Boeing Aircraft Route Planner

By Brayden Uber, Austin Harkins, Timothy Stokes, Melesio Albavera, Kaleb Kruse

Project Foundation

Project Goals:

- Develop several different route optimization algorithms. At least one involving more than one aircraft at once. (Create multiple competing heuristics. Analyze which provides better routes and why.)
- Implement cost calculation algorithm (time, fuel, personnel)
- Create input/output processing system (Reads in CSV file data and provides output to visualization mechanism)
- Create a mechanism to visualize the problem and solution(s). (Renders aircraft routes to screen in a user friendly format)

Project Boundaries:

- Limited to static location data
- Fixed cost factors only
- Single route calculation at a time

Required Resources:

- Development environment
- Algorithm testing framework
- Location and cost data sets (how input files are going to be formatted)
- Computing resources for calculations
- Front-end environment for visualization

Specify computing needs

- Need a computer capable of running Rust, Python, and Java code (more computing needs might become apparent in the future).

Users and Stakeholders

User Analysis

- Boeing and their clientele
- We are finding more optimal routes for their aircrafts to save them time and resources.
- They are to be assumed to be able to use simple websites and applications. However, it is assumed that our technology has to stay relatively simple and easy to use. They are expected to give the application information pertaining to a list of nodes, costs, and arcs in a certain format.
- Their key needs are an application that can generate a map of locations, have options for customizing available aircrafts, and an algorithm that finds aircraft routes and displays the information to them. A possible pain point is the speed that the application finds routes and how close the route is to the most optimal route.

Stakeholder Analysis

- Charles Tullock (boeing official)

Stakeholder Expectations and Requirements

A common problem in the aviation industry is planning the route an aircraft takes to visit many locations. Each location can be visited at any time, but the costs and benefits of moving between any two locations varies. This project will come up with heuristic solutions to create routes between locations while maximizing benefits and minimizing costs. Developers will be given input files that contain location identifiers and multiple types of costs (time, fuel, personnel) between locations. Students will create an application that consumes these files and produces an output containing the sequence of locations for the route and corresponding total computed costs.

Developers are expected to create multiple competing heuristics. Analyze which provides better routes and why. The input graph will not always be fully connected. Support planning a route for more than one vehicle simultaneously. Any vehicle could go to any location with an available arc but all locations must be visited just once. Also make sure to have a UI to visualize the problem and solution.

Functional Requirements

Feature ID: F001

Name: Route Optimization Algorithms

Description: This feature will involve several different competing heuristics to a variation of the traveling salesman problem. The heuristics will have their own unique strengths for example one heuristic might work well with multiple airships while a different heuristic works well for one airship. The algorithms will need to be able to find routes that allow the airship(s) to visit every given location at least once.

Priority: High

User Stories:

1. As a user I want to be able to find a low cost flight plan for my aircrafts so that I can save money that can be budgeted elsewhere.

Acceptance Criteria:

1. When locations, costs, and amount of aircrafts are given, the Route Optimization Algorithm shall find an optimized route.
2. The system shall have multiple different Route Optimization Algorithms, with at least one being able to handle multiple aircrafts.
3. At least one heuristic for multiple aircrafts is made, one heuristic for a sparse graph (every location can't visit every other location), and one heuristic that finds the optimal route for a small number of nodes.

Feature ID: F002

Name: Cost Calculation Algorithm

Description: This feature when given time, fuel, and personnel of a path will combine the individual weights into a weight for the total cost.

Priority: Low

User Stories:

1. As a user I want time, fuel, and personnel to be taken into account when calculating a route's cost so that I have an accurate total cost that I am going to have to pay for the overall flight plan.

Acceptance Criteria:

1. When time, fuel, and personnel are given, the Cost Calculation Algorithm will calculate the total costs from each location to every other location.
2. When time, fuel, and personnel are given and you can't travel from location A to location B, a null (or infinite cost) is returned instead.

Feature ID: F003

Name: Input/output Processing System

Description: The input/output processing system will read in CSV files and organize the data into a valid data structure in Rust for backend calculations. It will also write to a PUML file and pipeline the results of backend calculations to the Javascript front end.

Priority: High

User Stories:

1. As a user I want to be able to give the application a file so that it can make a route specific to my needs.

Acceptance Criteria:

1. When a CSV file of locations, costs, and amount of airships are given, the input/output system shall read the input into a suitable data structure for the route optimization algorithm.
2. When a route is found, the output processing system will write the route data into a PUML file.
3. When a route is found, the output processing system will pipeline the data via wasm-pack to the Javascript front-end.

