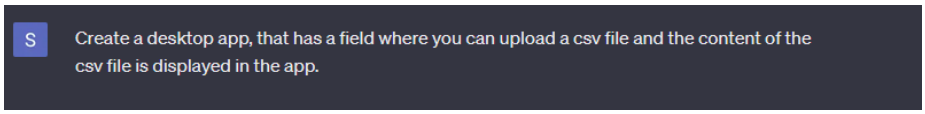

Figure 8: Quickstart Query

```python
class MainWindow(QMainWindow):
    def __init__(self):
        super().__init__()

        # Set up the user interface
        self.table = QTableWidget(self)
        self.setCentralWidget(self.table)

        # Add a file menu to allow users to upload a CSV file
        file_menu = self.menuBar().addMenu("File")
        upload_action = file_menu.addAction("Upload CSV File")
        upload_action.triggered.connect(self.upload_file)

        # Set the window title and show the window
        self.setWindowTitle("CSV Viewer")
        self.show()
```

```
def upload_file(self):
    # Open a file dialog to allow users to select a CSV file
    filename, _ = QFileDialog.getOpenFileName(self, "Open CSV File", "",

    # If the user cancels the file dialog, return
    if not filename:
        return

    # Parse the CSV file and store it in memory
    with open(filename, newline='') as csvfile:
        reader = csv.reader(csvfile, delimiter=',', quotechar='"')
        data = []
        for row in reader:
            data.append(row)

    # Display the contents of the CSV file in the app
    self.table.setRowCount(len(data))
    self.table.setColumnCount(len(data[0]))
    for i, row in enumerate(data):
        for j, val in enumerate(row):
            item = QTableWidgetItem(val)
            self.table.setItem(i, j, item)
```

Figure 9: Quickstart Answer

4.2.2 Autocomplete Code

Another way to avoid speed up code writing is to use ChatGPT as an autocompletion tool like in Figure 10).:

S

```
class StartView(QWidget):
    def __init__(self, parent):
        super().__init__()
        self.parent = parent

        #welcome text:
        self.figure = plt.figure(figsize=(50, 50))
        welcomeText = "This is a process mining tool.\nIt can create nice
looking graphs out of CSV files!"
        plt.text(0.5, 0.5, welcomeText, fontsize=24, ha='center',
va='center')
        plt.axis('off')
        self.canvas = FigureCanvas(self.figure)

        #two buttons 'LOAD EXISTING PROCESS' 'MINE NEW PROCESS
FROM CSV':
        #TODO

        # QVBoxLayout with welcome text on top and two buttons at
bottom:


complete this class
```

Figure 10: Autocompletion Query

```
class StartView(QWidget):
    def __init__(self, parent):
        super().__init__()
        self.parent = parent

        # Welcome text
        self.figure = plt.figure(figsize=(50, 50))
        welcome_text = "This is a process mining tool.\nIt can create nice
        plt.text(0.5, 0.5, welcome_text, fontsize=24, ha='center', va='cent
        plt.axis('off')
        self.canvas = FigureCanvas(self.figure)

        # Two buttons 'LOAD EXISTING PROCESS' 'MINE NEW PROCESS FROM CSV'
        load_button = QPushButton("LOAD EXISTING PROCESS")
        load_button.clicked.connect(self.load_existing_process)

        mine_button = QPushButton("MINE NEW PROCESS FROM CSV")
        mine_button.clicked.connect(self.mine_new_process)

        # QVBoxLayout with welcome text on top and two buttons at bottom
        layout = QVBoxLayout()
        layout.addWidget(self.canvas)
        layout.addWidget(load_button)
        layout.addWidget(mine_button)
        self.setLayout(layout)

    def load_existing_process(self):
        # TODO: Implement loading of existing process
        pass

    def mine_new_process(self):
        # TODO: Implement mining of new process from CSV
```

Figure 11: Autocompletion Answer

4.2.3 Introduction to new libraries

When I wanted to use the pickle library to save and load class instances and wanted a quick example (cf. Figure 12):

