Graph Based System for Evidential Reasoning

Divyarajsinh Chauhan

Follow this and additional works at: https://scholarworks.sjsu.edu/etd_projects

Part of the Other Computer Engineering Commons
Graph Based System for Evidential Reasoning

A Project Report
Presented to

Professor Leonard Wesley
Department of Computer Science
San Jose State University

In Partial Fulfillment
Of the Requirements for the Class

Spring-2023: CS 298
By
Divyarajsinh Chauhan

Dec 2023
The Designated Project Committee Approves the Project Titled
Graph Based System for Evidential Reasoning

By
Divyarajsinh Chauhan

Approved for the Department of Computer Science
San Jose State University
Dec 2023

Dr. Leonard Wesley  Department of Computer Science
Dr. Jon Pearce  Department of Computer Science
Dr. William Andreopoulos  Department of Computer Science
ABSTRACT

In the modern data driven world, graph editing tools have become very essential as they provide means to understand, visualize and manipulate complex relationships between various datasets. They have especially played a crucial role in the space of evidential reasoning, where it has made a significant impact in the decision making process by developers, analysts and researchers to understand and represent the connection in the data. Existing tools fail to handle huge amounts of data efficiently and also don’t have the features required to handle tasks related to evidential reasoning. To address these gaps, we developed Pygrapher Web UI tool. We developed it using technologies such as Vis.js, React, JSX and CSS. This tool is designed to provide the users an easy and interactive interface which can help them create complex graphs by manipulating the nodes and edges of the graph. The tool provides both robustness as well as is much more efficient in handling large amounts of data. The Data Management Layer of the Pygrapher, is supported by a reliable File System, that ensures that the data is handled securely. Future developments of PyGrapher will focus on integration with external systems and incorporation of a query language. These enhancements will be based on the user feedback and evaluations of the tool. We will ensure that PyGrapher continues to evolve and meet the changing demands in the field of Evidential Reasoning.

Index Terms: Graph Editing Tools, Evidential Reasoning, Dynamic Graph Visualization, Vis.js.
ACKNOWLEDGEMENTS

First and foremost, I would like to express my profound gratitude to Professor Leonard Wesley, for his ongoing support of my Project study and research as well as for his tolerance, inspiration, passion, and vast knowledge. Throughout the entire research and report-writing process, his advice was invaluable to me. A greater mentor and advisor for my Project study could not have been conceived in my opinion.

In addition to my committee chair, I would like to thank the other members of my project committee, William Andreopoulos and Jon Pearce, who provided me with advice for the project work as well as constructive criticism and ideas to assist me finish my project.
# Table of Contents

**ABSTRACT**

**ACKNOWLEDGEMENTS**

1. **INTRODUCTION**  
2. **PROBLEM STATEMENT**  
3. **LITERATURE REVIEW**  
4. **METHODOLOGY**  
5. **IMPLEMENTATION**  
6. **CONCLUSION**  
7. **FUTURE WORK**  
8. **ACCOMPLISHMENTS**  
9. **REFERENCES**
# Table of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Grasper-CL User Interface</td>
<td>6</td>
</tr>
<tr>
<td>2</td>
<td>Tinkerpop Graph UI</td>
<td>7</td>
</tr>
<tr>
<td>3</td>
<td>Cytoscape Graph Interface</td>
<td>9</td>
</tr>
<tr>
<td>4</td>
<td>CAPRI Input TXT file</td>
<td>12</td>
</tr>
<tr>
<td>5</td>
<td>System Architecture</td>
<td>13</td>
</tr>
<tr>
<td>6</td>
<td>Technical Workflow Diagram</td>
<td>15</td>
</tr>
<tr>
<td>7</td>
<td>Evidential Reasoning Graph in Pygrapher</td>
<td>16</td>
</tr>
<tr>
<td>8</td>
<td>Add a Node</td>
<td>18</td>
</tr>
<tr>
<td>9</td>
<td>Add an Edge</td>
<td>18</td>
</tr>
<tr>
<td>10</td>
<td>Edit a Node</td>
<td>19</td>
</tr>
<tr>
<td>11</td>
<td>Delete a Node</td>
<td>20</td>
</tr>
<tr>
<td>12</td>
<td>Node Customization</td>
<td>21</td>
</tr>
<tr>
<td>13</td>
<td>New Graph Pop-Up</td>
<td>22</td>
</tr>
<tr>
<td>14</td>
<td>Graph Saving and Loading</td>
<td>22</td>
</tr>
<tr>
<td>15</td>
<td>Graph Object Structure</td>
<td>24</td>
</tr>
</tbody>
</table>
1. INTRODUCTION

