Visualisation of airline travel - STAR analysis and implementation report

Adam Černý, Stanislav Lamoš

May 2024

1 Problem specification

The primary challenge we encounter in this assignment is the effective visualization of a complex network representing airports and their interconnections. The challenge lies not only in creating an aesthetically pleasing visualization but also in ensuring its functionality in facilitating insightful analysis of airport sizes and flight patterns within the network. Therefore, the problem at hand demands a comprehensive approach that integrates data visualization principles with domain-specific knowledge of airport operations and network analysis techniques.

1.1 Dataset

We were provided a GraphML file with data about domestic flights in the USA. GraphML is a file format used for representing graph data structures. It's based on XML and provides a standardized way to describe graphs, including nodes, edges, and their attributes. The network, loaded in GraphML format, comprises nodes symbolizing airports and edges denoting flights between these airports. Each airport is associated with its geographical coordinates, adding an additional layer of information to consider.

1.2 Visualisation tasks

In order to effectively visualize the airport network and its corresponding flights, our primary objective is to create a visualization that provides insights into two key aspects: the size of each airport based on the number of flights it handles, and the connections between airports represented by the flights. Achieving these objectives necessitates tackling several interrelated issues, including data parsing, spatial representation and interpretability. To achieve this, we aim to design and implement a visualization that portrays each node as an airport, with the size of the node correlating to the volume of flights it serves. Additionally, we seek to clearly depict the edges connecting these nodes, symbolizing the direct flights between airports. This visualization will enable viewers to discern

not only the relative importance of airports based on flight frequency but also the network of connections facilitating air travel between different locations. By focusing on these elements, our visualization will facilitate the analysis of airport activity and the broader dynamics of air transportation networks.

2 State-of-the-art

2.1 Visualisation of graph data

Graph visualization is a crucial and versatile tool with extensive applications across various fields. From everyday encounters like navigating computer file hierarchies, which are represented as trees, to organizational charts and taxonomies showcasing species relationships, graphs are integral to understanding complex structures. They also play a vital role in web mapping, browsing history, evolutionary biology, and molecular chemistry. Additionally, graphs are used in diverse applications such as class browsers in object-oriented systems, compiler data structures, state-transition diagrams in real-time systems, and entity relationship diagrams like UML in database structures. This broad spectrum of applications highlights the importance of graph visualization in efficiently representing and analyzing interconnected data beyond hierarchical formats [2].

The key issue in graph visualization is dealing with the size of the graph. Large graphs present challenges such as performance limitations and usability problems. When a graph contains numerous elements, it can strain the viewing platform and make it hard to distinguish between nodes and edges. Even before reaching this point, usability becomes an issue because detailed analysis is easiest with smaller graphs. While displaying an entire large graph can provide an overview, it complicates comprehension and analysis of the data structure. [2]. Our dataset contains over 230 nodes and 2100 edges and therefore, we will have to deal with this limitation. However, we are aware that much larger and more complex graphs may exist, that are harder to visualise because of information overload. Besides information overload, it is important to note that cognitive limitations of humans play their role in graph visualisation [2].

Many authors, such as [7] or [4], address the problem of drawing a graph so that it can be interpreted as easily as possible. This problem is called graph drawing. The fundamental graph drawing challenge can be summarized as follows: When provided with a group of nodes interconnected by edges (or relations), determine the optimal positions for the nodes and the paths for each connecting edge to be drawn.

In our work, we will not have to deal with this challenge because we can map the airline data directly onto their real-world geographical coordinates of the nodes (airports) using a standard projection such as Mercator.

Graph visualization offers different approaches for navigation and interaction, each with distinct advantages and drawbacks. Zoom and pan are conventional methods that are particularly effective for exploring large graph struc-

tures. Zoom, in particular, is well-suited for graphs due to their relatively simple graphic representations, allowing for straightforward adjustment of screen transformations without aliasing issues. Geometric zooming enlarges graph content, while semantic zooming adjusts information detail dynamically. However, zoom and pan can present challenges in interactive environments, especially when transitioning between different areas of a graph without losing context. For instance, moving swiftly from the west coast to east coast in our visualization of airline travel above the the U.S can be cumbersome, requiring users to zoom out, pan to the new area, and zoom back in. Additionally, achieving smooth transitions during simultaneous zooming and panning can be technically complex, often resulting in non-monotonic movements that can be disorienting to users. Focus+context techniques offer an alternative solution by allowing users to concentrate on specific details while retaining broader contextual information. These techniques, which complement rather than replace zoom and pan, mitigate the usability obstacles associated with losing context during zooming, particularly crucial in complex data sets where zoom remains essential. Ultimately, a combination of both zoom and pan techniques alongside focus+context methods provides a comprehensive approach to navigating and interacting with complex graph visualizations. [2]

