### Air Safe: Traffic Coordination and Delay Minimization

A PROJECT REPORT

***Submitted by***

BL.EN.U4CSE22252           Swagata Malik

BL.EN.U4CSE22278       Ruchi Chaurasiya

BL.EN.U4CSE22286          Shakshi Yadav

***in partial fulfillment for the award of the degree of***

**BACHELOR OF TECHNOLOGY**

IN

COMPUTER SCIENCE AND ENGINEERING



AMRITA SCHOOL OF ENGINEERING, BANGALORE

 AMRITA VISHWA VIDYAPEETHAM

**BANGALORE 560 035**

June-2020

**ABSTRACT**

This project tackles the ongoing challenges of air traffic delays, which have significant effects on aviation efficiency, customer satisfaction, and operational costs. It proposes an integrated system that combines several advanced techniques, including optimization algorithms, pathfinding methods, real-time data analysis, and predictive modeling, to optimize air traffic management. Optimization algorithms like Dynamic Programming (DP) and Genetic Algorithms (GA) are employed to improve resource allocation, scheduling, and minimize delays in flight operations. Pathfinding algorithms such as Dijkstra and A\* are used to identify the most efficient routes for flights, considering factors like congestion and weather conditions. The use of real-time data ensures the system can adapt dynamically to changing circumstances, enhancing decision-making in real-time. Additionally, a predictive model is developed to forecast potential delays based on historical data, helping airlines proactively manage disruptions. By combining these approaches, the system aims to optimize flight schedules, reduce delays, and enhance the overall efficiency of air traffic management, improving the passenger experience and reducing costs for airlines and airports.

**TABLE OF CONTENTS**

**Page no**

ACKNOWLEDGEMENT i

ABSTRACT ii

LIST OF FIGURES iii

CHAPTER 1- INTRODUCTION 01

* 1. INTRODUCTION TO FLIGHT dELAY 1-2
  2. MOTIVATION 2
  3. PROBLEM STATEMENT 3

CHAPTER 2 – LITERATURE SURVEY 3-6

CHAPTER 3 – REQUIREMENTS AND ANALYSIS 6

* 1. SOFTWARE REQUIREMENTS 6-7

3.2 HARDWARE REQUIREMENTS 7-7

CHAPTER 4 – DESIGN 8-11

CHAPTER 5 – IMPLEMENTATION 11

* 1. DATA PREPROCESSING 11-13
  2. DATA VISUALIZATION 13-16
  3. OPTIMIZATION TECHNIQUES 16-20
  4. SHORTEST PATH FINDINGS 20-25
  5. REAL- TIME ANALYSIS 25
  6. DELAY PREDICTION 25-26

CHAPTER 6 – RESULTS 26-30

CHAPTER 7 – CONCLUSION AND FUTURE ENHANCEMENT 31-32

REFERENCES 33-34

**LIST OF FIGURES**

* Fig 4.1 – Air Traffic Analysis and Optimization
* Fig 5.2.a – Data Distribution of Carrier delay
* Fig 5.2.b – Data Distribution of weather delay
* Fig 5.2.c – Data Distribution of NAS delay
* Fig 5.2.d – Correlation heatmap of numerical features
* Fig 5.5.a – Analysis of Actual v/s predicted delay
* Fig 6.2.1.a – Execution Time comparison between algorithms
* Fig 6.2.2.a – Execution Time comparison between pathfinding algorithms
* Fig 6.3.a – Real time analysis of delays

**CHAPTER 1 - INTRODUCTION**

* 1. **Introduction to Flight Delay optimization**

The efficiency in managing air traffic becomes necessary for safety and stress-free flying with present realities of underground airspace congestion. Delayed air traffic translates into a loss of revenues to airlines, disgruntled customers, and inefficient operations. The dominant systems in this domain have been relying on mostly traditional methods like genetic algorithms and Prim's algorithm for optimization problems. Although they constitute effective solutions, such methods are not exhaustive enough to create an encompassing system that combines multiple optimization methods with real-time analysis to tackle the even more dynamic problems of air traffic management.

To overcome this, we intend to fill the gaps by combining several optimization algorithms with shortest path algorithms and online delay analysis. The objective of the project is to bring all this in a singular framework that combines optimization techniques, shortest path algorithms, and analysis into a larger perspective of making air traffic management more efficient.

To ensure the reliability of the proposed work many optimization techniques were used on air traffic data that include greedy algorithms for quick heuristic-based solutions, linear programming for mathematically optimized results, genetic algorithms inspired by evolutionary principles, graph-based methods for structural optimization, dynamic programming to deal with sequential decision making, Prim's algorithm for minimum spanning tree construction, and simulated annealing for global optimal solution exploration. Analysis and comparison of each of these techniques will account for their effectiveness in optimizing air traffic operation.

Shortest path algorithms were used along with optimization algorithms to study a comparative savings which these costs might have been expected under some defined circumstances between two nodes in the air traffic network. Once again on these, two algorithms, Bellman-Ford and Dijkstra, were used for having already been proven to yield optimal results for different conditions in finding paths. Integrating these algorithms ensured the capability of identifying routes with minimal delays and better overall traffic flow.

* 1. **Motivation**

The increasing demands for travel by air have put tremendous pressure on existing air traffic management systems, highlighting the inefficiency of temporary and routing optimization in managing air traffic delays. Airspace congestion and the uncertainty of delays caused by the weather, technical problems, or limit on air traffic movement make the challenge of management much more difficult.

A notable limitation of these systems is the non-formation of an integrated framework for many optimization algorithms and real-time evaluation of delays. The disconnect makes the systems low scalable, low flexible, and low responsive to real-world dynamics. Furthermore, the lack of stringent validation procedures that have predicted delays compared with some actual delays further weakens confidence behind the solutions, which has increasingly limited their practical adoption in dynamic environments.

It is against this backdrop that the present research has been prompted to fill this space and also to make a complete system that will address route optimization's complexity along with dependability through effective validation. Diversity in optimization approaches paired with very predictive abilities is what this study envisions as bringing a solution that could scale and adapt to the new challenges of modern air traffic.

* 1. **Problem Statement**

Air-traffic routing and delay management could best be described as an arduous and dynamic challenge in association with the character of the flight networks as well as the growing demands of air travel. Most existing systems tend to rely significantly on isolated optimization methods, which include but are not limited to genetic algorithms and Prim's algorithm, to tackle specific aspects of the issue. However, such methods provide limited scope as well as inflexibility for incorporation with real-time delay prediction and validation processes.

Some of the important challenges include the following:

* Isolated Optimization Techniques
* Lack of Real-Time Predictive Analysis
* Insufficient Validation Mechanisms
* Limited Scalability

**CHAPTER 2 – LITERATURE REVIEW**

T. Zhaoo et. al [1] proposed the necessity for a multi-objective optimization model that can address the simultaneous optimization of departure slots, routes, and ATC workload. Also, Genetic algorithms have been previously utilized in various optimization scenarios, including flight scheduling and route assignment, but this paper aims to enhance their application by integrating multiple objectives into a single framework.

Kartik Gopalkrishna et. Al [2] proposed the stability results of MJLS (Markovian jump linear system) models and optimal control strategies. This research focus on the role of optimization and control in air traffic management. Practical algorithms are imperative for safe and efficient air traffic operations. A network is defined by nodes and edges; an adjacency matrix can be used to represent it. The measures of centrality include degree centrality, as well as eigenvector centrality. Delays and cancellations are better modeled as weighted, directed networks.

A. sikoraa et. al [3] proposed that the design of an unmanned VTOL aircraft designed to be used for the 2018 edition of the Medical Express UAV Challenge (ME) competition. This paper also explained about the system that retrieves coordinates of the permissible flight area from a microSD card and compares those with the system's current location to carry out an autonomous mission involving a flight from the starting point to an injured person, locating the landing point through vision algorithms.

Y. Yaguchi [4] proposed the applied blockchain technology on reservation systems with the aim of building a distributed autonomous UTM (UAS traffic management) with two major management functions: flight plan and dynamic management. They evaluated their targeted system by load test based on future estimates and also used an efficient data structure based on octree and this approach can handle the estimated future flight plan.

D.song et .al [5] proposed the Genetic Algorithms (GAs) used to generate an optimal flight path given the target positions. A new FMS is used with it that not only generates the optimum trajectory, but also controls action for path tracking to be smooth with the help of memory-based control strategy that can dynamically adjusted according to the varying flight condition. They have also used simulation for the verification of their proposed system.