The much needed result oriented and decision making tool is essential especially the ones that are capable of specialized and intangible graph modeling. The project is a presentation of designed PyGrapher enabled to carry out demand of the Evidential reasoning models [1]. PyGrapher was built as a result of observed issues aligned with TinkerPop [2] and Cytoscape [3]. The initiated development also satisfies the requirements of CAPRI.

Since there are limitations in the existing available tools, the development of PyGrapher is an attempt to create. Considering the objective of defeating performance limitations, a JavaScript React was used to make sure it is fast and able to handle large amounts of data. The graph data will be represented with an established text file format via CAPRI and to fulfill this PyGrapher was purposely designed. A requirement introduced by sophisticated probabilistic reasoning framework was attained by a tool responsible for atomization of built graphs data to the regulated input format. The entire process of generating graphs, sophisticated probabilistic reasoning and conversion was delivered by PyGrapher effortlessly.

The objective is to achieve inclusive and accommodating tools utilizing evidentiary reasoning. It also conquers the gap between probabilistic analysis and simple graph modeling. To reduce the main efforts in the process the new feature will be introduced in
future to convert JSON to TXT. Further in the project important elements of PyGrapher will be analyzed including its evidential reasoning showing the project’s progress.
2. PROBLEM STATEMENT

Existing tools utilized with Evidential Reasoning models are lacking efficient graph management and user oriented interfaces. Some Evidential Reasoning models require the graphs to be in a particular format which can then be translated and combined to carry out the analysis. To fill the limitation it is essential to build and analyze intricate graphs.

The main limitations include a large amount of graph data, less support for graph conversions and updates, and an inadequate user oriented interface for constructing and displaying graphs in the tools available in the present. The graph tool has shown its ability to make adjustments when compared to other choices together with Tinkerpop, Cytoscape, NetworkX [7], OriginLab [8], Graphlytic [9], and The GrasperaCL Graph Management System [10]. Keeping user experience in mind, the tool assures the performance together with making, loading, and saving graphs are simple and functional. To make the distinct tool important, conditions were made to be closely tied to Evidentiary Reasoning, along with flow of generated graphs into Evidential Reasoning. For the purpose of academic study, data analysis and visual presentation, the tool provides an alternative to use graph tool market achieving all the consumer demands for which alternative tools failed.
To generate, inspect, and maintain graphs for Shafer Dempster and Evidential Reasoning this tool offers a solution as a graphical user interface (UI) which will increase work efficiency of professionals and students. Developed tools can complete the task with no difficulty including creating graphs, defining evidentiary weights, visualizing belief propagation, and transforming graphs. Along with managing large amounts of graph data with proficiency, the tool also allows numerous graph layout techniques for flawless presentation.

The graph based UI tool offers a solution to the limitation with capability to analyze complicated Evidential Relationships, allowing decision making power under tentativeness, with obtaining thorough observation using Shafer Dempster Analysis with improvement.
3. LITERATURE REVIEW

The dynamic world we live in today requires data analysis and knowledge portrayal, to fulfill that requirement it is significant to have strong graph generation and maintenance tools. To overcome problems from complicated network analysis to functional database administration, technologies are very important in the dynamic domain. The present tools in this domain present significant characteristics, advantages and restrictions and it is required to inspect those. The unique characteristics of TinkerPop, Cytoscape, GraphCL and PyGrapher will be compared further. The comparison will highlight the functions of the existing tools in the domain of graph based applications. The aim is to provide a thorough explanation of similarities of each other. The explanation is important for the users when it comes to select the best tool for their specific needs, tasks and field.

