A PROJECT REPORT ON

AIR ROUTE FINDER

SUBMITTED TO

CHANDIGARH UNIVERSITY

By

Sweta Dey
24MCI10247

In partial fulfillment for the award of the degree of

MASTER OF COMPUTER APPLICATION

IN
ARTIFICIAL INTELLIGENCE AND MACHINE LEARNING

UNIVERSITY INSTITUTE OF COMPUTING,

CHANDIGARH UNIVERSITY,

ACKNOWLEDGEMENT

In performing our project, I would like to show my gratitude to all the faculty members of University Institute of Computing, Chandigarh University for giving me a good guideline for the project throughout numerous consultations. I would also like to expand my deepest gratitude to all those who have directly and indirectly guided me in completion of this project.

In addition, a thank you to my **Prof. Dr. Nisha Sharma**, my mentor who introduced me to the Art of Computer Programming, and her passion for the "underlying structures" had lasting effect. I also thank the University of Chandigarh for consent to include copyright pictures as a part of our paper.

I am also thankful to my classmate, **Nancy Nain(24MCI10124)**, for her encouragement and constructive feedback during this project. Her collaboration and support made the learning process enjoyable and enriching. Beside her, my other classmates have also made valuable comment suggestions on this project which gave me an inspiration to improve my application.

I thank all the people for their help directly and indirectly to complete my assignment.

CONTENT

PREFACE

Air travel plays a vital role in global connectivity, with efficiency and route optimization being key factors in ensuring smooth transportation. However, finding the shortest and most efficient route between airports can be a challenging task due to multiple route options, varying distances, and different connectivity levels.

This project leverages the *A search algorithm**, a well-known pathfinding technique, to determine the shortest air route between two airports. By utilizing a predefined air route graph, the system efficiently calculates the optimal path based on real-world distances and heuristics. The project features a graphical user interface (GUI) built with CustomTkinter, allowing users to select starting and destination airports and visualize the computed route on an interactive map.

While this implementation provides a solid foundation for route optimization, future enhancements could include real-time flight data integration, weather conditions, airline preferences, and dynamic cost estimation. This project serves as a valuable tool for exploring pathfinding algorithms and improving air travel efficiency.

PROBLEM STATEMENT

Given a network of airports and predefined air routes with associated distances, the goal is to develop an efficient pathfinding system that determines the shortest route between two selected airports. The challenge is to implement a heuristic-based search algorithm that accurately computes the optimal path while providing a user-friendly interface for visualization.

In practice, air route optimization depends on multiple factors such as direct connections, total travel distance, and real-time flight conditions. While the *A search algorithm** serves as a strong foundation for finding the shortest route, real-world applications may require additional considerations, such as dynamic flight schedules, weather conditions, and airline-specific constraints for more accurate and practical routing solutions.

3. INTRODUCTION

Air route optimization is a critical aspect of aviation and logistics, as finding the most efficient flight path between airports can significantly impact travel time, fuel consumption, and overall operational efficiency. Passengers and airline operators often face challenges in determining the shortest or most cost-effective route due to the complexity of interconnected flight networks. Factors such as direct connections, layovers, and distance play a crucial role in determining the best possible route between two locations. As the aviation industry continues to expand with an increasing number of flights and airports, an intelligent route-finding system becomes essential for optimizing travel.

In recent years, graph search algorithms have proven to be effective in solving shortest path problems in transportation networks. By leveraging advanced search techniques, we can analyze predefined air routes and their respective distances to determine the optimal path between airports. This project employs the *A search algorithm**, a widely used heuristic-based pathfinding approach, to compute the shortest air route efficiently. The algorithm considers both the actual travel cost and an estimated heuristic to make intelligent routing decisions.

To enhance usability, this system is integrated into an interactive graphical interface using CustomTkinter, allowing users to select their starting and destination airports and visualize the optimal route on a graphical map. The application dynamically processes user input, executes the pathfinding algorithm, and highlights the computed route in real time, providing an intuitive experience for users.

While this project establishes a fundamental approach to air route optimization, there is potential for future enhancements. Integrating real-time flight data, weather conditions**,** airline-specific constraints, and cost analysis can improve accuracy and practicality. By leveraging intelligent pathfinding techniques, this project demonstrates how computational methods can improve decision-making in air travel and logistics.