The authors in [3] discuss methods for reducing visual clutter in node-link diagrams used to represent large graphs. While matrix-based representations offer a cleaner layout, they are less intuitive. The focus is on reducing clutter caused by edge congestion. Since node positions often have a clearly defined meaning, e.g. in case of nodes depicting locations in traffic networks, it is not always possible to modify node positions to reduce visual clutter. Techniques like high-resolution rendering, anti-aliasing, alpha blending, and interaction (zooming, EdgeLens, Edge Plucking) are suggested. The concept of edge bundling, inspired by how wires are managed, is introduced. The Force-Directed Edge Bundling (FDEB) technique is proposed, which uses flexible springs to attract edges towards each other. This self-organizing approach results in reduced clutter and smooth edge bundles. Comparison is made with existing methods like Geometry-Based Edge Bundling (GBEB). The paper is structured with sections on visualization techniques, the FDEB technique, examples, comparisons with GBEB, rendering enhancements, conclusions, and future directions.

The work discusses the approach of utilizing straight-line edges in node-link diagrams as the starting point for a method called FDEB (Force-Directed Edge Bundling). Before delving into FDEB, it provides an overview of common techniques for visualizing graphs as node-link diagrams and methods for reducing visual clutter caused by edge congestion.

The authors in [3] broaden the related work mentioned by the authors of the papers [1], [2] or [6].

In the exploration of techniques for reducing edge clutter in graph visualization, it has been noted that various methodologies exist to address this challenge. One avenue of investigation involves visualizing graphs in a clustered manner, wherein edges are drawn between clusters of nodes rather than individ-

ual nodes, thereby mitigating the visual overload caused by numerous individual edges. Additionally, interaction-based techniques such as EdgeLens and Edge Plucking have been explored for dynamically curving edges away from a user's focal point, although these are primarily applicable in interactive visualization scenarios.

Moreover, researchers have considered the efficacy of representing edges using half-lines, a method that shows only the initial segment of the line between nodes, thereby potentially reducing visual clutter. However, challenges exist in accurately identifying the target node in such representations.

Recent advancements in edge clutter reduction techniques, particularly those leveraging edge bundling, have also been explored. For instance, researchers have delved into confluent graph drawing, which merges groups of edges to be drawn together, albeit with limitations regarding the applicability of this method to all graph types. Additionally, approaches like hierarchical edge bundling (HEB), which bends each edge toward a hierarchy, have been investigated. However, these methods may present challenges related to curvature variation in bundled edges.

This exploration underscores the significance of addressing edge clutter in graph visualization and highlights the diverse range of methodologies available to mitigate this issue. In the pursuit of efficient edge bundling within general graphs, conventional methods like Hierarchical Edge Bundling (HEB) face challenges in crafting suitable hierarchies that faithfully represent high-level edge patterns. An alternative approach, Force-Directed Edge Bundling (FDEB), offers a compelling solution. FDEB employs a self-organizing, force-directed method, utilizing linear spring forces between edge segments and electrostatic forces between interacting edges. Through iterative simulations, it achieves edge bundling by updating subdivision point positions based on aggregated forces, facilitating a visually cohesive representation of graph structures.[3]

2.2 Visualisation of airline data

The paper [5] discusses the increasing availability of large datasets detailing vehicle movements like airplanes, which have applications in Air Traffic Control (ATC) and other analyses. It focuses on analyzing airplane movement data sets for ATC scenarios, but visualizing these large data sets poses challenges in computational and visual scalability due to their size and high-density traffic regions. To address this, the paper introduces a visualization system employing techniques like animation and density maps for efficient visualization over large intervals. Computational scalability is ensured by implementing these techniques on the GPU. The system extends previous work by incorporating temporal attributes and is demonstrated on both medium and large-scale datasets. The paper concludes with a discussion on related work, the proposed visualization techniques, and visualization results.