Rahul Rampure et .al [6] proposed various algorithms to simulate evolutionary processes to find optimum solutions. The GA generate an initial random population of chromosomes representing flight paths. The algorithm iterates toward convergence, returning the best solution for aerial path and ground delay. The used model in this paper produces around 24,000 flights that yield optimal paths and delays. Average sector density reduced by 18%, occupancy was reduced by 9% while keeping flight times and delays as the model decreases congestion.

Deepu dev S. et. Al [7] proposed the significance of GDPs (Ground delay programs) in air traffic flow management culminates with addressing demand-capacity mismatches at airports. These programs impose departure delays on inbound flights to enhance the operating efficiency and safety in air traffic operations. The main focus of the research is on integrating data analytics and machine learning to understand the detailed dynamics of GDPs. This research includes predicting the probabilities of flight delays and optimal parameter adjustments for key effectiveness.

Mehran Makhtoumi [8] proposed a directed acyclic graph with nodes and arcs implemented to present a flight delay projection model that projects flight delays, whereas Bayesian networks have been utilized to analyze and visualize the delays among airports. This paper presents scheduling methods optimizing strategic slot assignment for both arrival and departure during a specified period at an airport.

Coline Ramee et. al [9] proposed the used of A\* algorithm to find the shortest path through the network with the objective to minimize cruise fuel burn which saves 8% of fuel burn. This paper has used fuse data and compared to actual flight to estimate the benefits. They also explained that optimization algorithm is fast enough that it could be run multiple times during the flight to account for updated weather information, to help pilots request updated paths from Air Traffic Control as large airlines can rely on dispatchers, flight specialists, and meteorologists to optimize flights.

Krishan Kumar et .al [10] proposed two faces: find automatically solutions to the conflicts, and find the optimal solution regarding conflicts. This is a comparison with the main idea behind the solution of the conflict detection and being as close as possible to the current ATC system. They also proved that the algorithm cannot only optimize the departing time and route of every flight under reasonable time-horizon   
but also reduce the workload of ATC with the help of the simulation results based on the operational flight data.

R. A. Altava et. al [11] proposed that a Dynamic Programming and Branch & Bound algorithms have been applied to the specific problem of trajectory generation in aviation. Heuristics such as A\* and Beam search accelerate route optimization at the cost of limiting flexibility and optimality. Sadeque Hamdan et. al [12] proposed a model that considers rerouting of flights and continuing flights by presenting a bi-objective integer linear programming mathematical model that minimizes the total delay cost and the total CO2 emissions. This paper also used weighted comprehensive criterion method: a scalarization technique that minimizes the total weighted deviation from each objective's ideal solution.

Abdelghani Fadil et .al [13] proposed the congested airspace and minimize the cost of flight delays considering uncertainty in sector capacity by utilizing scenario approach optimization method approximating the optimal solution and a mixed-integer programming model is developed based on the deterministic sector capacity obtained. And also another uncertain element in air traffic flow management is the flight demand, which can also be incorporated in future work.

Qing Cai et. al [14] proposed that a bi-objective optimization model with the objective of minimizing the TVR(technical vertical risk) along with the delay due to aircraft speed variability while applying evolutionary computation techniques.  
They also focused on few such assumptions like flight level altitude and constant speed have been made to implement the solution.

Abdelhamid Boujarif et. al [15] aims at including the optimization of airport capacity in the ATFM (air traffic flow management) where mathematical formulation considers the departure and arrival capacities as decision variables to be optimized. Also, minimize rerouted flights  
as rerouting happens because to gain more time for take-off or to make a landing operation.

Wen Tian et. al [16] explained that focus on a multi-objective and a multi-constrained air traffic flow management model, utilize a multi-objective genetic algorithm thus, realizing the balance of the air traffic flow in airspace, and reduce airspace congestion effectively. Minimize the difference between the maximum number of flights and the average of the counts of flights above each sector capacity.

Wen-Bo Dua et. al [17] proposed to apply the Granger causality test to construct a delay causality network (DCN) for air traffic. This paper also describes the causality relationship, which states that there is a causal relationship if the delays seen at one airport can account for the delays that arise at another one after a few hours.

Qiang Li et .al [18] focused on building a delay propagation network based on Bayesian Network approach to study the complex phenomenon of delay propagation within a large network consisting of the 100 busiest airports in the United States. This paper also indicates that the relationship of flight delays among airports is mainly caused by connected flights.

Dhawal Thakkar et. al [19] research includes flight scheduling that specifies the legs of the flights and corresponding arrival and departure times with the aim to find strings that would give us the least propagated delays while ensuring that the customer feedback-based importance assignments to flights are taken into consideration. Zhiwei Xing et. al [20] uses genetic algorithm to solve and also to minimize the number of rerouted flights. This research also provide a reference solution for the real-time adjustment of flight queue, so as to assist the controller to schedule efficiently

**CHAPTER 3 – REQUIREMENTS**

The air traffic optimization and delay prediction system involves both software and hardware. On the software side, core functionalities will be implemented using Java and Python programming languages along with libraries for optimization, machine learning, and graph algorithms. The coming addition is an Excel file that will contain information on air traffic. Hardware-wise, at least a multi-core processor, 8 GB of RAM, and sufficient 500 GB SSD storage will provide the capacities to carry out complex computations.

* 1. Software Requirements

1. Programming Languages and Dataset:

* Python: To obtain the practical realization of the proposed work.
* Excel file containing dataset: Used historic data for analysis.

1. Frameworks and Libraries:

* Tkinter: For constructing the graphical user interface, or GUI, of the application
* Pandas: For data manipulation and analysis specifically for reading and pre-processing the dataset
* Scipy: For use in optimization algorithms such as linear programming
* NetworkX: For use in graph-based algorithms such as Dijkstra's and Prim's algorithms
* Scikit-learn: For machine learning models, mainly Random forest, in predictive models
* Seaborn: For data visualization, which is not necessary but applies to more advanced visualization
* Matplotlib: For basic plotting and charting purposes
* Random: For generating random solutions, used in genetic algorithms and simulated annealing one.

1. Development Environment:

* Jupyter Notebook: For interactive testing, debugging, and visualizing machine learning models. elaborate
  1. Hardware Requirements

1. Processing Unit A Multi-Core Processor (Quad Core or higher): Being able to perform very significant data-intensive operation, like optimization algorithms, route calculations, and delay prediction, especially when such operations are performed on a very large dataset.
2. Memory (RAM):

* 8GB-RAM: Fairly enough for everyday optimizations and normal large datasets.
* 16GB-RAM and above: Perfectly suitable for bulky datasets while applying complex algorithms or distributed computing.