3.1 GrasperCL

With providing multiple options including generating, visualizing, and adjusting graphs with effortlessness using interactive environments these skills sets GraphCL apart. Its suitability for quick prototyping and instructional purposes is demonstrated by this feature. It's an excellent tool for experimenting and researching with graph data since users can easily generate a variety of graph topologies using it.
Even yet, GraphCL might not have the sophisticated features and specialized assistance needed to meet the unique requirements of evidentiary reasoning models. When dealing with sophisticated and large-scale graphs, there may be performance issues when addressing complicated evidential reasoning scenarios. In addition, it's possible that GraphCL won't support the strict format requirements needed for evidential reasoning by default, requiring extra data preprocessing. Its responsiveness is also restricted, as it makes use of antiquated UI technology.
3.2 Tinkerpop

With its many features, TinkerPop is a flexible framework for graph computing that makes managing graph data easy. Because it is extendable and flexible, it has been widely recognized and used in the fields of graph databases and graph processing. Its emphasis on graph traversal, which enables users to effectively query and manage complex graph structures using the Gremlin query language, is one of its distinguishing characteristics. Gremlin is an effective tool for in-depth data exploration and analysis because it offers a strong and expressive vocabulary for graph traversal.

Figure 2: Tinkerpop Graph UI

TinkerPop is an excellent tool for database administration and graph processing since it lets users do sophisticated things with graph data. It might not, however, place the same emphasis on intuitive graph modeling as PyGrapher. Although very skilled in graph
building, TinkerPop might not offer as many interactive and user-friendly graph modeling tools. Furthermore, TinkerPop's wide range of features might not be tailored to the unique needs of evidential reasoning, which often requires a specific graph representation style.

3.3 Cytoscape

Cytoscape has garnered widespread acclaim for its prowess in network visualization and analysis, particularly in domains such as biology, bioinformatics, and social network analysis. It excels in rendering complex networks in visually engaging layouts, empowering researchers and analysts to derive insights from network data. Cytoscape boasts an extensive ecosystem of plugins that offer diverse features for indepth network analysis, rendering it an invaluable resource for scientific investigations.
While Cytoscape excels in network visualization, it falls short for the specific needs of evidential reasoning, particularly in handling Dempster-Shafer theory-based probabilistic reasoning. Its capabilities may not fully support the unique data formats and intricate probabilistic computations required in evidential reasoning models. Additionally, Cytoscape lacks the specialized user interface and direct data processing features, like JSON to TXT conversion, crucial for seamless integration with systems like CAPRI. These gaps in functionality and customization underscore the necessity for a dedicated tool like PyGrapher, tailored to the specific requirements of evidential reasoning and advanced data analysis.

3.4 Pygrapher

PyGrapher, developed as part of this project, encompasses a distinctive set of strengths. Rooted in the React framework, PyGrapher offers speed, efficiency, and an intuitive user interface. Its design is meticulously tailored to confront the intricacies of evidential reasoning—an area demanding precise attention. PyGrapher aptly acknowledges the critical importance of representing graph data in a specialized text file format, seamlessly aligning with the stringent mandates of evidential reasoning models. The Pygrapher tool has been discussed in more detail in the next section.
4. METHODOLOGY

This section presents the methodology employed in the development of the graph-based UI tool for Evidential Reasoning.

4.1 Introduction to Pygrapher (Web)

The need of PyGrapher as a new graph editor is driven by recognized limitations in existing tools, such as TinkerPop [3] and Cytoscape [4]. While these tools have served certain graph-related functions well, they exhibit notable shortcomings. TinkerPop, while robust for graph processing, can be challenging for users seeking a more intuitive and user-friendly graph creation interface. Its focus on graph processing might not align seamlessly with the needs of users in specialized domains like evidential reasoning. On the other hand, Cytoscape, known for its network visualization capabilities, may lack the desired speed and efficiency, particularly when dealing with large and complex graphs. Furthermore, both TinkerPop and Cytoscape may not offer the tailored support required for evidential reasoning models, often necessitating specific input formats like TXT file. In contrast, PyGrapher, with its React-based architecture, addresses these limitations by providing a fast, efficient, and specialized tool tailored for evidential reasoning, offering a significant advancement in the realm of graph editors. The integration of a JSON to TXT converter into the web UI and GUI, as part of future work, further emphasizes PyGrapher's commitment to overcoming the identified shortcomings and providing an enhanced user experience for graph-based applications.
PyGrapher, built on the React framework, offers better performance compared to several off-the-shelf systems. The inherent efficiency of React, with its virtual DOM and streamlined rendering mechanisms, contributes to a smoother and more responsive user experience, particularly when handling intricate graphs. Moreover, PyGrapher's tailored design for evidential reasoning ensures that its performance is optimized for the unique demands of this domain. While individual performance metrics may vary, the combination of React and PyGrapher's specialization translates into an overall robust and efficient performance that meets the expectations of users engaged in graph-based applications.