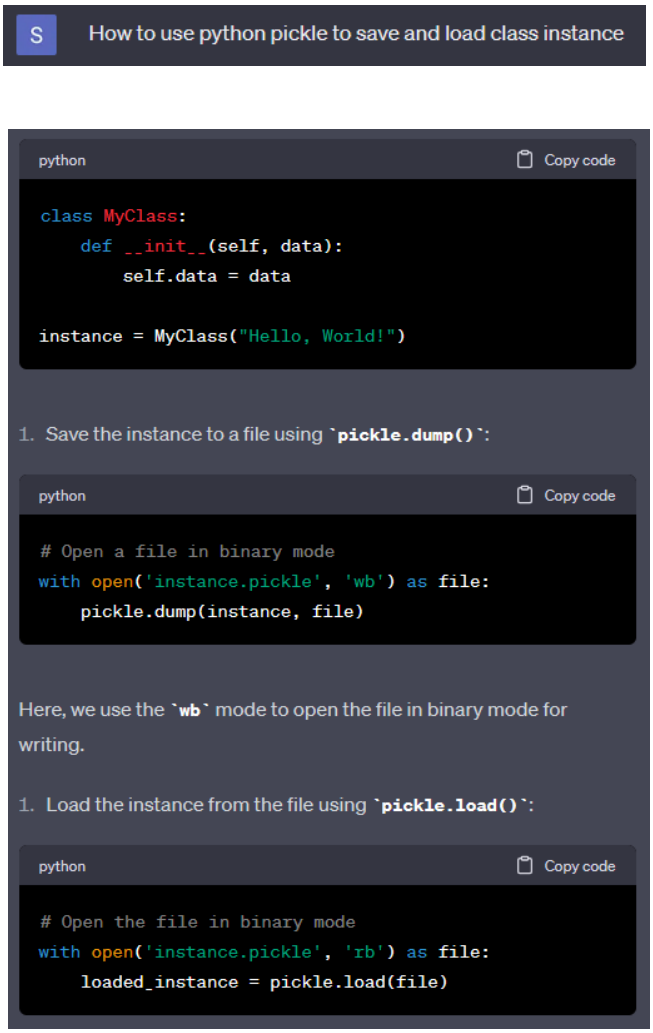

Figure 12: Pickle Example

4.2.4 Using Python without learning Python

When I wanted to easily add new algorithm to the project, I needed abstract classes. The initial answer from ChatGPT didn't work, so I fed ChatGPT with the error. The solution even turned out to be very complicated. I saved myself a lot of time and nerves. Figure 13).