**CHAPTER 4 – DESIGN**

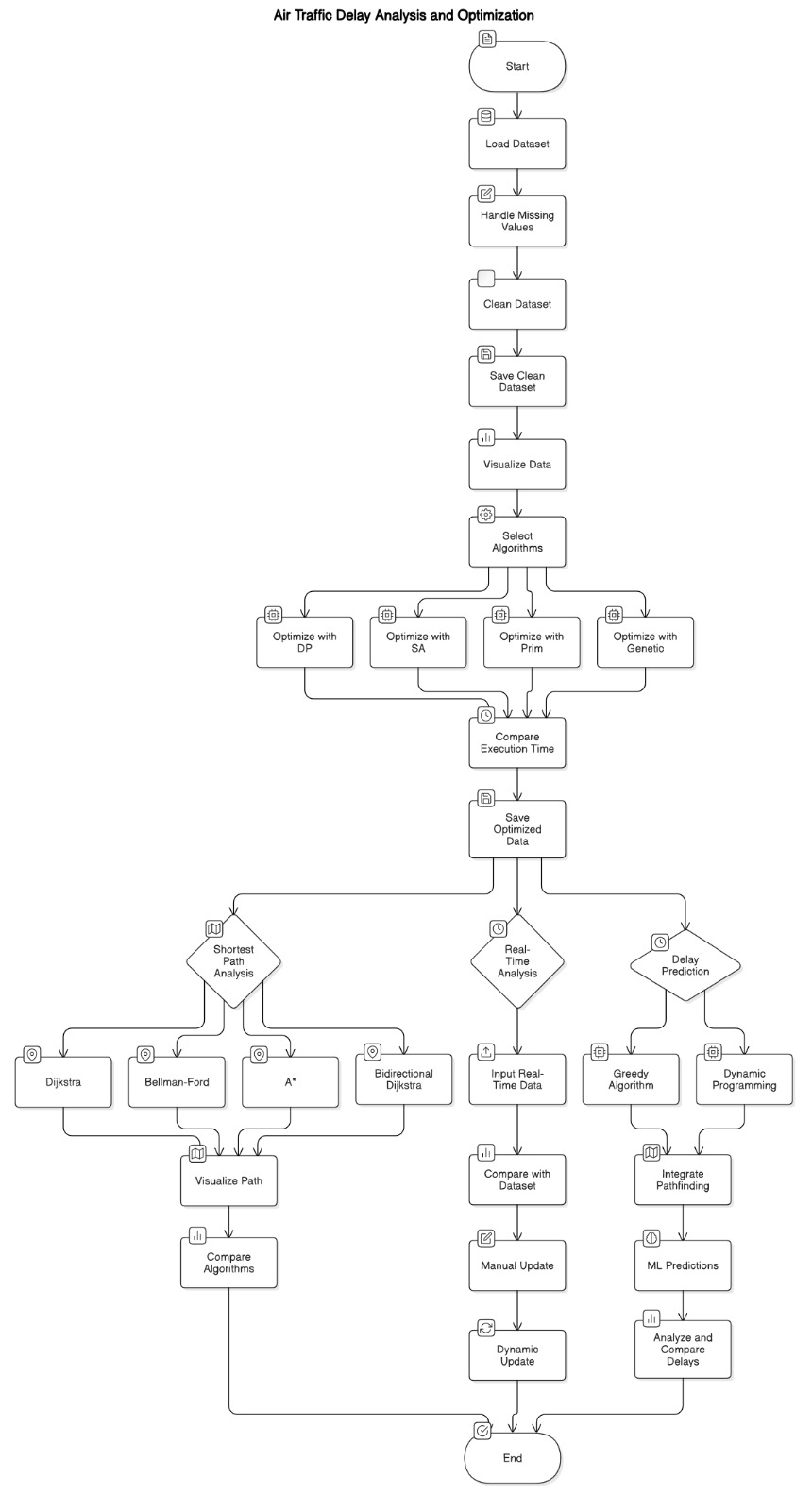


Fig 4.1: Air Traffic Analysis and Optimization

The image represents a flowchart outlining the process for Air Traffic Delay Analysis and Optimization. Below is a step-by-step explanation of the flow for each block:

**Start:**

* The process begins by loading the air traffic dataset, which contains information on delays, routes, and other relevant data.

**Data Preparation:**

* Handle Missing Values: Missing data points in the dataset are identified and handled. This could involve filling in missing values or removing rows with missing data.
* Clean Dataset: The dataset is cleaned to ensure it is free from errors or irrelevant information.
* Save Clean Dataset: The cleaned dataset is saved for further analysis.

**Visualization:**

* Visualize Data: The cleaned dataset is visualized, allowing insights into the air traffic data, such as delay patterns and trends.

**Algorithm Selection:**

* A selection of optimization algorithms is chosen for processing, including:
  + Dynamic Programming (DP)
  + Simulated Annealing (SA)
  + Prim's Algorithm (for minimum spanning tree optimization)
  + Genetic Algorithm (for evolutionary optimization)

**Optimization Process:**

* The system applies each selected optimization algorithm on the dataset, resulting in optimized solutions for air traffic management.
* Compare Execution Time: The execution times of each algorithm are compared to evaluate their efficiency.
* Save Optimized Data: The optimized data is saved for use in subsequent steps.

**Shortest Path Analysis:**

* The shortest path analysis is performed using different algorithms to identify the optimal routes in the air traffic network:
  + Dijkstra Algorithm
  + Bellman-Ford Algorithm
  + *A*\* Algorithm
  + Bidirectional Dijkstra Algorithm
* These algorithms help identify the most efficient paths between source and destination nodes in the air traffic network.

**Real-Time Analysis:**

* Input Real-Time Data: Real-time data, such as current delays, flight statuses, and weather conditions, are fed into the system.
* Compare with Dataset: The real-time data is compared with the existing dataset to assess the impact of new information on current routes and delays.
* Manual Update: If necessary, the system allows manual updates to the dataset based on real-time inputs.

**Delay Prediction and Comparison:**

* Delay Prediction: The system predicts delays using techniques like:
  + Greedy Algorithm
  + Dynamic Programming
* ML Predictions: Machine learning models are also used to predict delays based on historical data and real-time inputs.

**Final Integration and Update:**

* Integrate Pathfinding: Pathfinding algorithms are integrated with delay predictions to provide optimal routing solutions considering delays.
* Dynamic Update: The system dynamically updates the air traffic flow and optimization strategies as new data is processed.

**Analysis and Comparison:**

* Analyze and Compare Delays: The system analyzes the delay predictions and compares the effectiveness of the algorithms used to manage and minimize delays.

**End:**

* The process concludes once the delay analysis and optimization steps have been completed.

**CHAPTER 5 – IMPLEMENTATION**

Let's expand each point with brief, detailed explanations to help you build out your 10-page implementation section.

**5.1) Data Pre-processing**

**Dataset Overview:**

The dataset, air-traffic.csv, contains extensive flight data, including factors like:

* **arr\_flights:** Number of arriving flights.
* **carrier\_delay:** Delays due to airline issues.
* **weather\_delay:** Delays caused by weather conditions.

Understanding these variables is crucial because each directly impacts delay predictions and resource optimization.

**Data Cleaning:**

* **Handling Missing Values:**  
  Initially, missing values were identified using df.isnull().sum(). For categorical and numerical columns, different strategies were applied:
  + **Numerical Data:** Replaced missing values with the median to avoid skewing the data.
  + **Categorical Data:** Missing values in flight carriers were filled with the mode.
* **Outlier Detection:**  
  Outliers, particularly in carrier\_delay and weather\_delay, were detected using boxplots. For instance, any delay beyond the 95th percentile was flagged as an outlier. These were either removed or capped to improve model accuracy.

**Feature Engineering:**

* **New Features:**  
  Derived additional insights by creating new variables like total\_delay (sum of all delay types) and delay\_ratio (delay per flight).
* **Normalization:**  
  Delays were normalized to ensure all features had comparable scales, crucial for optimization algorithms and machine learning.

**5.2) Data Visualization**

**Exploratory Data Analysis (EDA):**

* **Histograms:**  
  Plotted for columns like carrier\_delay and weather\_delay to understand their distribution.