---

```plaintext
#FILENAME: analysis_spec.txt
# This file contains the evidential analysis specs
# for an evidential HMM (eHMM)
# ANALYSISNAME: Evidential_HMM_Analysis
### INPUT NODES ###
# START Node
STARTANALYSISNODE:
ANALYSISNODENAME: START
ANALYSISNODETYPE: Input
FRAME_NAME: HSF
FROMANALYSISNODES: None
TOANALYSISNODES: DIS_1
ENDANALYSISNODE:

# S1_EMITION Node
STARTANALYSISNODE:
ANALYSISNODENAME: S1_EMITION
ANALYSISNODETYPE: Input
FRAME_NAME: S1_EMITION
FROMANALYSISNODES: None
TOANALYSISNODES: DIS_2
ENDANALYSISNODE:
```

Figure 4: CAPRI Input TXT file
Pygrapher serves a specific purpose which is to provide an intuitive and user-friendly platform for creating, visualizing, and analyzing graphs, with a particular emphasis on supporting evidential reasoning. Its adaptability to the unique requirements of evidential reasoning models positions PyGrapher as a valuable tool for users seeking to translate their graph data seamlessly into the necessary txt file format, for advanced analysis. Refer to the sample graph input txt file which is used by the CAPRI system for analysis in the given figure. A significant advancement on the horizon is the integration of a JSON to TXT converter directly into both the web UI and GUI, further streamlining the evidential reasoning workflow. By offering a specialized environment that aligns with the complexities of evidential reasoning, PyGrapher empowers users to make informed decisions and derive meaningful insights from intricate graph-based scenarios.

4.2 System Architecture

Figure 5: System Architecture
Within the architecture, users engage in the graph creation process through Pygrapher Web and Pygrapher GUI. My primary focus was on the creation of Pygrapher Web, a web-based tool designed to offer users an accessible platform for crafting and visualizing graphs directly through a browser interface. This web centric tool enhances usability, allowing users to construct intricate graphs for scenarios related to evidential reasoning in a user-friendly web environment.

After users create a graph on Grapher Web, they can save their work in JSON format, encapsulating the graph's structure and relevant data. The subsequent transformation of this JSON file into a TXT file, a task facilitated by the Pygrapher Input Converter component, ensures a standardized and universally readable format for further processing.

The TXT file then serves as input for the CAPRI backend, the core of the system. In this backend, developed to execute core functionalities, sophisticated algorithms rooted in Dempster-Shafer theory come into play. These algorithms manage the translation and combination processes, leveraging the probabilistic reasoning inherent in Dempster-Shafer theory to synthesize and analyze information from the user-generated graph. This systematic flow, starting with user interaction on Pygrapher Web and culminating in advanced probabilistic reasoning within the CAPRI backend, establishes a
comprehensive tool for decision-making in scenarios involving complex graph-based evidential reasoning.

4.3 Technical Workflow Diagram

There are two main sections in the Pygrapher Web UI: The User Interface and the Data Management Layer. In the application, React.js is used which has various react components associated with it for modularity. The visualization and interaction of the graph is facilitated by the 'Graph Canvas Window' and it directly interacts with the Vis.js library. Then there is another component called ‘Graph Navigation Bar’ which is responsible for new graph creation, graph loading, graph saving and editing the graph attributes like graph name, project name and the type of analysis. Besides that, on the right side of the window, there is the node/edge customization bar which is used to change the attributes of a node or an edge including the shape, size, color, node/edge properties among others.
In the Data Management Layer, there is ‘File System’, which is used to store the graphs in the form of JSON and also to load them back into the application. It helps in data persistence and graphs can be saved or loaded without any dependency on the network since it is not stored on any cloud. Also, in terms of safety, since the data doesn’t travel over the network, the chances of data theft are also very low. Increasing the reliability of incase of local system failure is part of future research for this tool.