Feature ID: F004

Name: User Interface System

Description: After the back-end completes calculations of the optimal path, that information will be given to the front-end user interface to be displayed in a manner that most efficiently conveys the optimal path. This user interface system will be prioritizing ease of use and ease of understanding for the user so that anyone could easily describe the information being shown to them without difficulty. A system of nodes and paths will be used to visualize the different potential routes an aircraft could take, where the optimal path would be highlighted and bolded to make it more visible and identifiable to the user.

User Stories:

1. As a user I want to be able to visualize plan routes so that I can understand where my aircrafts will be flying and to verify that the application is creating complete routes.

Priority: Medium

Acceptance Criteria:

1. When a route is given in the form of a csv (possibly puml) file, a UI will read the file and show a colored path of the nodes and their arcs on the map.
2. When a world is given in the form of a csv (possibly puml) file, a UI will read the file and show all the nodes and their connections.
3. The UI system shall allow you to select how many aircrafts are available for planning your route.
4. The UI system shall allow you to select which route optimization algorithm you want to use (if the algorithm is available for given requirements such as more then one aircraft).
5. When the user has a CSV file the UI system will have an option to take the CSV file from the user and give it to the Input/output Processing System.

Feature ID: F005

Name: Route Algorithm Analyzer

Description: This algorithm will compare the route optimization algorithms routes which includes the time it takes to find a route, the cost of the route, and possibly how flexible the algorithm is (can it solve multiple and single aircraft routes, full graphs, sparse graphs, et.).

Acceptance Criteria:

1. Where the system has a Route Optimization Algorithm , the system shall have a timer to keep track of how long it took to find a route. (Possible requirement?)

2. When given multiple routes from different Route Optimization Algorithms, the Route Algorithm Analyzer shall compare the costs between routes and output which route is cheaper.

Technical Requirements

Specify how your system will be built:

Development Specifications

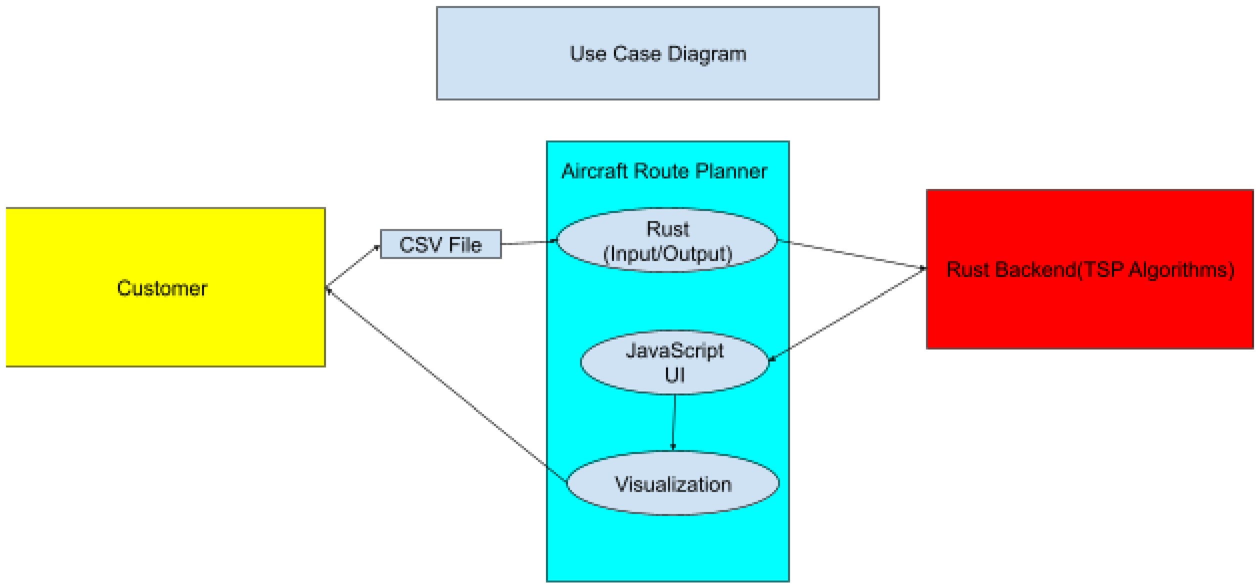
1. The back-end system that handles input/output, the algorithmic data, and unit tests of the former will be built in Rust utilizing the graph and graph_builder crates. The front-end system that handles visualization will be built in Javascript, HTML, and CSS. These systems will be connected via the wasm-pack crate.
2. May incorporate reading the initial CSV file into SQL if many data operations are needed.
3. No API integrations needed.
4. Development tools and environments. (To be determined.) The application will be built on Linux. Github will be used for version control. Netlify will be used to host builds of the user interface. The backend will be written in Rust and will link to Javascript via wasm-pack. The frontend will be written in Javascript, HTML, and CSS.