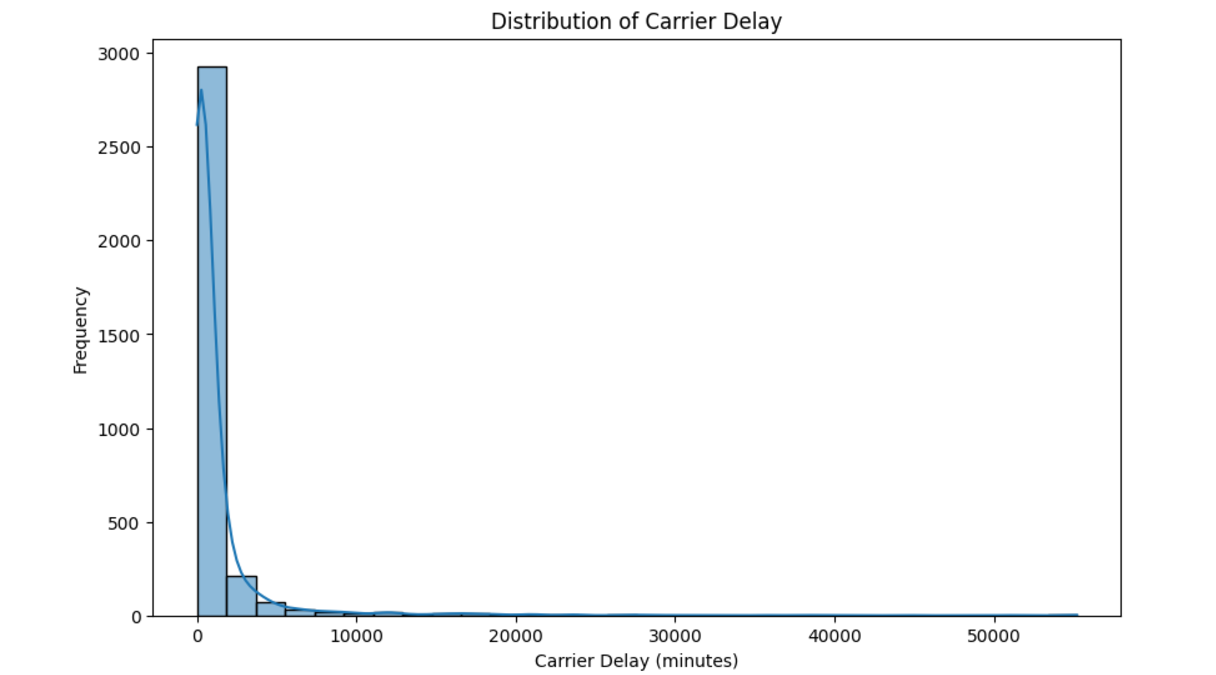


Fig 5.2.a: Data distribution For carrier delay

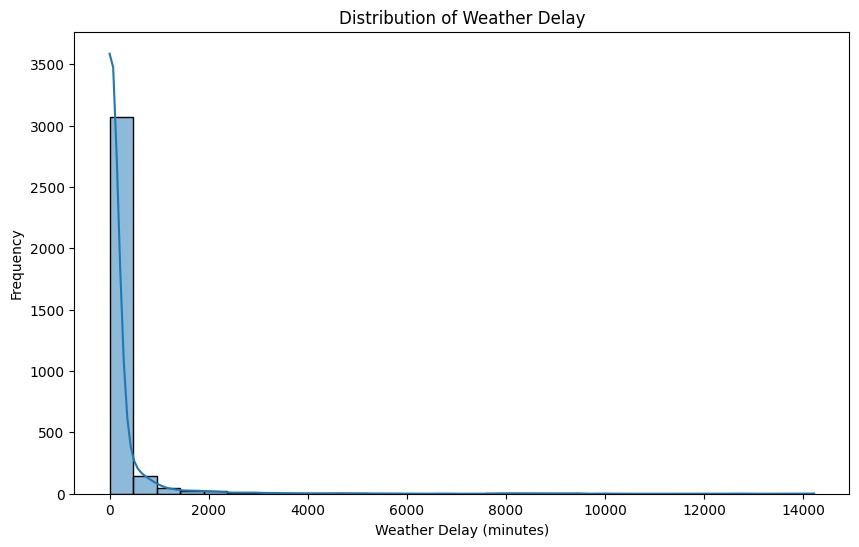


Fig 5.2.b: Data distribution For weather delay

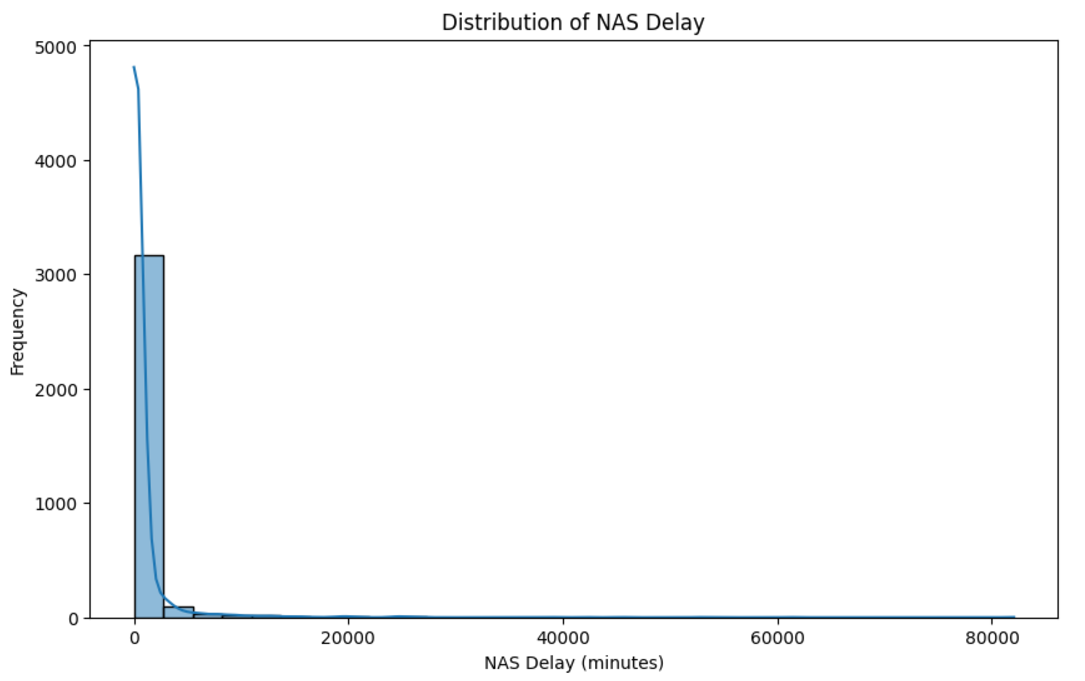


Fig 5.2.b: Data Distribution for NAS delay

* + **Observation of the distribution:** Carrier delays showed a right-skewed distribution, indicating most delays are short but some extreme outliers exist.

**Correlation Analysis:**

* A heatmap was generated to visualize correlations between variables:
  + **Insight:** arr\_del15 (arrival delays exceeding 15 minutes) showed strong correlations with weather\_delay and carrier\_delay.
* **Boxplots and Heatmap:**

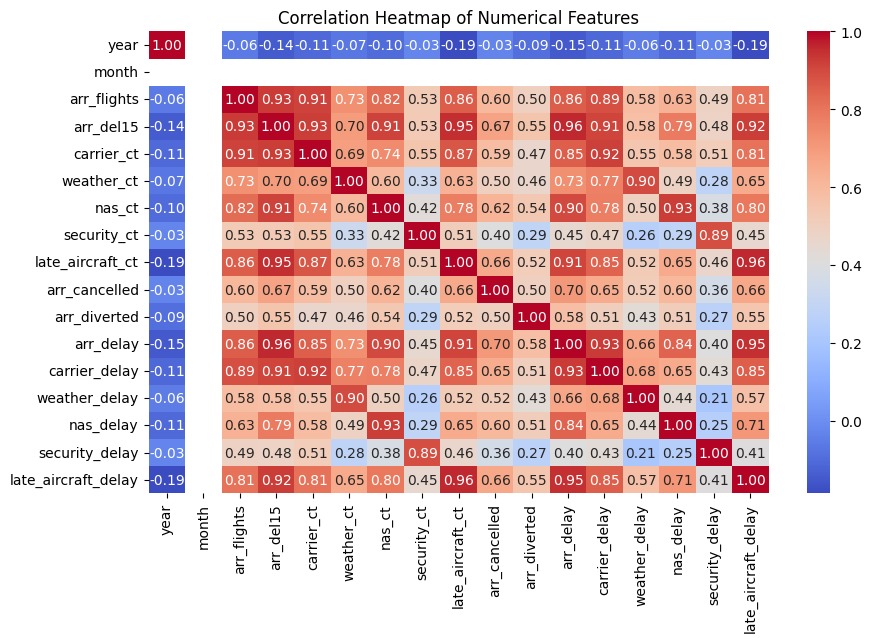


Fig 5.2.d: Correlation heatmap of numerical features

**Correlation Heatmap Analysis for Flight Delay Optimization**

**Key Insights:**

1. **Strong Positive Correlations:**
   * **arr\_del15 vs. arr\_delay (0.96):**  
     Flights with delays over 15 minutes strongly correlate with overall arrival delays, making arr\_del15 a crucial predictive feature.
   * **carrier\_delay vs. arr\_delay (0.93):**  
     Carrier-related delays significantly impact overall delays. Addressing airline-specific issues can reduce total delays.
   * **late\_aircraft\_ct vs. late\_aircraft\_delay (0.96):**  
     Late aircraft incidents are highly correlated with late aircraft delays, emphasizing the need to improve turnaround times.
2. **Moderate Positive Correlations:**
   * **weather\_ct vs. weather\_delay (0.66):**  
     Weather-related incidents moderately impact delays, but operational factors like carrier delays have a greater influence.
3. **Weak/Negative Correlations:**
   * **year and month:**  
     Minimal correlation with delays, suggesting that temporal factors are less influential compared to operational issues.