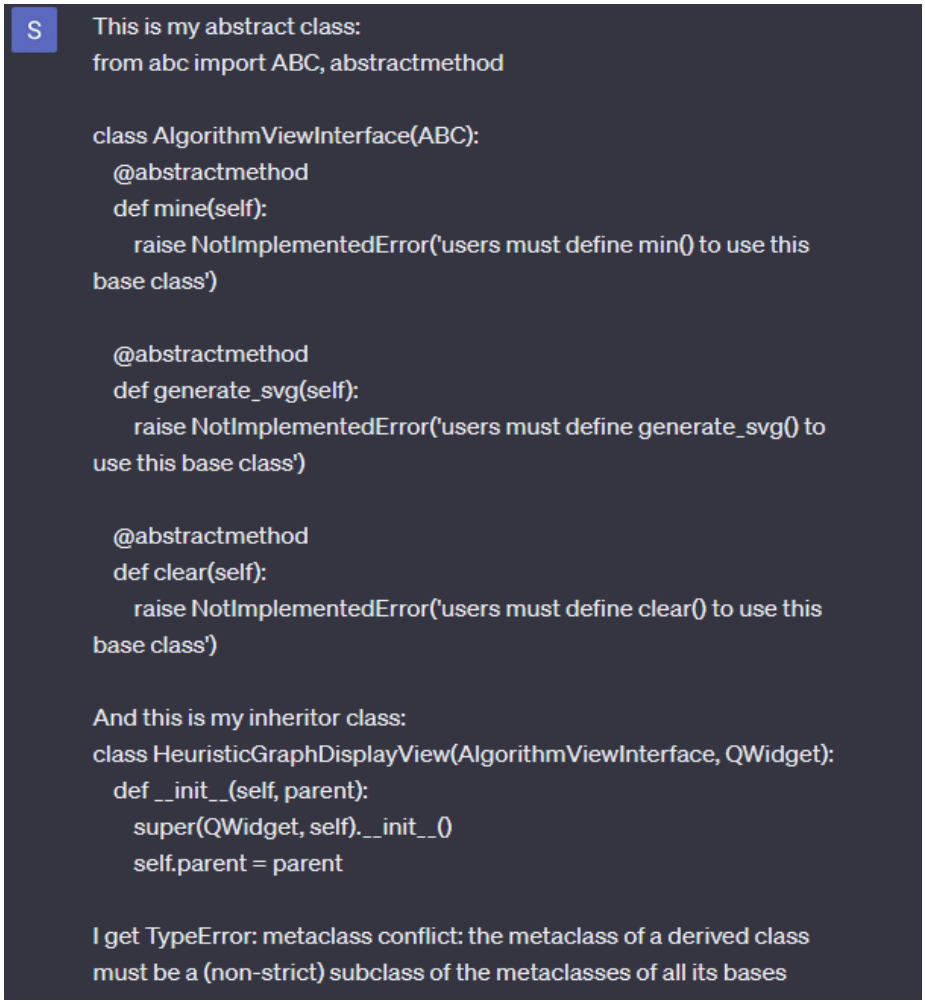

Figure 13: Python Abstract Query

To resolve the metaclass conflict, you can define a new metaclass that inherits from both `ABCMeta` and the metaclass of `QWidget`, like this:

ruby Copy code

```ruby
from abc import ABC, ABCMeta, abstractmethod
from PyQt5.QtWidgets import QWidget

class HybridMeta(ABCMeta, type(QWidget)):
    pass

class AlgorithmViewInterface(ABC, metaclass=HybridMeta):
    @abstractmethod
    def mine(self):
        raise NotImplementedError('users must define min(

    @abstractmethod
    def generate_svg(self):
        raise NotImplementedError('users must define gene

    @abstractmethod
    def clear(self):
        raise NotImplementedError('users must define clea

class HeuristicGraphDisplayView(QWidget, AlgorithmViewInt
    def __init__(self, parent):
        super().__init__(parent=parent)
```

Figure 14: Python Abstract Answer

4.2.5 Letting ChatGPT write your test cases

This is probably the most hated task programmers have to do. Writing test cases is time consuming and do not bring immediate progress results. Especially if it is done to ensure other people know they messed up when modifying your code in the future. (cf. Figure 15).