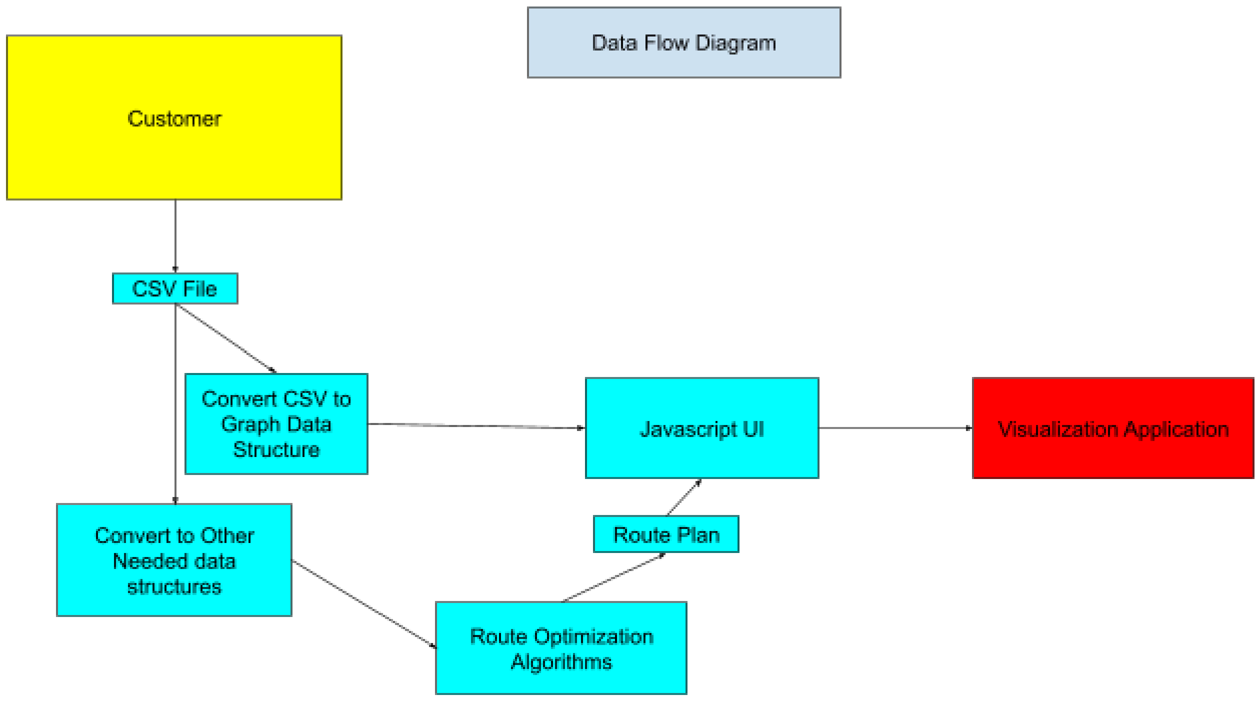
Quality Requirements

1. The performance of the system is expected to take at most 20 seconds to find a solution for graphs with less than 30 nodes. The input system is expected to read the CSV file and produce the graph in <1 second for files <1000 lines long.
2. No security requirements given time constraints.
3. If given a valid map of locations, a valid flight plan will always be given to the user.
4. Compatible with all computers capable of running a website.

Documentation and Diagrams

Required Diagrams




Data Flow Diagram
Customer
CSV File
Convert CSV to Graph Data Structure
Javascript UI
Visualization Application
Convert to Other Needed data structures
Route Plan
Route Optimization Algorithms

Project Planning Timeline

Organize your development timeline around the planned sprints

Sprint 1:

- Implement functional Input/Output processing

- Implement one functional simple algorithm
- Construct barebones user interface.

Sprint 2:

- Implement/revise algorithms taking into account edge weights
- Implement ability to select algorithm in UI
- Implement highlighted travel path

Sprint 3:

- Implement the ability to add new nodes/edges in the UI
- Implement/revise algorithm heuristics
- Implement ability to detect and traverse partial graphs
- Add testing to ensure created paths are valid and complete

Sprint 4:

- Add Route Algorithm Analyzer
- Implement ability to handle multiple disjoint graphs
- If other features are met work on implementing a second concurrent airship

Sprint 5:

- More User Interface fine tuning
- Finish up Route Algorithm Analyzer
- Finish up unit tests

Sprint 6:

- Polish User Interface
- Tune Algorithm heuristics

Validation Strategy

Review Process

- We are going to self review our documentation.
- At least one peer review is needed to confirm a commit on the project's repository.
- We are going to check in with our professor weekly.

- When given feedback we will have a team meeting about how to perform final adjustments.