3.1 Objective

- **Develop an air route optimization system**
  The system will compute the shortest and most efficient route between airports.
  It will help users visualize and determine optimal flight paths.

- **Implement the A search algorithm***
  The A* algorithm will be used to find the shortest path between airports.
  It ensures efficiency by combining actual travel costs with heuristic estimates.

- **Create a user-friendly interface**
  A **CustomTkinter**-based graphical interface will allow users to select airports.
  The interface will display the computed route visually for better understanding.

- **Ensure accuracy and efficiency**
  The system will utilize well-defined airport locations and distances.
  The algorithm will be optimized to ensure quick and precise route calculations.

- **Enhance graphical representation**
  Airports and routes will be displayed on a graphical map.
  The shortest path will be highlighted for easy interpretation.

- **Enable future scalability and improvements**
  The system will allow integration of real-time flight schedules and weather data. Future enhancements may include cost analysis, airline-specific routes, and additional optimization techniques.

3.2 Scope of the Project

- **Automated Air Route Optimization**
  This project focuses on developing an intelligent system to determine the shortest and most efficient air travel routes between airports. It helps users quickly identify optimal flight paths based on predefined routes and distances.

- **A Search Algorithm Integration***
  The system implements the A* search algorithm to find the shortest path between airports. The algorithm combines actual travel costs with heuristic estimates, ensuring fast and efficient route calculations.

- **Interactive Graphical User Interface**
  A **CustomTkinter**-based interface allows users to input departure and destination airports, visualize the air route network, and see the shortest route displayed dynamically on a graphical map.

- **Graph Representation of Air Routes**
  The project includes a graphical representation of airport locations and their connections. The shortest route is visually highlighted, making it easier for users to interpret travel paths.

- **Distance Calculation and Route Optimization**
  The system calculates the shortest travel distance between airports and provides users with the most cost-effective route. This enhances decision-making for travel planning and logistics management.

- **Future Enhancements and Scalability**
  The system is designed to allow future improvements, such as real-time integration of flight schedules, weather conditions, and cost-based route optimization. Additional algorithms and machine learning techniques can also be incorporated to refine predictions and efficiency.

3.3 Survey of Technology

Operating System: Windows Operating System

Software used: VS Code

Language used: Python(Tkinter)

3.4 Requirements

Hardware requirements for running this project are as follows:

OS: - Windows XP and above

RAM: Ideally 2GB or above

Graphics: Integrated / Min 2GB (discrete)

Hard disk: Min 128 GB

Software requirements for running this application are as follows:
RAM: 1 GB or above.

Memory Space: 500 MB or above.

Languages and Platform used:

Platform: Output window

Language: Python

4. LITERATURE REVIEW

4.1 Overview of A* Search Algorithm

A* (A-star) is a widely used pathfinding and graph traversal algorithm that efficiently finds the shortest route between nodes in a weighted graph. It operates by combining two key techniques: Actual Cost (g-score**)**, which measures the distance from the starting point to the current node, and Heuristic Estimate (h-score), which predicts the cost to reach the goal. The sum of these two scores (f-score = g + h) helps prioritize nodes that are most likely to lead to the optimal path.

This algorithm is highly efficient, ensuring an optimal solution when an admissible heuristic is used. A* is widely applied in navigation systems, AI for games, robotics, and network routing due to its accuracy and scalability.
Key advantages of A* Search Algorithm include:

- **Optimality**: Guarantees the shortest path when using an admissible heuristic.
- **Efficiency**: Finds the best route faster than uninformed search methods like Dijkstra's algorithm.
- **Flexibility**: Adapts to various heuristic functions to optimize performance.
- **Scalability**: Performs well in large graphs with complex route networks.

In this project, A* is employed for air route optimization, determining the shortest path between airports based on pre-defined connections. By integrating heuristic-based pathfinding, the system provides an efficient and visually interactive way to plan optimal flight routes.

4.2 Applications of A* in Air Route Finder

A* is a highly efficient search algorithm used in air route optimization, ensuring the shortest and most cost-effective flight paths between airports. Its heuristic-driven approach enhances real-time decision-making in aviation networks.