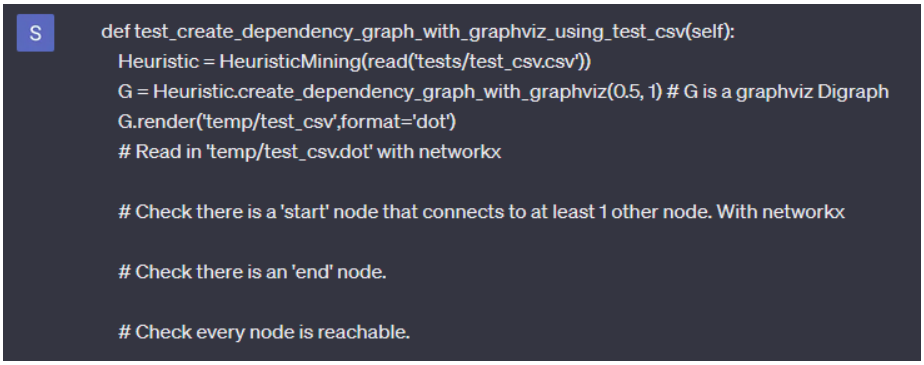

Figure 15: Test case Query

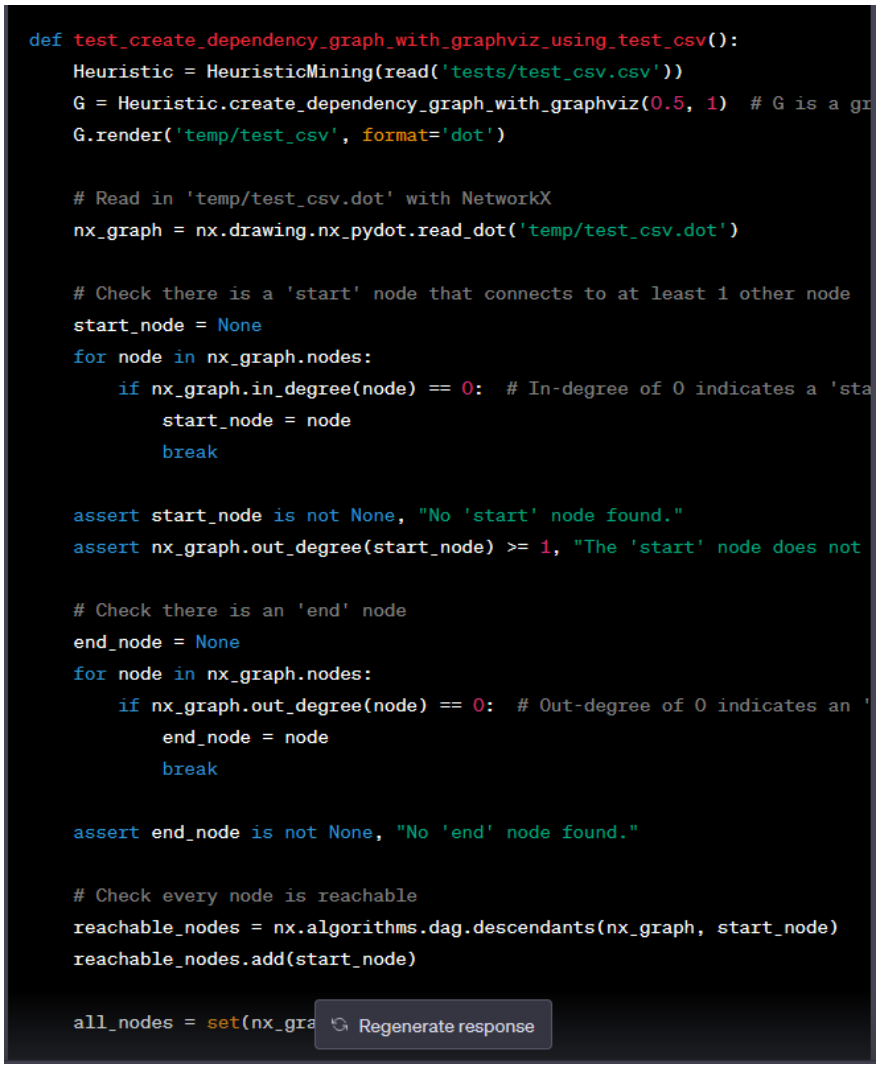

Figure 16: Test Case Answer

4.2.6 Limits of ChatGPT

ChatGPT is a very powerful tool for programmers to cut time reading documentation and get more work done. But as the technology is still very new, it can't fully replace programmers ... yet. There are a many problems that it cant't solve:

- The knowledge of ChatGPT is limited to 2021 (as of the date of this thesis). Any libraries created after that are likely unknown to ChatGPT. This was a problem for me since the dash-interactive-graphviz library was published in 2021. And there were no equivalent libraries for my use case. In the end I had to research with good old Google again.
- ChatGPT's knowledge of a lot of libraries are deprecated. When confronted with code using functions not known to ChatGPT, it does not tell you that it doesn't know how to use those functions. It will instead write imaginative code that obviously does not work.
- Code dumping of more than a hundred lines of code caused ChatGPT more often than not to cut the answer in half, upon reaching timeout.
- Using ChatGPT without thinking about software engineering can lead to some heavy technical debt. Programmers still have to structure and refactor the code manually.

5 Conclusion and Future Work

In conclusion, this project has laid the foundation for an open-source business process visualization desktop application, taking inspiration from the Fluxicon DISCO app. The initial implementation includes the Heuristic Mining Algorithm, a powerful mining algorithm for process discovery. The application is developed using the Python programming language and incorporates the PyQt5 library for its user interface. Additionally, the interactive graph display is achieved through the dash-interactive-graphviz library.

Moving forward, there are several exciting avenues for future work and enhancements to be explored. Some of the key areas to focus on include:

1. Algorithm Expansion: The application can benefit from incorporating additional mining algorithms to provide users with a broader range of options for process discovery and analysis. This can involve exploring other established algorithms in the field and implementing them within the application.
2. Enhanced User Experience: Improving the user interface and overall user experience is essential for the application's success. One aspect to consider is expanding the clickable area of the nodes within the graph, making it more intuitive and convenient for users to interact with the process models.

Additionally, further enhancing the information provided in the pop-up messages about the nodes can provide users with more detailed insights.

3. New Features: Introducing new features can greatly enhance the functionality and value of the application. For example, incorporating the ability to display case coverage instead of appearance frequency in the graph, can provide users with a more comprehensive understanding of the process models. Identifying and implementing other innovative features can further improve the application's capabilities. A good reference are DISCO's [10] many different features.

4. Design Improvements: Continuously refining the visual design and aesthetics of the application is crucial for creating a polished and professional user interface. Exploring different design approaches and incorporating user feedback can help to create a visually appealing and intuitive application.

By addressing these areas, the application can evolve into a comprehensive tool that empowers users to efficiently visualize and analyze business processes, thereby facilitating process optimization and decision-making.

[10] https://fluxicon.com/disco/, accessed 2023-07-07

References

[1] Dongkuan Xu, Y. T. A comprehensive survey of clustering algorithms. *Annals of Data Science* (2015).

[2] Marian Lux, S. R. M. Ddcal: Evenly distributing data into low variance clusters based on iterative feature scaling. *Journal of Classification* (2023).

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