Figure 6: Internal Workflow Diagram
4.4 Using Pygrapher

In the context of evidential reasoning, the graph tool plays a crucial role in conducting a thorough analysis of health-related variables by constructing a well-organized graph. Within this graph, three fundamental input frames—Temperature, Blood Pressure, and Lipid Levels—function as nodes, each representing a distinct random variable with various associated propositions. For example, the Temperature frame encompasses propositions such as low, medium, and high.

![Evidential Reasoning Graph in Pygrapher](image)

Figure 7: Evidential Reasoning Graph in Pygrapher

To facilitate effective reasoning and information exchange among these input frames, compatibility relations are established. These relations act as a channel, facilitating the translation of mass distributions from one frame to another. The compatibility relation is
particularly pivotal in the creation of a cross product frame, where the mass distributions from the Temperature and Blood Pressure frames seamlessly come together.

Following this integration, the translated mass distributions from Lipid Levels are amalgamated with the cross-product frame (Temperature * Blood Pressure). This process involves the application of the Dempster-Shafer combination rule, a method designed to ensure a coherent and consistent synthesis of evidence across diverse frames. This step aptly captures the collective information derived from the various input variables.

The methodology is then extended to the ultimate output frame, representing the random variable "Disease." Here, the translated and combined mass distributions from the previous frame (Lipid Levels * (Temperature * Blood Pressure)) are utilized to calculate probabilities associated with different diseases. This final output frame serves as a comprehensive indicator of potential health outcomes based on the interplay of temperature, blood pressure, and lipid levels.

This approach to evidential reasoning provides a visual representation of the complex relationships and dependencies among health variables. The graph structure not only facilitates a comprehensive understanding of probabilistic outcomes but also establishes a transparent and interpretable framework for decision-making in the realm of medical diagnostics. The systematic application of compatibility relations and combination rules
within the graph tool contributes to a robust and coherent evidential reasoning process, positioning it as a valuable tool for healthcare professionals and researchers in the field.

The following sections outline the key features available in the graph UI tool:

4.4.1 Add Nodes and Edges

Users can easily add nodes and edges to the graph by selecting the appropriate options in the tool's interface. This feature allows for the creation of complex graph structures that represent the relationships and connections between different entities or concepts.

![Figure 8: Add a Node](image)
4.4.2 Edit Nodes and Edges

The tool gives the option to edit existing nodes and edges within the graph. Users can modify the properties of nodes, such as changing the label, color, size, shape, and additional properties stored in JSON format. Similarly, edges can be edited to adjust their appearance and properties.
4.4.3 Delete Nodes and Edges

When nodes and edges are no longer required, users can remove them from the graph. With the help of this function, users can dynamically alter the graph structure and improve their analysis by eliminating unnecessary data. When changing a node or edge, the user has the option to delete it by using the delete button found in the customisation window.
4.4.4 Customization Options

The graph UI tool offers various customization options to enhance the visual representation of the graph. Users can choose from a range of colors, shapes, and sizes to differentiate nodes based on their attributes or significance. This feature helps in conveying information effectively and making the graph visually appealing.

Each node in the graph can have associated properties stored in JSON format. In the context of Evidential Reasoning, the properties would be the beliefs of the random variable. This feature enables users to attach additional information or metadata to
nodes, allowing for a more comprehensive representation of the underlying data. Users can define and update the properties of nodes to capture relevant attributes and details.

![Node Customization](image)

**Figure 12: Node Customization**

4.5 *New Graph Functionality:*

The user can choose to create a new graph by providing the graph name, the project name and the graph type. The graph type has two options: GALLERY and ANALYSIS.
4.5.1 Load and Save Graph Functionalities:

The graph UI tool provides the ability to export and import graph data, allowing users to save their work or share it with others. Users can export the graph as a file in a standard
format, such as JSON, that can be easily imported into the tool for further analysis or collaboration. This feature facilitates seamless sharing of graph structures and promotes interoperability with other graph analysis tools or platforms.