**Implications:**

* **Feature Selection:**  
  Focus on variables like carrier\_delay, late\_aircraft\_delay, and arr\_del15 for predictive modeling.
* **Resource Optimization:**  
  Prioritize reducing carrier and late aircraft delays to have the most significant impact on overall delay reduction.

This analysis highlights critical areas for optimization, forming a foundation for improving flight delay performance.

Compared delays by airline. Boxplots revealed significant variance in delays across different carriers, suggesting operational differences.

**5.3) Optimization techniques**:

The explanation for flight optimization project, focusing on the comparison of the Dynamic Programming (DP), Simulated Annealing (SA), Genetic Algorithm, and Prim's Algorithm based on the execution time chart:

Flight Optimization Problem

In air traffic management, flight optimization refers to finding the best routes, schedules, and traffic management solutions to minimize operational costs, delays, and maximize efficiency. Optimization algorithms like Dynamic Programming (DP), Simulated Annealing (SA), Genetic Algorithms, and Prim's Algorithm can be applied to solve such problems, each with its own strengths and weaknesses.

Algorithm Overview

1. Dynamic Programming (DP)

* Equation: For problems like the shortest path or scheduling, DP solves problems by breaking them down into simpler subproblems. The general form of a recurrence relation in DP could be:

dp[i]=min⁡(dp[i−1]+cost[i])\text{dp}[i]

where dp[i]\text{dp}[i] represents the minimum cost to reach the ii-th stage, and cost[i]\text{cost}[i] is the cost to transition between stages.

Pseudo Code for dynamic programming:

def dp\_optimization(data):

n = len(data)

dp = [float('inf')] \* n

dp[0] = 0 # Starting point

for i in range(1, n):

for j in range(i):

dp[i] = min(dp[i], dp[j] + cost(i, j)) # Find minimum cost path

return dp[n-1]

* Use in Flight Optimization: DP is ideal for flight schedule optimization where decisions depend on previous stages, and subproblems can be solved optimally.

1. Simulated Annealing (SA)

* Equation: The temperature equation controls the likelihood of accepting worse solutions during the optimization process:

P(E,E′)=exp⁡(−(E′−E)T)P(E, E')

where EE is the current solution's cost, E′E' is a neighboring solution's cost, and TT is the temperature (which decreases over time).

Pseudo Code for SA:

def simulated\_annealing():

T = initial\_temperature

current\_solution = random\_solution()

while T > stopping\_temperature:

neighbor = get\_neighbor(current\_solution)

if accept\_move(current\_solution, neighbor, T):

current\_solution = neighbor

T \*= cooling\_rate

return current\_solution

In pseudo code for SA, initial temperature and cooling temperature are key parameters that control the algorithm's exploration and convergence.

* Use in Flight Optimization: SA is useful for flight route planning and scheduling when the problem involves complex configurations and the possibility of finding near-optimal solutions.

3. Genetic Algorithm (GA)

* Equation: In a genetic algorithm, the fitness of a solution is evaluated, and crossover and mutation operations are applied:

fitness=f(solution)\text{fitness}

where the fitness function measures how good a solution is.

Pseudo Code:

def genetic\_algorithm():

population = initialize population()

for generation in range(max\_generations):

fitness = [evaluate\_fitness(ind) for ind in population]

parents = select\_parents(population, fitness)

offspring = crossover(parents)

offspring = mutate(offspring)

population = new\_generation(offspring)

return best\_solution(population)

* Use in Flight Optimization: GA is applied to problems like air traffic scheduling where multiple solutions evolve, and the algorithm gradually finds the best solution over several generations.

4. Prim's Algorithm

* Equation: Prim's algorithm finds a minimum spanning tree (MST) for a graph:

MST=min⁡(∑weights of the edges in the tree)\text{MST}

Pseudo Code:

def prim(graph):

mst = []

visited = [False] \* len(graph)

visited[0] = True

while len(mst) < len(graph) - 1:

min\_edge = find\_min\_edge(visited)

mst.append(min\_edge)

visited[min\_edge.destination] = True

return mst

* Use in Flight Optimization: Prim's algorithm can be used to minimize the cost of connecting airports in a flight network, ensuring the minimum cost route between all locations.
* Applicability:
  + DP: Best suited for problems where solutions can be broken down into overlapping subproblems (e.g., shortest path).
  + SA: Effective for large, complex problems where finding near-optimal solutions is sufficient (e.g., flight scheduling).
  + GA: Good for problems with a large search space and many possible solutions (e.g., air traffic management).
  + Prim's Algorithm: Ideal for problems involving the construction of a minimum spanning tree, such as optimizing the flight network between airports.

For flight optimization project, the choice of algorithm depends on the problem at hand:

* F route optimization Dynamic Programming and Prim’s Algorithm may be ideal due to their efficiency in such scenarios.
* For more complex scheduling or resource allocation problems where near-optimal solutions are acceptable, Simulated Annealing and Genetic Algorithms offer better flexibility and may outperform DP and Prim's Algorithm in terms of practical applicability.

**5.4) Shortest path between any source and destination using different algorithms:**

The GUI interface allows users to input a source and destination airport code (like ABY for Albany, GA, and LAX for Los Angeles), select one of the shortest path algorithms, and then calculate and display the shortest path, the path length, and the execution time taken by the algorithm.

The algorithms used for pathfinding are:

* Dijkstra’s Algorithm
* Bellman-Ford Algorithm
* Bidirectional Dijkstra Algorithm
* A\* Algorithm

1. Dijkstra’s Algorithm: Dijkstra’s algorithm finds the shortest path from a source node to a target node in a weighted graph, where all edge weights are non-negative.

Pseudo code

def dijkstra\_algorithm(graph, source, target):

check\_node\_exists(graph, source)

check\_node\_exists(graph, target)

return nx.dijkstra\_path(graph, source, target)

* check\_node\_exists(graph, source): Verifies if the source node exists in the graph.
* check\_node\_exists(graph, target): Verifies if the target node exists in the graph.
* nx.dijkstra\_path(graph, source, target): Calls NetworkX’s built-in Dijkstra algorithm to find the shortest path from source to target.

1. Bellman-Ford Algorithm: Bellman-Ford is another shortest path algorithm that can handle graphs with negative edge weights. It is slower than Dijkstra's and can also detect negative weight cycles.

Pseudo code

def bellman\_ford\_algorithm(graph, source, target):

check\_node\_exists(graph, source)

check\_node\_exists(graph, target)

return nx.bellman\_ford\_path(graph, source, target)

* check\_node\_exists(graph, source): Verifies if the source node exists in the graph.
* check\_node\_exists(graph, target): Verifies if the target node exists in the graph.
* nx.bellman\_ford\_path(graph, source, target): Calls NetworkX’s implementation of the Bellman-Ford algorithm to find the shortest path from the source to the target.

1. Bidirectional Dijkstra: Bidirectional Dijkstra improves the standard Dijkstra algorithm by running two simultaneous searches: one from the source and the other from the target. When the two searches meet, the shortest path is found.

Pseudo code

def bidirectional\_dijkstra(graph, source, target):

check\_node\_exists(graph, source)

check\_node\_exists(graph, target)

return nx.bidirectional\_dijkstra(graph, source, target)[1]

* check\_node\_exists(graph, source): Verifies if the source node exists in the graph.
* check\_node\_exists(graph, target): Verifies if the target node exists in the graph.
* nx.bidirectional\_dijkstra(graph, source, target): Runs bidirectional Dijkstra using NetworkX’s built-in function. The result is a tuple, and we extract the shortest path from it by accessing the second element [1].

1. A\* Algorithm: A\* (A-star) is an informed search algorithm that combines the benefits of Dijkstra’s algorithm and heuristics. The heuristic helps guide the search towards the target, making it more efficient.

Pseudo code

def a\_star\_algorithm(graph, source, target):

check\_node\_exists(graph, source)

check\_node\_exists(graph, target)

return nx.astar\_path(graph, source, target, heuristic=lambda u, v: 0)

* check\_node\_exists(graph, source): Verifies if the source node exists in the graph.
* check\_node\_exists(graph, target): Verifies if the target node exists in the graph.
* nx.astar\_path(graph, source, target, heuristic=lambda u, v: 0): Calls NetworkX’s implementation of the A\* algorithm with a custom heuristic function. Here, we are using a trivial heuristic (000), meaning it behaves like Dijkstra’s algorithm.