Researchers explore dynamic visualization techniques for airplane flight data. Inspired by prior research, they integrate animation and density maps to depict flight attributes such as position, height, direction, and speed. Departing from

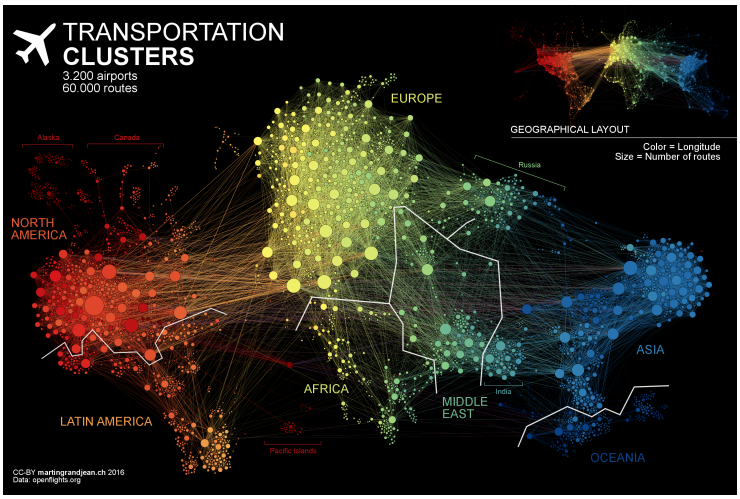

Figure 1: Visualisation of world air travel published in [8]. In the visualisation, it is simple for the user to determine the size of airports (based on the amount of flights), volume of traffic between airports and the region of the world an airport is in.

traditional methods, they overlay animation on density maps to offer a comprehensive view of flight dynamics. Using a sliding time window, they select relevant data points and render trail segments with transparency textures to show airplane positions and movements. Color mapping enhances the visualization by representing altitude and flight direction. To address visual clutter, they apply transfer functions. Overall, their project aims to provide actionable insights for aviation planning and management through innovative data visualization techniques.[5]

The most similar work we were able to find is [8]. The task in this article is comparable to ours but the domain is the whole world, not only the U.S.A as in our case. The visualization by Martin Grandjean showcases over 3,200 air traffic hubs and their relationships beyond geographical proximity. The resulting visualisation can be seen in Figure 1.

3 Implementation

For the app's backend, we used Python with NumPy to handle the computation-intensive tasks of the Force-Directed Edge Bundling (FDEB) algorithm, leveraging NumPy's efficient array operations for swift data processing. For the frontend, we used PySide to create an interactive GUI. PySide's integration with the Qt framework allowed us to design sophisticated visualizations, ensuring a seamless connection with the backend. This combination enabled the development of a robust application with strong computational and visualiza-

tion capabilities.

3.1 GUI

3.1.1 Code Structure

In `main.py`, we implemented the core of the visualization based on the template provided in Moodle. We extended the template to suit our needs while maintaining the provided structure. The code integrates various modules and classes to create an interactive and visually appealing representation of U.S. air travel data.

As stated above, we used PySide6's integration with Qt and our custom implementation of the FDEB algorithm (described later).

The code structure is the following:

- The `VisGraphicsScene` class manages the graphical elements of the visualization, including airport nodes and flight path edges. It handles user interactions, such as selecting airport nodes from the graph and highlighting corresponding edges.
- The `VisGraphicsView` class extends `QGraphicsView` to provide advanced view manipulation capabilities, including zooming and panning. It ensures that user interactions are intuitive.
- The `MainWindow` class is the main application window. It initializes the dataset, sets up the GUI components, and maps the data to visual elements. It includes methods for creating Qt widgets, handling city list selection, and generating and mapping data to the graphical scene.
- The `generateAndMapData` method is crucial for converting the dataset into graphical representations. It includes projecting geographical coordinates using the Mercator projection, scaling node sizes based on airport connectivity, and performing edge bundling to optimize the visualization of flight paths.
- The `main` function initializes the application and launches the main window, providing a cohesive structure for running the visualization.

3.1.2 Airport representation

In our visualization, we varied the size of the airport nodes based on the number of flights, which corresponds to the degree of each node in the graph. To achieve this, we increased the base size of each node by multiplying it by a constant factor for each degree increment. This approach allowed us to represent the relative importance of each airport visually. To ensure clarity and prevent extreme variations, we clamped the node sizes to a predefined minimum and maximum value. This method provides a balanced and intuitive representation of the data, highlighting major hubs while maintaining the visibility of smaller airports.