5. IMPLEMENTATION

The implementation included the use of the Vis.js library to visualize and manipulate the graph, React.js as a front-end framework coupled with JSX and CSS3. All of these technologies are explained in more detail in the below sections.

5.1 Vis.js

Network visualization is a technically designed tool that displays networks composed of interconnected nodes and edges. This user-friendly visualization platform offers functional personalization options, including varying node and edge shapes, colors, sizes, and even integrating pictures. Its compatibility extends across numerous contemporary web browsers, allowing it to handle up to few thousands nodes and edges without any issues. To deal with larger datasets, this tool employs clustering capabilities. The display itself relies on HTML canvas technology [11]. Within the context of vis.js, the network is represented as an object encompassing its constituent parts, including nodes, edges, and additional details.
5.2 React.js

React.js is a tool that helps developers make websites, especially for parts where users click and interact. It's been around since 2013, made by the folks at Facebook. The cool thing about React.js is that it lets you write your code in a mix of HTML and JavaScript, which makes things easier to understand and change later. It's like building with Lego blocks - you create small pieces and then put them together to make your website look great. It's also really fast because it only changes the parts of the website that need to be updated. Plus, it's easy to learn, and there's a big group of people using it who help each other out. This is why a lot of web developers like to use React.js for their projects.

5.3 Design Patterns Used

In terms of PyGrapher's design methodology, we adopted React's modular design principle. By using this technique, we had the ability to divide the application into independent manageable components, which made the codebase more organized and structured. To encourage code reuse and facilitate easy testing, each user interface component was handled as an independent module. The development cycle's feature and enhancement integration was facilitated by this modular strategy. Furthermore, to successfully handle the dynamic and interactive nature of graph data, we developed strong state management procedures. This was essential to maintain the application's responsiveness and accurate real-time reflection of visualization from the user activities.
The choice of design and approaches played a crucial role in overcoming the challenges we encountered and greatly aided in the productive development and functioning of PyGrapher.

5.4 Challenges Faced

In the process of developing the Pygrapher tool, one of the primary challenges we came across in coding was to maintain fixed positions of the nodes in the graph. This challenge occurred every time there were changes made to the node or edge component which caused unwanted reorganization of the graph. The nodes and the edges would automatically move positions, disturbing the user’s layout and the interface less intuitive for analyzing the models in evidential reasoning. We developed a strategy to capture and fix the node’s coordinates to overcome this challenge.

We made use of the 'dragEnd' event functionality given by the vis.js, and created a custom approach that helped us update the node’s X and Y coordinates every time they were altered or changed. This method helped in maintaining the graph’s consistency visually without altering the overall layout of the graph which is very important for data interpretation and analysis.

5.5 Performance Optimization
It is important to handle vast graph data efficiently to provide a good and smooth user experience. Pygrapher has several performance optimization techniques integrated with it, such as lazy loading and caching, in order to make sure that the user has quick and effortless interactions even if they are working with large and complex data.

5.6 Iterative Development Process

Keeping in mind all the inputs from experts and feedback from the users the Pygrapher tool for Evidential Reasoning was developed. This was an iterative process, this tool was powered by Vis.js library. This resulting approach made sure that the tool meet all the user’s needs and we also made sure that the platform was user friendly to create graphs, visualize them and also perform analysis to help in decision making. While this tool is fast and efficient, it may have some issues with the performance when working with larger graphs. We are working on overcoming this issue and increasing the tool’s effectiveness.
6. **CONCLUSION**

This study is focused on the development and deployment of Pygrapher which is a graph based UI tool. This tool is designed to help in the process of decision making and Evidential Reasoning. Pygrapher provides users an interactive and user-friendly UI for creating, visualizing and altering the structures of graphs that represent evidential relationships and beliefs.

We have made use of technologies such as React, Vis.js, HTML and CSS to make Pygrapher more interactive and dynamic in nature. This will help the users to create complex graph structures, understand and explore the data in depth which would in turn help in better decision making process. Some of the key components of Pygrapher include its ability to create, edit, delete the nodes and edges of the graph. It also allows the user to customize the properties of the node such as node’s color, node’s size, it’s label and shape.