**Steps to Use the GUI Interface:**

1. **Source and Destination Input**:
   * Users are prompted to enter the **source** and **destination** airports using text fields.
   * These fields take **airport codes** (like ABY for the source and LAX for the destination) to represent the starting and ending locations of the flight.
2. **Algorithm Selection**:
   * A dropdown menu (or radio buttons) allows the user to choose from one of the four available algorithms:
     + **Dijkstra**: Best known for finding the shortest path in a graph, particularly when all edges have non-negative weights.
     + **Bellman-Ford**: Works even with graphs that have negative weights and detects negative weight cycles.
     + **Bidirectional Dijkstra**: Runs two Dijkstra searches, one forward from the source and one backward from the destination, to speed up the search process.
     + **A\* (A-star)**: Uses a heuristic to find the shortest path, often faster than Dijkstra because it incorporates information about the destination.
3. **Calculate the Shortest Path**:
   * Once the source, destination, and algorithm are selected, the user clicks a "Calculate" button.
   * The application will then perform the selected algorithm to compute the shortest path from the source to the destination.
4. **Display Results**:
   * **Path**: The program will output the list of airports (or cities) that the path traverses in order.
   * **Path Length**: It will also show the distance or weight of the path (e.g., the total distance in miles or kilometers).
   * **Execution Time**: This is the time the algorithm took to find the solution. It provides performance feedback and helps in comparing the efficiency of different algorithms.

**Comparison of Algorithms:**

The GUI will display the results from the selected algorithm, including:

* The path taken between the source and destination.
* The distance (or path length).
* The execution time of the algorithm, which gives insight into how long the algorithm took to calculate the shortest path.

**Flow of the Application:**

1. The user provides a source and destination.
2. They select the algorithm they want to use to find the shortest path.
3. The program computes the shortest path and displays:
   * The path taken (in terms of airport codes).
   * The length of the path (distance between airports).
   * The execution time (time taken by the algorithm to compute the result).
4. The user can easily compare the execution times and paths calculated by different algorithms using the graphical representation (like the bar chart) shown in the image.

* The **GUI interface** offers a simple way for users to interact with pathfinding algorithms.
* Users can input any pair of source and destination airport codes to calculate and compare the shortest paths.
* **Execution time** comparison allows users to see how different algorithms perform in terms of speed.

**5.5) Real Time Analysis:**

The system combines historical data and real-time inputs to analyze airport delays.

1. **Optimized Dataset Analysis**:
   * The bar chart visualizes total delays (in minutes) for different airports. This dataset is pre-processed and optimized for quick insights into delay patterns (e.g., ALB shows the highest delays).
2. **Real-Time Data Input**:
   * Users can manually add live flight delay details, such as arrival, carrier, NAS, and late aircraft delays, for specific airports (e.g., CCU). This allows dynamic updates to the analysis.

By integrating historical trends with real-time data, the system provides a clear overview of delays and actionable insights for delay management.

**5.6) Delay Prediction:**

This visualization predicts delays using an optimized dataset and algorithms. Here's a breakdown:

**Key Elements:**

1. **Inputs:**
   * Airport codes are to be entered for example: **ATL** (Atlanta) and **ORD** (Chicago O'Hare).
2. **Algorithms Used:**
   * **Dijkstra's Algorithm**:
     + Finds the shortest path between the airports for example: ATL and ORD via LAX with a total delay of **95 minutes**.
   * **A\* Algorithm**:
     + Similarly computes the shortest path with the same total delay of **95 minutes** but integrates heuristics for efficiency.
   * **Machine Learning (ML) Prediction**:
     + Predicts the delay based on historical data and ML models, yielding a delay of **47.35 minutes** for ORD.

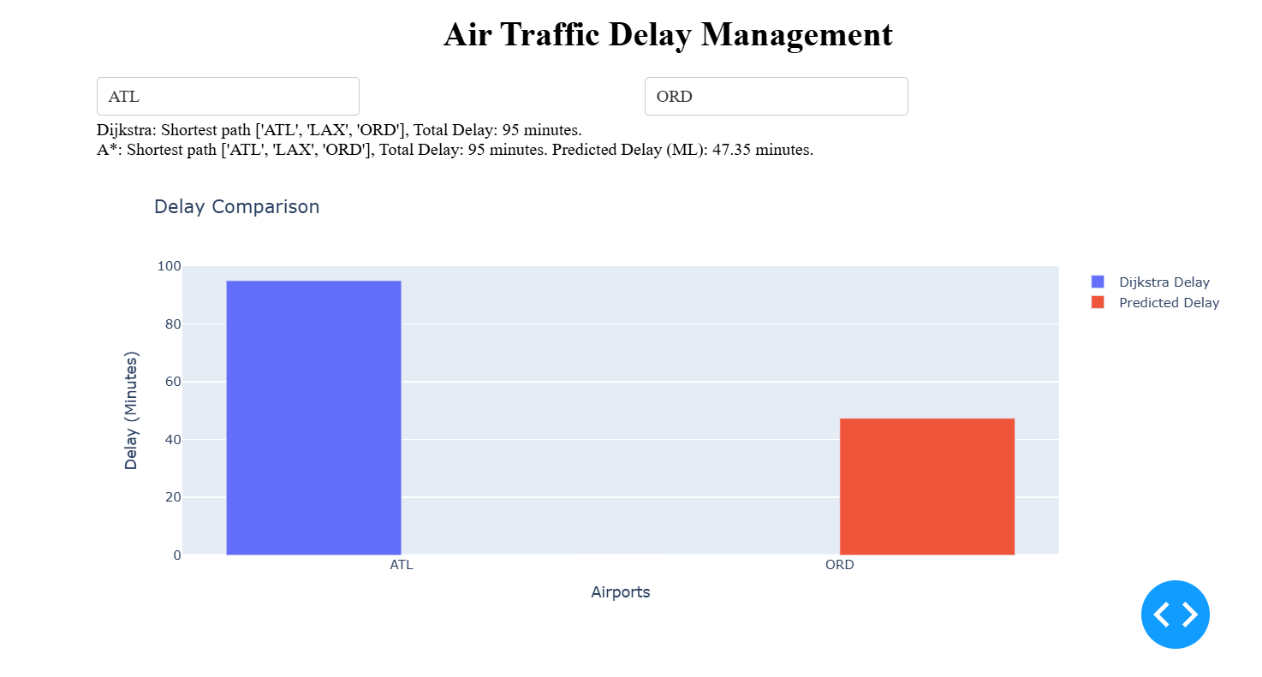


Fig 5.5.a: Analysis of Actual v/s predicted delay

1. **Bar Chart Comparison:**
   * **Dijkstra Delay** (blue bar): Displays the computed delay for ATL (95 minutes).
   * **Predicted Delay** (red bar): Uses the ML model to predict delays for ORD, showing 47.35 minutes.

The system integrates pathfinding algorithms (Dijkstra, A\*) and machine learning predictions to analyze and compare delays. The chart provides a clear visual distinction between computed delays (from pathfinding) and ML-based predictions, emphasizing efficiency improvements when leveraging predictive models.

**CHAPTER 6 – RESULTS**

This subsection summarizes the performance of each optimization technique with their complexity explanations, followed by a comparison of their results. We also provide a graphical representation of the outcomes to illustrate the differences in efficiency.

**6.1) Time Complexities of each optimization techniques and pathfindings**

1. Dynamic Programming:

In our project, the algorithm processes a list of delays, and for each delay, it performs some operation. Iterating over a list of n elements, the algorithm performs a constant amount of work on each element. Therefore, the total time to process the entire list is proportional to n, the number of elements leading the overall complexity to O(n).

1. Simulated Annealing:

O(k⋅m) = O(logcooling\_rate​(Tstop/​Tinit​​)⋅m)

Where

* m is the number of flights, and
* k is the number of iterations determined by the cooling schedule.

1. Genetic Algorithm(GA):

* Generating an initial population of size P (e.g., random subsets or schedules derived from the dataset) takes O(P⋅N) as as each individual in the population involves processing all N rows.
* Evaluating the fitness for minimizing delays takes O(N) hence per generation it takes O(P.N).
* Combining parent solutions to create offspring for merging flight schedules or swapping portions of data takes O(N).
* Overall complexity: O(G.P.N)

where,

G: Number of generations.

P: Population size.

N: Number of flights.