3.1.3 GUI capabilities

The GUI capabilities of our visualization include several interactive features designed to enhance user experience and data interpretation:

- **Zoom and Pan:** The zoom and pan functionality enables users to navigate the visualization, though its responsiveness is currently hindered by a GUI implementation bottleneck, which is noted for future improvement.
- **Node Selection:** When an airport node is selected, all flight paths (i.e., edges) to and from that airport are highlighted, allowing users to easily identify its connections and significance within the network.
- **Airport Selection by Name:** Users can select an airport by name from a list on the right side of the GUI, which triggers the same highlighting effect as direct node selection. This dual interaction method provides flexibility and improves the overall usability of the visualization.

3.1.4 Future work

The GUI implementation has emerged as a bottleneck due to its current inefficiencies. The responsiveness and performance of the interactive features lag behind the computational capabilities of the backend that can be precomputed and therefore work very fast. A potential solution to this issue is the implementation of multithreading. By distributing the GUI tasks across multiple threads, the responsiveness and efficiency of the interface could be significantly enhanced. This approach would allow the frontend to handle user interactions and visual updates more smoothly without being hampered by the heavy computational tasks processed by the backend. Integrating multithreading into the PySide framework could thus ensure a more fluid and responsive user experience, better aligning the performance of the frontend with the robust capabilities of the backend. Besides multithreading, rendering using a GPU would also surely result in a more responsive user experience.

Moreover, exploring alternative visualization frameworks beyond PyQt holds promise for the creation of a more responsive application. Our PyQt implementation, while usable, does not offer the ideal responsiveness required for our application's demands. Exploring alternative frameworks could unlock the potential for smoother user interactions and improved performance overall.

Furthermore, adding a map of the U.S. beneath the visualization could enhance clarity by providing geographical context for the airport nodes. However, we encountered several challenges that prevented us from successfuly implementing this feature. Initially, we attempted to overlay an SVG image with the nodes, but accurately mapping the airport locations proved to be more complex than anticipated. Subsequently, we experimented with the `cartopy` library to precisely map the airports to their geographical coordinates. While this approach improved location accuracy, it introduced issues with application responsiveness, particularly in handling click events to highlight selected cities

and corresponding edges. Due to these challenges, we prioritized maintaining the interactivity of the visualization over incorporating a background map. Nevertheless, we acknowledge that achieving both a map overlay and interactive features is feasible and could be explored in future work.

3.2 Force directed edge-bundling (FDEB)

In the file `fdeb.py`, we have implemented the Force directed edge-bundling (FDEB) algorithm to reduce visual clutter. Firstly, we compute the edge compatibility measures to address the issue of the bundling amount often being too high. In [3] the authors introduced the following compatibility concepts:

- Angle compatibility:

$$C_a(P,Q) \in [0,1] \text{ as } C_a(P,Q) = |\cos(\alpha)|$$

  where

$$\alpha = \arccos(\frac{P \cdot Q}{|P||Q|})$$

- Scale compatibility:

$$C_s(P,Q) = \frac{2}{l_{avg} \cdot min(|P|,|Q|) + \max(|P|,|Q|)/l_{avg}}$$

  where

$$l_{avg} = \frac{|P| + |Q|}{2}$$

- Position compatibility:

$$C_p(P,Q) = \frac{l_{\text{avg}}}{l_{\text{avg}} + ||P_m - Q_m||}$$

- Visibility compatibility:

$$C_v(P,Q) = \min(V(P,Q), V(Q,P))$$

where

$$V(P,Q) = \max\left(1 - \frac{2||P_m - I_m||}{\|I_0 - I_1\|}, 0\right)$$

where I_m is the midpoint of I_0 and I_1.

We define the total edge compatibility $C_e(P,Q) \in [0,1]$ between two edges P and Q as

$$C_e(P,Q) = C_a(P,Q) \cdot C_s(P,Q) \cdot C_p(P,Q) \cdot C_v(P,Q).$$

The combined force F_{pi} is now redefined as

$$F_{pi} = k_P \cdot (||p_{i-1} - p_i|| + ||p_i - p_{i+1}||) + \sum_{Q \in E} \frac{C_e(P,Q)}{||pi - q_i||}.$$

We use an iterative refinement approach to enhance bundling performance. Starting with initial subdivision points and step size, we perform simulation cycles with a fixed number of iteration steps. After each cycle, we double subdivision points and halve the step size. By setting specific initial values and decreasing iteration steps per cycle, we achieve smooth edges with an adequate number of subdivision points. Implementing a threshold for edge compatibility reduces the number of interactions, significantly improving performance. Post-processing involves smoothing edges with a Gaussian kernel to enhance appearance and ease of following bundles, with the smoothing amount adjustable to preference.