Key Applications:

- **Optimal Flight Path Calculation:** Determines the shortest and most efficient route between two airports based on distance and connectivity.
- **Real-Time Route Adjustment:** Helps reroute flights dynamically in case of delays, weather disruptions, or air traffic congestion.
- **Fuel Efficiency Optimization:** Reduces fuel consumption by selecting the most direct and cost-effective flight path.
- **Navigation System Integration:** Used in flight planning software to assist pilots and air traffic controllers in route selection.
- **Emergency and Alternate Routing:** Identifies alternative airports and detours in case of emergencies or flight diversions.
- **Scalability for Large Networks:** Efficiently processes complex global air route networks, making it suitable for airline logistics and transportation planning.

A*'s speed, accuracy, and adaptability make it an essential tool for optimizing air travel, reducing operational costs, and improving overall flight efficiency.

4.3 Existing Solutions and Tools

Several existing solutions and tools are available for laptop price prediction, leveraging various machine learning models and online databases.

- **Flight Planning Software:** Platforms like SkyVector and FlightAware help plan routes but require manual input and lack dynamic optimization.
- **Graph-Based Routing Algorithms:**
  - **Dijkstra's Algorithm:** Finds the shortest path but is slow for large networks.
  - **Bellman-Ford Algorithm:** Handles negative weights but is inefficient.
  - *A Search Algorithm:** Optimizes routes efficiently using heuristics.
- **Air Traffic Management Systems:** Used by FAA (NextGen) and SESAR for large-scale air traffic control but not for individual route optimization.
- **Navigation APIs & GIS Tools:** Google Maps API and OpenRouteService provide shortest paths but are designed for roads, not air traffic.

Our A*-based air route finder improves efficiency by dynamically calculating the shortest, most cost-effective flight paths in real-time.

5. SYSTEM DESIGN AND ARCHIECTURE

5.1 Data Structures

- **Graph (graph):**
  - Represented as an adjacency list (dictionary of dictionaries).
  - Stores airports as keys and their neighboring airports with distances as values.

- **Airport Positions (airport_positions):**
  - A dictionary mapping each airport to its (x, y) coordinates for visualization.

- **Priority Queue (open_set):**
  - Implemented using a min-heap (heapq) to efficiently select the next airport with the lowest cost in A* search.

- **Cost and Heuristic Scores (g_score, f_score):**
  - Dictionaries storing the actual cost from the start node and the estimated total cost (heuristic + actual cost).

- **Path Reconstruction (previous):**
  - A dictionary tracking the previous airport in the shortest path to reconstruct the route.

- **Tkinter UI Components (start_combobox, end_combobox, route_text, distance_text):**
  - Used to store user selections and display computed routes and distances.

5.2 Algorithm Implementation

- **Graph Representation**
  - Define the air route network as an adjacency list (dictionary of dictionaries).
  - Store airport positions for visualization.

- **A Search Algorithm***
  - Initialize the open set (priority queue) with the start airport.
  - Maintain cost (g_score) and estimated cost (f_score) dictionaries.
  - Use the heuristic function (Euclidean distance) to estimate the shortest path.
  - Expand nodes by selecting the airport with the lowest f_score.
  - Update costs and reconstruct the path once the destination is reached.

- **Graph Visualization (Tkinter)**
  - Draw nodes (airports) and edges (routes) on a canvas.
  - Display airport labels and distances between connected nodes.

- **User Input & Route Calculation (CustomTkinter)**
  - Take user input for start and destination airports via dropdowns.
  - Run the A* algorithm to compute the shortest route.
  - Highlight the optimal path on the graph.

• **Results Display**
  - Show the computed shortest path and total distance.
  - Handle invalid inputs and display error messages when necessary.