1. Prim’s Algorithm:

* Building the graph: O(E), where E is the number of edges.
* Constructing the Minimum Spanning Tree (MST) takes O((V+E)logV) using a priority queue
* Overall complexity: O((V+E)logV)

1. Dijkstra’s Algorithm:

Often, uses a priority queue (often implemented as a binary heap), resulting in a time complexity of:

O((E+V)logV)

1. Bellman-Ford Algorithm:

* The Bellman-Ford algorithm has a time complexity of O(V\*E). This is because the algorithm iterates over all edges for V−1 times.

Where,

* V is the number of vertices and
* E is the number of edges in the graph.

1. Bidirectional Dijkstra Algorithm:

The bidirectional Dijkstra algorithm has a time complexity of: O(E⋅logV)

1. A\* Algorithm:

* The time complexity of the A\* algorithm depends on the heuristic. In the worst case (using a trivial heuristic), it behaves like Dijkstra’s algorithm: O((E+V)logV)
* With a good heuristic, A\* can be faster than Dijkstra because it prunes parts of the search space

6.2) Execution Time Comparisons

6.2.1) Execution Time Comparisons for optimization techniques:

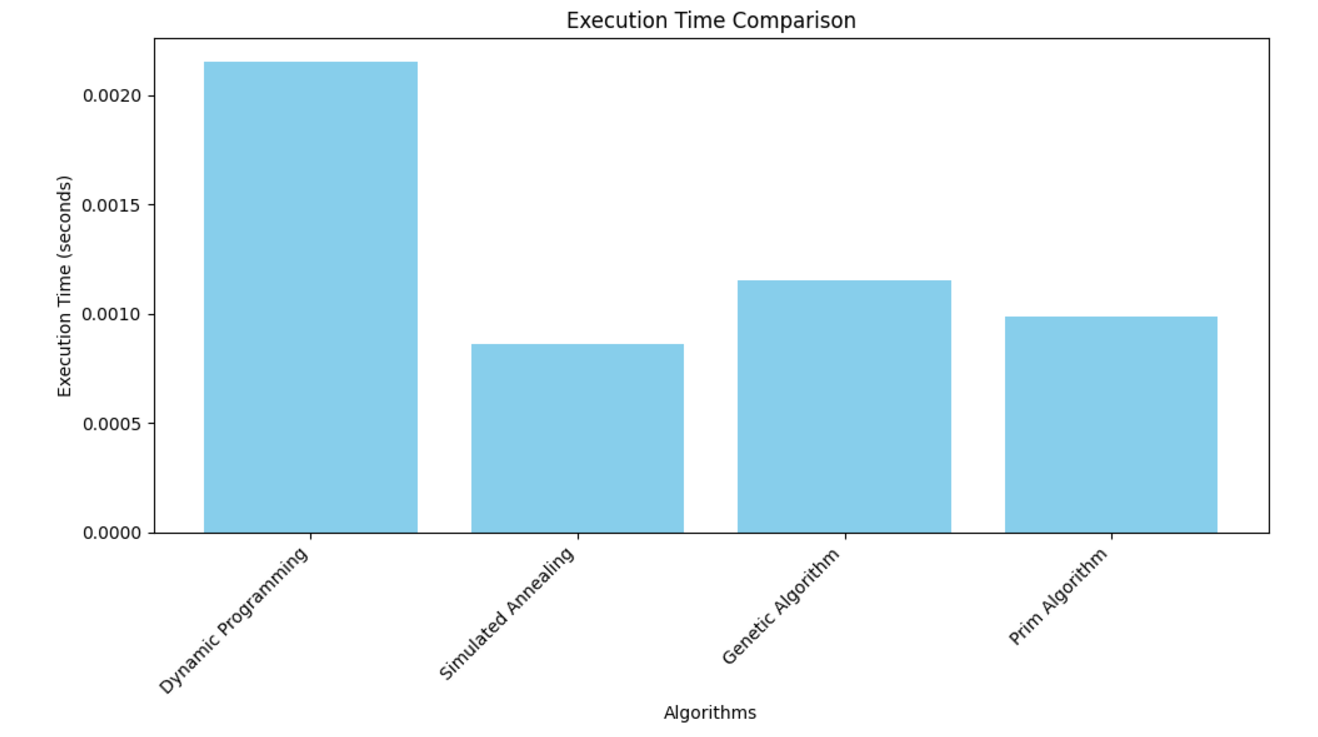


Fig 6.2.1.a: Execution Time comparison between optimization techniques

The above figure presents the execution times for the four algorithms:

* Dynamic Programming takes the longest time to compute, likely due to its exhaustive search of all possible subproblems and recursive nature.
* Simulated Annealing performs better but still requires significant time as it searches for a good solution through random moves and annealing.
* Genetic Algorithm has a moderate execution time, as it uses population-based approaches (selection, crossover, mutation), which are computationally intensive but can be parallelized.
* Prim's Algorithm has the fastest execution time, which is expected since it is a greedy algorithm and can be implemented efficiently with priority queues or binary heaps.

Algorithm Performance

* Execution Time: From the execution time chart, it is clear that Prim's Algorithm is the fastest, likely due to its efficient greedy approach. Simulated Annealing and Genetic Algorithm take slightly longer as they explore multiple solution paths, but they are more flexible for complex problems. Dynamic Programming takes the longest time, which could be attributed to its exhaustive nature in solving overlapping subproblems.

6.2.2) Execution Time comparisons for pathfinding algorithms

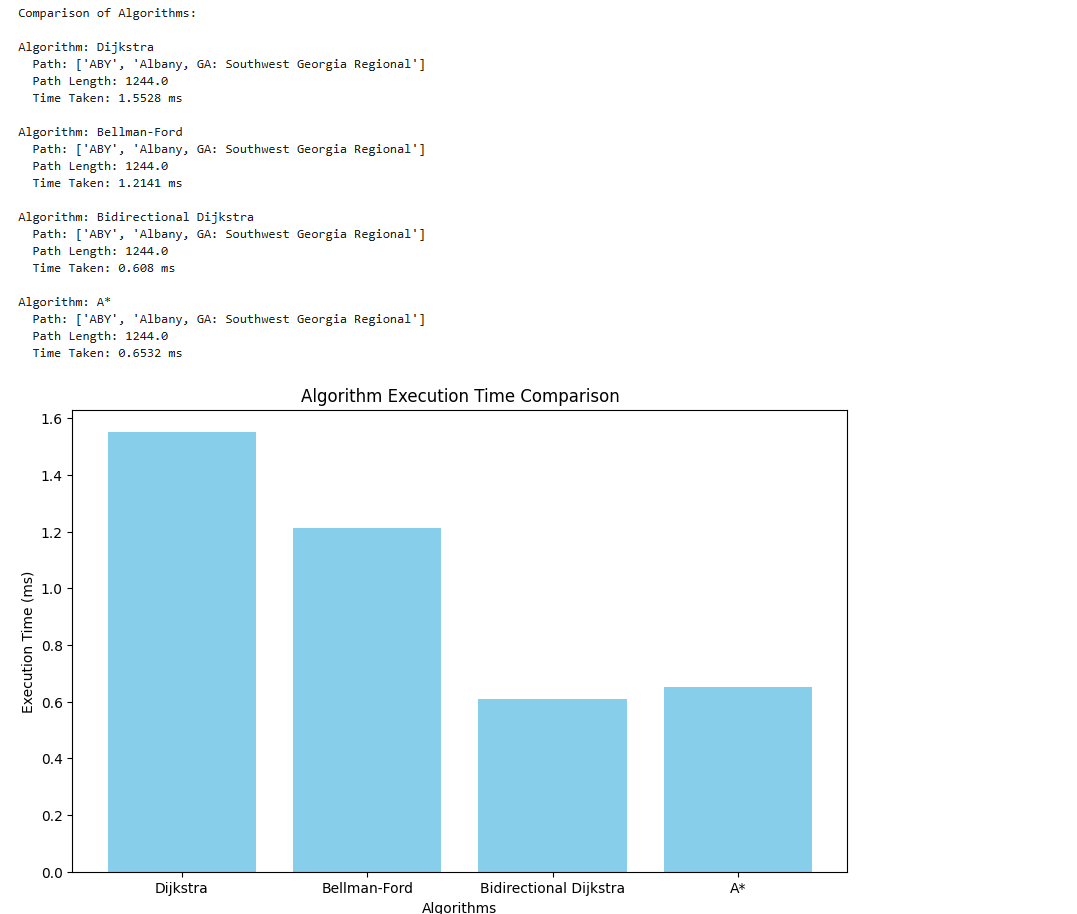


Fig 6.2.2.a: Execution Time comparison between pathfinding algorithms

In the figure, it’s clearly visible that the **execution time** of each algorithm is compared. Here’s a breakdown:

* **Dijkstra’s Algorithm** has a longer execution time compared to A\* and Bidirectional Dijkstra, but it's widely used for single-source shortest path problems.
* **Bellman-Ford** takes longer than Dijkstra for this task, especially in graphs with many edges.
* **Bidirectional Dijkstra** has least execution time making it suitable for large scale datasets.
* **A\*** typically takes the least time for finding the shortest path, as it uses heuristics to guide the search towards the destination, often making it more efficient than Dijkstra in specific cases.