3.2.1 Future work

While our implementation could be optimized through further utilization of NumPy vectorization, enhancing performance, we did not prioritize this optimization. This decision was based on the observation that the edges can be pre-computed and subsequently loaded almost instantly, mitigating the immediate need for extensive optimization efforts in this regard.

Despite not prioritizing optimization through extensive utilization of NumPy vectorization, it is worth noting that such enhancements could still yield significant benefits. Moreover, while our current focus lies on pre-computing edges for near-instant loading, optimizing performance through vectorization remains an interesting avenue for future exploration. Additionally, such improvements could facilitate real-time algorithmic steps visualization, thereby extending the utility and applicability of our work beyond its current scope.

4 Results of the visualization

4.1 Initial state

In the initial state depicted in Figure 2, although the Mercator projection was employed to map the nodes to their respective geographical coordinates, the visualization of the edges (i.e., flights) between these nodes remains difficult to interpret. Consequently, edge bundling is necessary to enhance the clarity of the visualization.

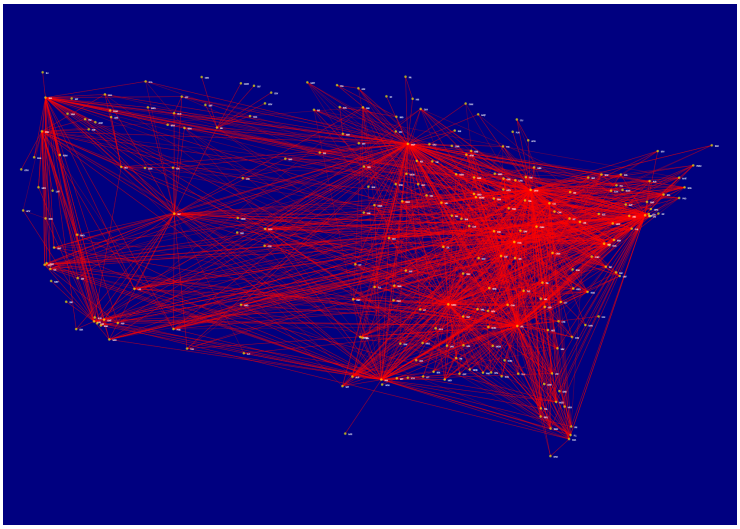

Figure 2: Initial state of the edges with only Mercator projection used to map the nodes.

4.2 Force directed edge bundling and edge opacity

In Figure 3, the implementation of force-directed edge bundling significantly improves the identification of the most frequently used flight paths, thereby enhancing the overall clarity of the visualization. However, determining the most utilized airports remains challenging. Additionally, adjusting the opacity of the edges helps to highlight larger clusters, further improving the overall interpretability.

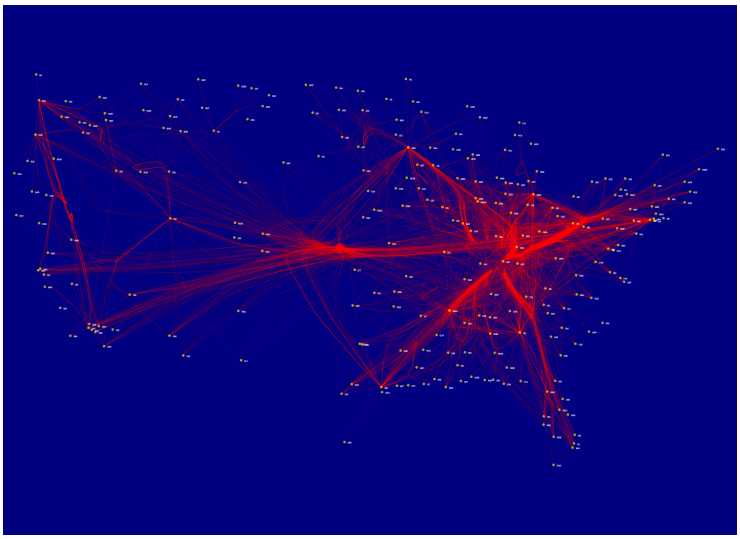

Figure 3: This is a caption for the figure.