6. METHODOLOGY

6.1 Step-by-Step Process of Random Forest

- **Initialize Data Structures** – Define the air route graph using an adjacency list and store airport coordinates for heuristic calculations.
- **Initialize Open and Closed Sets** – Use a priority queue (min-heap) for the open set and dictionaries to track g_score (actual cost) and f_score (estimated total cost).
- **Add Start Node** – Insert the start airport into the open set with f_score initialized as the heuristic distance to the destination.
- **Expand Nodes** – Extract the airport with the lowest f_score, check if it's the destination, and explore its neighbors.
- **Update Costs** – Calculate the tentative g_score for each neighbor, update the f_score, and store the previous airport for path reconstruction if a better route is found.
- **Repeat Until Goal Reached** – Continue expanding nodes until the destination is reached or the open set is empty (no path exists).
- **Reconstruct Path & Compute Distance** – Trace back from the destination to the start using the previous node mapping, compute the total distance, and return the shortest route.

6.2 Input and Output Specifications

Input: The user selects the starting airport and destination airport from a predefined list of available airports in the system.
Output: The system calculates and displays the shortest air route between the selected airports along with the total travel distance. The path is visually highlighted on the map.

6.3 User Interface Design

Tkinter is a built-in Python library for creating GUIs. In this project, it enables users to select airports via dropdowns, find the shortest route with a button, and reset the interface. A canvas visually displays the airport network, highlighting routes dynamically, while labels show the calculated path and distance for an intuitive user experience.

- **Title:** Displays "A* Air Route Finder."
- I**nputs:** Users select Starting Airport and Destination Airport from dropdowns.
- **Route Calculation:** Click "Find Route" to compute the shortest path using A*.
- **Output:**
  - Displays the shortest route.
  - Shows the total distance.
  - Highlights the route on a map.
- **Reset Button:** Clears inputs and refreshes the interface.

