**PROJECT PROGRESS DESCRIPTION (35 pts)**

## Project Abstract (2 pts)

|  |
| --- |
| The *Railway Network Optimization System* is a desktop-based application designed to model, analyze, and optimize railway networks using graph algorithms. Users can dynamically create railway networks by entering stations and distances, and then visualize and analyze the network to derive insights such as shortest paths, minimum spanning infrastructure, and key central stations. The primary goal is to provide an intuitive GUI-based system that leverages algorithms like Dijkstra’s, MST, and Centrality measures to improve operational planning and decision-making for railway networks. |

## Updated Project Approach and Architecture (2 pts)

|  |
| --- |
| The system is built using **Python** with a **Tkinter GUI** for user interaction and **NetworkX** for graph processing. Visualization is handled via **Matplotlib**, embedded within the GUI using FigureCanvasTkAgg. The architecture follows an event-driven design where each GUI frame is context-aware, and the central graph (self.G) is persistently stored for algorithmic operations. Key libraries and tools used:   * **Tkinter**: GUI framework * **NetworkX**: Graph construction and algorithms (Dijkstra, MST, Centrality) * **Matplotlib**: Graph visualization * **Pickle/GPickle**: Graph import/export Communication is entirely local, with all operations executed client-side without external servers. |

## 

## Tasks Completed (7 pts)

|  |
| --- |
| Task Completed |
|  **Designed the complete GUI architecture** using   **Implemented dynamic station input** and matrix-based graph construction.   **Integrated Dijkstra’s algorithm** for shortest path computation with route visualization.   **Developed Minimum Spanning Tree (MST) generation** for infrastructure optimization.   **Implemented centrality analysis** (degree and betweenness) to identify critical stations.   **Added import/export features** for railway networks using GPickle format.   **Visualized graphs using Matplotlib**, including edge weights and interactive results. |

## Challenges/Roadblocks (7 pts)

|  |
| --- |
| **1. Dynamic Matrix Input in GUI**: Creating an adaptable interface to handle NxN matrices for distances was complex. We solved it by mapping (i, j) index pairs to individual entry widgets using a dictionary.  **2. Graph Visualization Integration**: Integrating Matplotlib plots into Tkinter without freezing the UI was challenging. Resolved using FigureCanvasTkAgg for thread-safe embedding.  **3. Input Validation**: Incorrect user inputs (non-integers, empty entries) caused crashes. Resolved by thorough validation with try-except and informative message boxes.  **4. MST on Directed Graph**: As MST requires undirected graphs, we convert the directed graph to undirected on-the-fly before applying the algorithm.  **Upcoming Challenge**: Integrating **Ford-Fulkerson (Max Flow)** in a GUI context is complex, as it requires handling capacities and flow constraints. We plan to implement it with guided edge capacity input and visualize flow edges distinctly. |

## Project Outcome/Deliverables (2 pts)

|  |
| --- |
| Key deliverables:   * A functional GUI application for modeling railway networks * Implemented algorithms: Dijkstra, MST, Centrality, and Ford-Fulkerson (pending) * Export/Import capabilities using .gpickle * Complete codebase with documentation * Final presentation and demo video |

# Progress Overview (2 pts)

|  |
| --- |
| Around **85% of the project is completed**. GUI framework, graph algorithms (except max flow), and visualizations are fully functional. Only the Ford-Fulkerson module and final polishing remain. Core features are on schedule; visualization and import/export were completed ahead of schedule. |

# Codebase Information (2 pts)

|  |
| --- |
| Repository Location:  Private Repository -Local System  Repository Name:  DAA  Important Commits:   * + graph-creation-GUI: Matrix input and graph creation   + dijkstra-mst-visuals: Added shortest path and MST visualization   + centrality-module: Centrality measures and plot integration   + import-export-feature: File handling for graphs |

## Testing and Validation Status (2 pts)

|  |  |  |
| --- | --- | --- |
| Test Type | Status (Pass/Fail) | Notes |
| Validated path correctness across various node pairs.  Checked MST structure on connected and disconnected graphs.  Tested degree and betweenness centrality visual scaling.  Verified GPickle file operations for saving/loading graphs.  Max-Flow (Ford-Fulkerson) | Pass  Pass  Pass  Pass  Fail | Outputs match expected shortest distances; UI correctly highlights path.  Warns correctly on disconnected graphs; MST rendered accurately.  Node sizes and colors correspond correctly to centrality values.  No data loss observed during import/export; handles errors gracefully.  Feature not yet implemented; to be validated in next development phase. |

# Deliverables Progress (2 pts)

|  |
| --- |
| **Deliverable** **Status**  GUI Application Completed  Dijkstra’s Algorithm Integration Completed  MST Implementation Completed  Centrality Analysis Completed  Ford-Fulkerson Algorithm In Progress  Graph Import/Export Feature Completed  Final Demo & Documentation Pending |