4.3 Variable node sizes

Figure 4 illustrates that assigning variable sizes to each airport based on the frequency of flights departing from or arriving at it enhances the visualization, effectively highlighting the airports of greatest significance within the dataset.

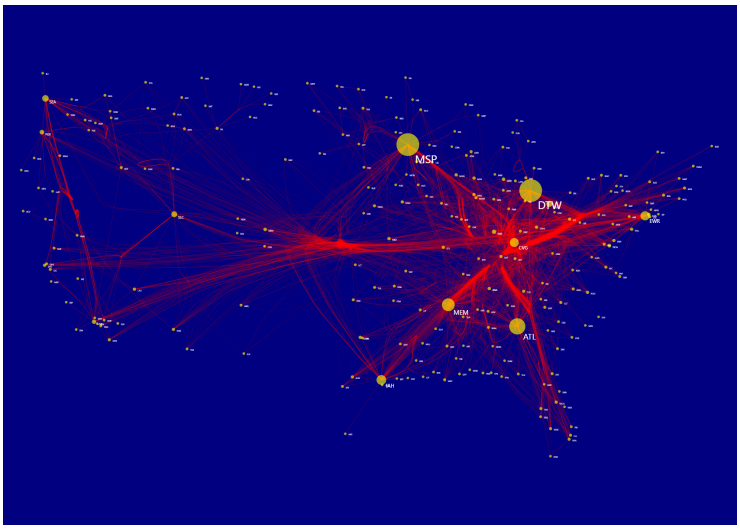

Figure 4: Visualisation with variable node and label sizes.

4.4 Airport node selection

Figures 5 and 6 demonstrate the capability of our visualization tool to enable users to highlight airports (nodes) of interest and discern their associated flights with clarity. Notably, these images showcase the functionality across varying airport sizes, illustrating that the highlighting feature remains effective irrespective of node dimensions. Additionally, users can select airports either by directly clicking on the nodes or by utilizing an alphabetical list provided within the GUI for enhanced accessibility.

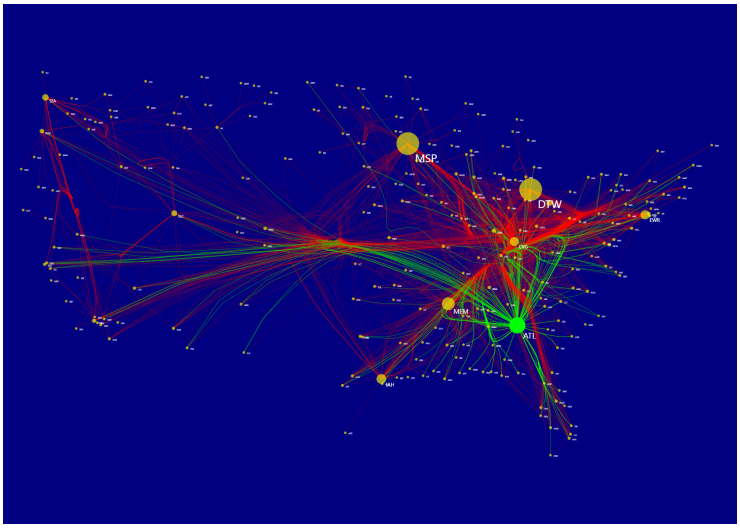

Figure 5: Visualisation of a large node being selected.

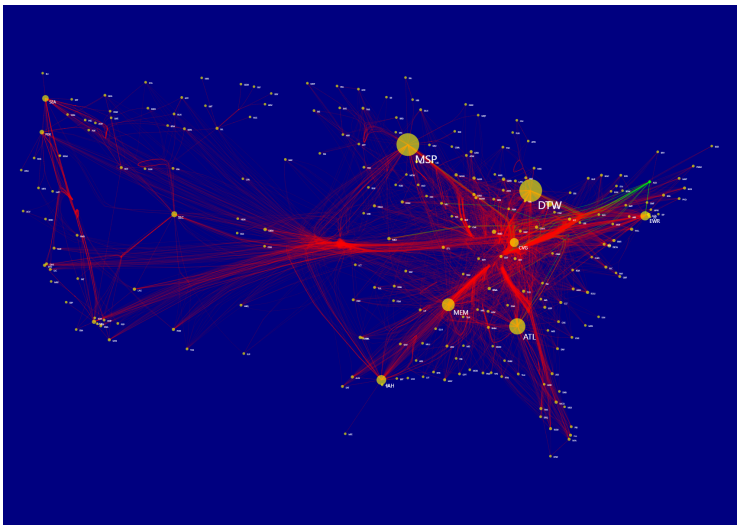

Figure 6: Visualisation of a small node beiing selected.