7. CODE IMPLEMENTATION

7.1 Algorithm Code

air.py > heuristic

```python
import sys
import heapq
import customtkinter as ctk
from tkinter import messagebox
import math

# Define air route graph as adjacency list
graph = {
    'Chandigarh': {'Delhi': 4, 'Ambala': 3},
    'Delhi': {'Chandigarh': 4, 'Shimla': 5, 'Ambala': 12},
    'Ambala': {'Chandigarh': 3, 'Dehradun': 7, 'Delhi': 12},
    'Shimla': {'Delhi': 5, 'Amritsar': 16},
    'Dehradun': {'Ambala': 7, 'Hindon': 2, 'Shimla': 10},
    'Hindon': {'Dehradun': 2, 'Chandigarh': 5},
    'Amritsar': {'Shimla': 16, 'Delhi': 5}
}

# Positions for heuristic calculations (approximate coordinates)
airport_positions = {
    'Chandigarh': (150, 150),
    'Delhi': (400, 150),
    'Ambala': (275, 275),
    'Shimla': (400, 470),
    'Dehradun': (275, 400),
    'Hindon': (350, 350),
    'Amritsar': (450, 400)
}
```

```python
# Heuristic function (Euclidean distance)
def heuristic(airport1, airport2):
    x1, y1 = airport_positions[airport1]
    x2, y2 = airport_positions[airport2]
    return math.sqrt((x2 - x1) ** 2 + (y2 - y1) ** 2)

# A* Algorithm
def a_star(graph, start_airport, end_airport):
    open_set = [(0, start_airport)]  # (priority, airport)
    g_score = {airport: sys.maxsize for airport in graph}
    g_score[start_airport] = 0
    f_score = {airport: sys.maxsize for airport in graph}
    f_score[start_airport] = heuristic(start_airport, end_airport)
    previous = {airport: None for airport in graph}

    while open_set:
        _, current_airport = heapq.heappop(open_set)

        if current_airport == end_airport:
            route = []
            while current_airport:
                route.append(current_airport)
                current_airport = previous[current_airport]
            return route[::-1], g_score[end_airport]
```

```
36  def a_star(graph, start_airport, end_airport):
54              for neighbor, weight in graph[current_airport].items():
55                  tentative_g_score = g_score[current_airport] + weight
56                  if tentative_g_score < g_score[neighbor]:
57                      g_score[neighbor] = tentative_g_score
58                      f_score[neighbor] = tentative_g_score + heuristic(neighbor, end_airport)
59                      previous[neighbor] = current_airport
60                      heapq.heappush(open_set, (f_score[neighbor], neighbor))
61
62      return [], sys.maxsize  # No path found
63
64  # GUI Setup
65  root = ctk.CTk()
66  root.title("A* Path Finder")
67
68  # Create the canvas
69  canvas = ctk.CTkCanvas(root, width=500, height=500, bg="white")
70  canvas.pack(pady=(10, 20))
71
```

```
72  # Functions to Draw and Highlight
73  def draw_graph():
74      canvas.delete("all")  # Ensure fresh redraw of the graph
75      for airport, neighbors in graph.items():
76          x1, y1 = airport_positions[airport]
77          for neighbor, weight in neighbors.items():
78              x2, y2 = airport_positions[neighbor]
79              canvas.create_line(x1, y1, x2, y2, fill='gray')
80              mid_x, mid_y = (x1 + x2) // 2, (y1 + y2) // 2
81              canvas.create_text(mid_x, mid_y, text=str(weight), font=("Arial", 10), fill="black")
82          canvas.create_oval(x1-5, y1-5, x1+5, y1+5, fill='blue')
83          canvas.create_text(x1, y1-10, text=airport, font=("Arial", 10))
84
85  def highlight_path(route):
86      for i in range(len(route) - 1):
87          x1, y1 = airport_positions[route[i]]
88          x2, y2 = airport_positions[route[i + 1]]
89          canvas.create_line(x1, y1, x2, y2, fill='red', width=2)
90
```

```
91  #Graph
92  draw_graph()
93
94  dropdown_frame = ctk.CTkFrame(root)
95  dropdown_frame.pack(pady=(20, 10))
96
97  ctk.CTkLabel(dropdown_frame, text="Starting Port:").grid(row=0, column=0, padx=10)
98  start_combobox = ctk.CTkComboBox(dropdown_frame, values=list(graph.keys()))
99  start_combobox.grid(row=0, column=1, padx=10)
100
101 ctk.CTkLabel(dropdown_frame, text="Destination Port:").grid(row=0, column=2, padx=10)
102 end_combobox = ctk.CTkComboBox(dropdown_frame, values=list(graph.keys()))
103 end_combobox.grid(row=0, column=3, padx=10)
104
105 button_frame = ctk.CTkFrame(root)
106 button_frame.pack(pady=(10, 20))
107
108 route_text = ctk.StringVar()
109 distance_text = ctk.StringVar()
```

```
111   def find_route():
112       start_airport = start_combobox.get().strip().upper()
113       end_airport = end_combobox.get().strip().upper()
114
115       if start_airport not in graph or end_airport not in graph:
116           messagebox.showerror("Error", "Invalid airport code.")
117           return
118
119       draw_graph()
120       route, distance = a_star(graph, start_airport, end_airport)
121
122       if distance == sys.maxsize:
123           messagebox.showinfo("Result", "No route found.")
124           return
125
126       highlight_path(route)
127       route_text.set(" -> ".join(route))
128       distance_text.set(f"{distance} km")
129
130   find_button = ctk.CTkButton(button_frame, text="Find Route", command=find_route)
131   find_button.grid(row=0, column=0, padx=10)
132
133   def reset():
134       start_combobox.set("")
135       end_combobox.set("")
136       route_text.set("")
137       distance_text.set("")
138       draw_graph()
```

```
140   reset_button = ctk.CTkButton(button_frame, text="Reset", command=reset, fg_color="red", hover_color="#cc0000")
141   reset_button.grid(row=0, column=1, padx=10)
142
143   result_frame = ctk.CTkFrame(root)
144   result_frame.pack(pady=(10, 20))
145
146   ctk.CTkLabel(result_frame, text="Shortest Route:", font=("Arial", 12, "bold")).pack()
147   route_label = ctk.CTkLabel(result_frame, textvariable=route_text, font=("Arial", 12))
148   route_label.pack()
149
150   ctk.CTkLabel(result_frame, text="Shortest Distance:", font=("Arial", 12, "bold")).pack()
151   distance_label = ctk.CTkLabel(result_frame, textvariable=distance_text, font=("Arial", 12))
152   distance_label.pack()
153
154   root.mainloop()
155
```