Working with Evidential reasoning models has become for effective and efficient with the use of Pygrapher. It can be used to understand and analyze complex data and relationships in a easier and simple manner which helps the user make more informed decision when working with graph data.
The tool consists of key features such as drag & drop, interactive selection of node, automatic layout algorithms, undo and redo option to make changes to the nodes and also provides with the option to import/export the graph for further study and usage.

Pygrapher tool has applications in various domains such as risk assessment, predictive modeling and also in analyzing the data. This user-friendly usage of the tools makes it simple to be used by anyone working with large and complex data such as analyst, decision makers of companies and also researchers.

Going ahead, the features of Pygrapher can be enhanced with advanced editing options which can also be collaborative, synchronize graph data real time, and also provide data from external sources. We also look forward to creating a module named converter module which data from one format to another i.e from JSON to TXT format. This can be integrated with the front end and is understood by the CAPRI system.
7. Future work

For future work on Pygrapher tools, various improvisations can be in the area of user interaction for making it more simple by making the frontend of Pygrapher GUI and Pygrapher Web integrated with the JSON converter.

In the present scenario, the user has to download the JSON file manually and then feed it into the converter module. This process of improvisation would cut down the manual process for the users where we embed the JSON converter into the frontend UI’s. This implementation would not only make tool more efficient but also make the analysis and creation of graph with nodes and edges simpler.

Another prospect to explore with Pygrapher would be to check if moving to a graph database from JSON storage would be an appropriate choice. This enhancement would require in depth research on how to integrate graph database using open source graph databases such as Neo4j. Graph databases hold an advantage to provide more efficient graph querying ability when compared to the classic JSON storage. To make the system more dynamic and simple to run queries for data analysis and to generate insights it would be a better improvisation to store graph data within a graph database. This modification would not only improve the performance of the system but also increase it’s scalability especially when we are working with complex graph data.
We can also create more enhanced visual representations of the graphs to understand more complex connections in the graph data. This representation would not only help the user start the graph structure better but also help in analyzing the data. It is important to get constant feedback from the user after tool usage to understand if the tools are aligned to their requirement and also get feedback on where improvisation is required.

Furthermore one of the key features of the tool is to ensure if it can work smoothly with large scale data to produce the graphs. By increasing the processing speed of the graph data it enables the performance of the tool to be more optimized which makes the user experience to be much more convenient. By constantly adapting to the technology advancement we can make the tool a valuable asset and also draw insights from reasoning models.
In order to develop the Pygrapher - A web-based graph management tool, a step by step process was followed. The process began by understanding the application of graphing tools in the field of Evidential Reasoning. The process involved researching the existing tools and libraries available and doing Proof of Concepts for them. After finalization of the technologies and tools that were going to be used to create the Pygrapher, a requirement specification was created which consisted of all required functionalities in detail. Post that, the tool was developed iteratively with continuous user feedback. To make sure the tool is reliable, thorough testing was done including unit testing, functional testing and performance testing. The below points explain the development process in more details:

1. Proof-of-Concept: Different libraries and tools were analyzed by creating small demo projects of them. The libraries and tools included D3.js, Vis.js, Grasper-CL, TinkerPop and Cytoscape.
2. Comparison of Features: The features of different libraries were analyzed and compared to see how closely they align with the tool that I intended to develop.
3. Finalization of vis.js: Based on the study of these evaluations, Vis.js was chosen as the library that was going to be used to develop the Pygrapher tool. Several things were considered before finalization, some of them being the functionalities
available in the libraries, the kind of documentation they are offering and also the community support of them.

4. Implementation of Pygrapher: To develop the pygrapher, Vis.js was used for graph visualization and interaction. For higher responsiveness, modular architecture and ability to handle large amounts of data, React.js was used along with JSX and CSS.

5. Testing and Optimization: To test the tool, different testing techniques were used. It involved unit testing to test the individual components. Functional testing was also done for thorough testing of functionalities provided in the requirement specification. Performance testing was also to test how the application was performing under high data and load.

6. Apart from that, to deepen the understanding of the theoretical aspect, I went through different research papers to Evidential Reasoning to better understand the application of a graphing tool in this field. This research was important to better understand both the technical aspect and theoretical aspect of the project.
REFERENCES