4.5 Complete app

Finally, Figure 7 presents the full graphical user interface developed for our visualization tool. In addition to the aforementioned visualization and list of airports, it incorporates a convenient button enabling users to effortlessly export the visualization into SVG, enhancing versatility and usability.

Examples of the exports in SVG and high-resolution PNG utilized in this report are available in our Git repository [1] for reference.

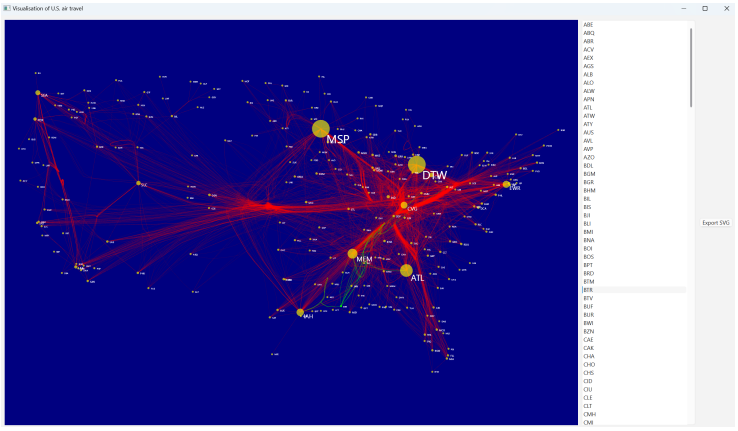

Figure 7: Final GUI of our visualisation application.

[1] Git repository: `https://github.com/cernyad/VIZ-semestral`

5 Conclusion

In this report we examined various approaches to graph visualizations, focusing particularly on the context of airline travel data, and discussed associated challenges. The insights gathered from this review were directly applied in our semester project. The visualization resulting from this STAR analysis aims to enable users to readily assess airport traffic volumes and compare airport sizes and importance based on the amounts of flight lines on the airports.

In the resulting visualisation, FDEB operates effectively, providing clear visualizations that distinctly delineate areas of higher and lower air traffic. Our implementation of FDEB is fully parametrized, offering a pathway towards optimizing bundling quality through the refinement of hyperparameters. Moreover, the potential for vectorization and better rendering presents an exciting opportunity to substantially enhance the efficiency of our implementation.

References

[1] Thomas MJ Fruchterman and Edward M Reingold. "Graph drawing by force-directed placement". In: *Software: Practice and experience* 21.11 (1991), pp. 1129–1164.

[2] Ivan Herman, Guy Melançon, and M Scott Marshall. "Graph visualization and navigation in information visualization: A survey". In: *IEEE Transactions on visualization and computer graphics* 6.1 (2000), pp. 24–43.

[3] Danny Holten and Jarke J Van Wijk. "Force-directed edge bundling for graph visualization". In: 28.3 (2009), pp. 983–990.

[4] Michael Kaufmann and Dorothea Wagner. *Drawing graphs: methods and models*. Springer, 2003.

[5] Tijmen Klein, Matthew van der Zwan, and Alexandru Telea. "Dynamic multiscale visualization of flight data". In: *2014 International Conference on Computer Vision Theory and Applications (VISAPP)*. Vol. 1. 2014, pp. 104–114.

[6] Yehuda Koren, Liran Carmel, and David Harel. "Drawing huge graphs by algebraic multigrid optimization". In: *Multiscale Modeling & Simulation* 1.4 (2003), pp. 645–673.

[7] Roberto Tamassia. *Handbook of graph drawing and visualization*. CRC press, 2013.

[8] Visual Capitalist. *A Network Map of the World's Air Traffic Connections*. `https://www.visualcapitalist.com/air-traffic-network-map/`. Accessed: April 11, 2024. 2018.