6.3) Real-time analysis result

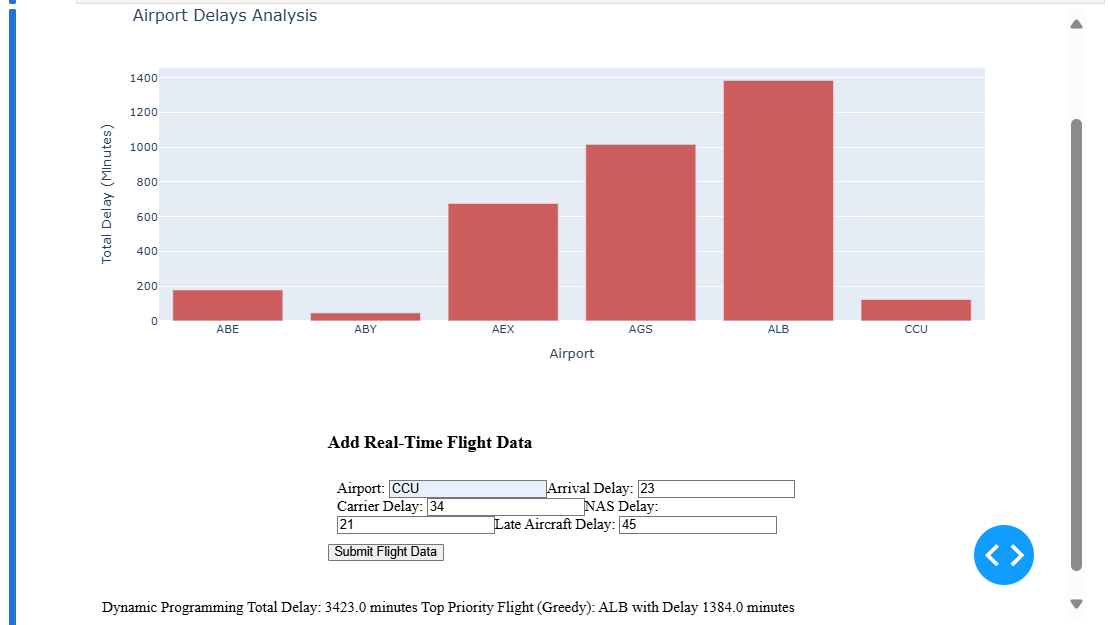


Fig 6.3: Real time analysis of delays

* Algorithms like **Dynamic Programming** calculate cumulative delays across all flights, providing a total delay metric (e.g., 3423 minutes).
* The **Greedy Algorithm** identifies the top-priority flight or airport with the most significant delay (e.g., ALB with 1384 minutes) for immediate focus.

**CHAPTER 7 – CONCLUSION AND FURTURE ENHANCEMENT**

**Conclusion:**

The main objective of the current project is to solve modern air traffic management problems by incorporating many different approaches such as the multi-optimization types, shortest path algorithms, and real-time analysis into one general framework. Besides greedy algorithms, linear programming, genetic algorithms, graph-based methods, dynamic programming, Prim's algorithm, and simulated annealing are used as a broad set of algorithms in the system. The shortest path algorithms such as Bellman-Ford and Dijkstra along with optimization techniques have proved beneficial to minimizing delay, operational inefficiencies, and an increase in air traffic flow overall optimization in air traffic operations. The accomplishments can serve as strong guidelines to the assistance of various techniques and the ability to make a more efficient, reliable, and adaptive management of air traffic systems.

**Future Enhancement:**

This future research may seek to further augment the real-time data feed with weather conditions, airspace use, and flight-specific variables in a truly intelligent interpretation of the flight schedule and routing decisions. Also, prediction power and proactive optimization of routing would be improved with machine learning algorithms for more accurate forecasting of delays. To make decision-making more holistic, the framework may be extended toward multi-objective optimization, in terms of fuel consumption, environmental impacts, and safety, to name the obvious ones. Further, it would give indications of scalability and performance if such an effort were explored in larger, global air traffic systems where international airspace limitations and airport capacities would have to be taken into consideration.

**REFERENCES**

Follow IEEE format and if websites are there put it at the end and also books.

1. T. Zhaoa, J. Escribano M, Mingwei Zhangb, S. Fub, Y. Fenga, Mireille E., Arnab M., P. Angeloudisa, W. Yotto Ochienga ,” Enhancing air traffic operational efficiency by reducing network scale”, 2024, Aerospace Traffic and Safety
2. Karthik Gopalakrishnan , Hamsa Balakrishnan,” Control and Optimization of Air Traffic Networks”, Annu. Rev. Control Robot. Auton. Syst. 2021.
3. A. Sikoraa, R. Czvyba ,” Independent Flight Management System for Unmanned VTOL Aircraft”,2018
4. Y. Yaguchi ,” Flight Plan Management System for Unmanned Aircraft Vehicles Using Blockchain”, 2021 International Conference on Unmanned Aircraft Systems (ICUAS)
5. S. Kanury,Y. D. Song,” Flight Management of Multiple Aerial Vehicles Using Genetic Algorithms”,2016
6. Rahul Rampure, Raghav Tiruvallur, Vybhav Acharya, Shashank Navad, Preethi P ,” Air Traffic Management using a GPU-Accelerated genetic algorithm “,2023
7. Deepudev S, Palanisamy P, Varun P. Gopi, Manjunath K. Nelli and Asokkumar K ,” Performance Improvement of Air Traffic Flow Management Ground Delay Program using Machine-learning and Mixed Integer Linear Programming based Algorithm”,2020
8. Mehran Makhtoumi ,” Delay Mitigation in Air Traffic Flow Management”,2023
9. C.Ramee,J.A.Kim,C.Justin,S.Briceno ,” Aircraft Flight Plan Optimization with Dynamic Weather and Airspace Constraints”,2020
10. Krishan Kumar, Ravendra Singh, and Zubair Khan ,” Air Traffic enroute conflict detection using adaptive resonance theory map neural networks (ART1)”,2014
11. R. A. Altava, J. C. Mere, D. Delahaye, T. Miquel,” Flight Management System Pathfinding Algorithm for Automatic Vertical Trajectory Generation”
12. Sadeque Hamdan, A. Cheaitou, Oualid Jouini, Zied Jemai, Imad Alsyouf and Maamar Bettayeb ,” An Environmental Air Traffic Flow Management Model “,2019
13. Abdelghani Fadil, Abdelghani Fadil, Yang Yang, and Bin Hao ,” A Scenario Optimization Approach for Air Traffic Flow Management with Sector Capacity Uncertainty”
14. Qing Cai, Haojie Ang, and Sameer Alam ,” A Multiobjective Optimization Approach for Reducing Air Traffic Collision Risk”,2021
15. Abdelhamid Boujarif, Sadeque Hamdan, and Oualid Jouini ,” Impact of Airport Capacity Optimization on the Air Traffic Flow Management”,2021
16. Wen Tian, and Minghua Hu ,” Study of Air Traffic Flow Management Optimization Model and Algorithm Based on Multi-Objective Programming “,2020
17. Wen-Bo Dua, M. Yuan Zhanga, Yu Zhangc, Xian-Bin Caoa, Jun Zhangd, “Delay causality network in air transport systems”,2018
18. Qiang Li, Ranzhe Jing ,” Characterization of delay propagation in the air traffic network ”2021
19. Dhawal Thakkar , Balamurugan Palaniappan,” Aircraft routing using dynamic programming and reinforcement learning: A customer-centric approach”,2024
20. Zhiwei Xing and Yunxiao Tang,” The model for optimizing airport flight delays allocation”,2016