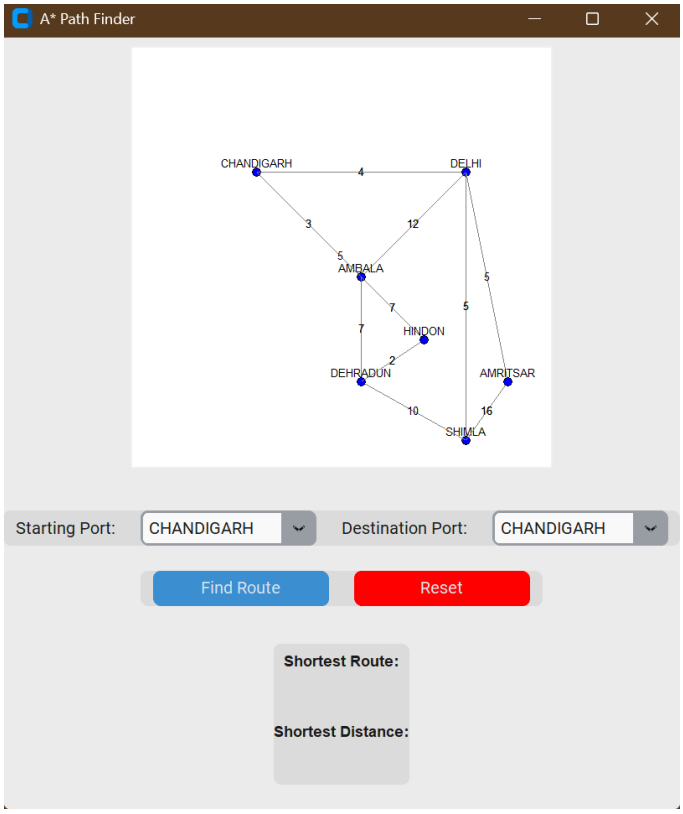
7.2 Explanation of Key Functions

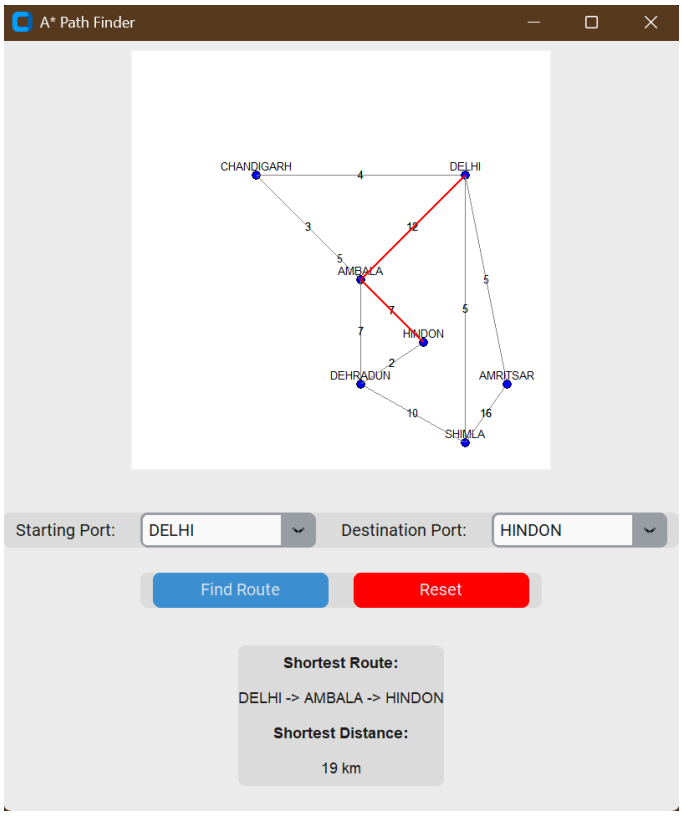
- **heuristic(node, goal)** – Calculates the estimated cost from the current node to the goal using a heuristic function (e.g., straight-line distance).
- **get_neighbors(node)** – Returns a list of neighboring airports along with their respective distances.
- **a_star_search(start, goal)** – Implements the A* algorithm to find the shortest air route by prioritizing paths with the lowest total estimated cost (g + h).
- **reconstruct_path(came_from, current)** – Traces back from the goal node to reconstruct the optimal path.
- **tkinter.Tk()** – Initializes the GUI window for user interaction.
- **tkinter.OptionMenu()** – Creates dropdown menus for selecting departure and destination airports.
- **tkinter.Button()** – Triggers the route-finding process when clicked.
- **canvas.create_line()** – Draws the flight path visually on the interface.

7.3 Data Input

The code loads and processes the dataset, representing airports and flight routes in a graph structure. Node coordinates and connections are stored in an adjacency list. User inputs for departure and destination airports are collected through Tkinter's OptionMenu(), formatted as graph nodes, and used in the A* algorithm to find the shortest air route efficiently.

7.4 Snapshots





7.5 Performance Analysis

The A* algorithm efficiently finds the shortest air route by balancing path cost and heuristic estimation. It ensures optimal route selection with minimal computational overhead. While performance depends on the graph size and heuristic accuracy, optimizations such as priority queue implementation and admissible heuristics enhance speed and efficiency, ensuring real-time route computation for users.

8. BENEFITS

- **Optimized Route Planning**
  The system finds the most efficient air routes, minimizing travel time and fuel consumption while ensuring optimal connectivity.

- **Cost Reduction for Airlines**
  By calculating the shortest and most cost-effective routes, airlines can reduce operational expenses, improve efficiency, and enhance profitability.

- **Enhanced Decision-Making**
  Real-time route suggestions help pilots, air traffic controllers, and logistics teams make informed decisions for better flight management.

- **Improved Passenger Experience**
  Passengers benefit from reduced delays, smoother flight paths, and better scheduling, leading to a more efficient travel experience.

- **Scalability and Adaptability**
  The system can be expanded to integrate real-time weather data, airport congestion levels, and dynamic airspace changes for improved accuracy and efficiency.

9. CHALLENGES AND LIMITATIONS

- **Data Dependency**
  The system relies on accurate and up-to-date flight data, including airports, waypoints, and air traffic. Incomplete or outdated datasets may lead to suboptimal route predictions.

- **Handling Dynamic Airspace**
  The model does not account for real-time changes such as weather conditions, temporary flight restrictions, or air traffic congestion, which can affect route efficiency.

- **Computational Complexity**
  Finding the shortest or most efficient route using A* search requires significant processing power, especially when dealing with large-scale airspace maps.

- **Scalability Issues**
  Expanding the system to include global air routes may increase computational time, requiring optimizations like heuristic improvements or parallel processing.

- **User Input Limitations**
  Incorrect airport codes, missing waypoints, or unrealistic flight constraints in the Tkinter-based interface may result in route calculation failures or inaccurate results.

10. CONCLUSION

The Air Route Finder project efficiently implements the *A search algorithm** to determine the shortest and most optimal flight paths between airports. By integrating Tkinter for a user-friendly interface, it allows users to input departure and destination locations, visualize routes, and receive optimized flight paths. While the system effectively calculates routes, it has limitations such as reliance on static data and lack of real-time air traffic or weather considerations. Future enhancements, including real-time data integration and AI-driven optimizations, can improve accuracy and scalability, making it a more robust tool for aviation route planning.

10.1 Future Scope

- **Real-Time Flight Data Integration**
  Incorporating real-time air traffic and weather data will enhance route accuracy and reliability.

- **Multi-Criteria Optimization**
  Expanding the algorithm to consider factors like fuel efficiency, flight duration, and layovers for better route planning.

- **Scalability for Global Use**
  Enhancing the system to support international air routes and larger datasets for widespread application.

- **Enhanced User Interface**
  Improving the GUI with interactive maps and real-time flight tracking for better visualization.

- **Web and Mobile Deployment**
  Developing a cloud-based API or mobile application to make the Air Route Finder accessible anywhere.

11. REFERENCES

- Hart, P. E., Nilsson, N. J., & Raphael, B. (1968). A Formal Basis for the Heuristic Determination of Minimum Cost Paths. *IEEE Transactions on Systems Science and Cybernetics, 4(2), 100-107.*

- Russell, S., & Norvig, P. (2020). *Artificial Intelligence: A Modern Approach (4th ed.).* Pearson.

- Wikipedia Contributors. (2024). A* Search Algorithm. *Wikipedia, The Free Encyclopedia.*

- Python Software Foundation. (2024). NetworkX: Python Package for Complex Network Analysis.