INTER IIT TECH MEET 12.0

Mphasis Optimization (Team 41)

Passenger Re-accommodation for a planned Schedule Change

1 Abstract

Airlines routinely change their flight schedules for various reasons like seasonal demands, picking new routes, time changes needed based on daylight savings, changes to flight numbers, operating frequency, timings, etc. Many passengers will be impacted due to these schedule changes and they need to re-accommodate to the alternate flights. Airlines need a solution to analyze the impact on the passengers with their planned schedule changes and automatically identify suitable alternate flights for the impacted passengers.

2 Objective

Identify optimal/best alternate flight solutions for all the impacted passengers (impacted due to planned schedule change) based on the provided rule sets. Also, ensuring the validity of the solution with all the rule sets enforced. Rank the solutions based on various factors like time to reach the destination, impact on the purchased ancillary services, etc. as well as include Ranking of passengers on the solution based on re-accommodation priority based on factors like passenger type (unaccompanied minor, on duty employee, loyalty customer levels, paid class of service, etc). You can enforce the rule sets as part of optimization constraints or as pre or post-process calculation.

3 Input

3.1 Python libraries

- *pandas*: Used to read the input files using *read_ csv* method provided by this library.
- *python-dotenv*: Helps to environment variables like input CSV file paths which contain the data to process.
- *os*: Used to fetch the environment variables like input CSV file paths after loading them using *load_ dotenv* method.
- *datetime*: Used to calculate the difference between given dates and time.

The data type used to store the input data is a dictionary since the lookup and insertion of the data require a time complexity of $\mathbf{O(1)}$.

3.2 Extraction of airport data

A file containing the airport code and airport data of each airport is read and data is stored in a dictionary with the airport code as the key and its city as value and another dictionary with the airport code as the key and its latitude and longitude as value.

3.3 Parsing the data of flights and passengers

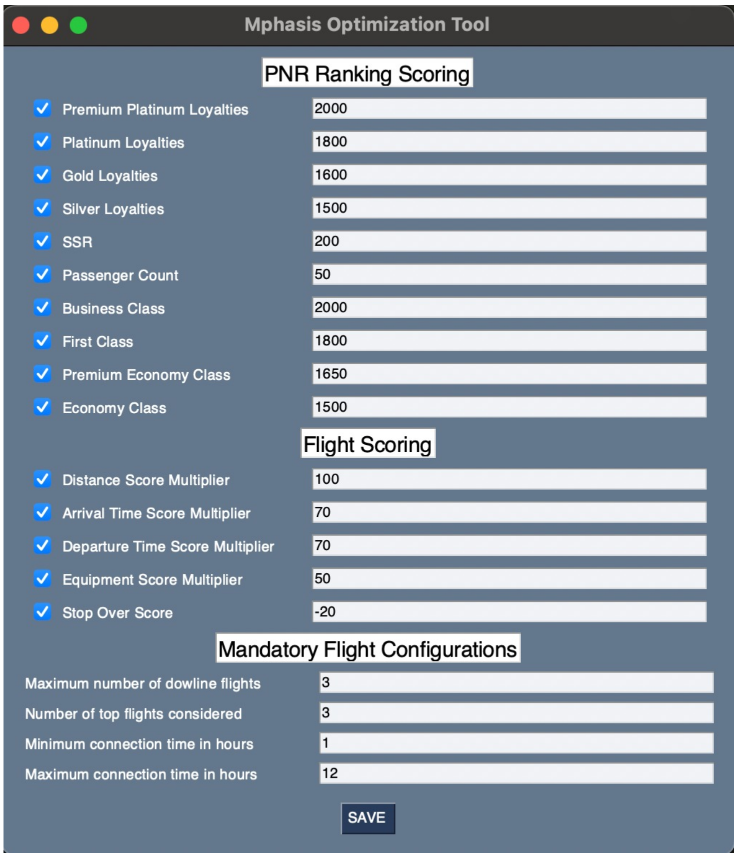
- The get_inventory function reads the inventory sheet and provides a dictionary with the inventory ID as the key, allowing us to retrieve flight data just by its inventory ID. Similarly, get_schedule reads the schedule sheet and returns a dictionary with the schedule ID as the key. get_date_inventory_list method returns a dictionary with a list of inventory IDs for flights on that day for each date.
- Passenger data, like flight data, is read and stored in a dictionary with the document ID (DOC_ID) as the key, whereas booking data is kept in a dictionary with the booking ID as the key.

3.4 Execution

All input parsing functions are run once in *__init__.py* file and the dictionaries generated are stored for to avoid re-reading data.

4 GUI for Score Flexibility

Built the window layout to edit the PNR and Flight ranking scores. It also helps to select number of Top alternate paths, Maximum number of connecting flights in alternate path, Min and Max connection time in hours.



PNR Ranking Scoring

✓	Premium Platinum Loyalties	2000
✓	Platinum Loyalties	1800
✓	Gold Loyalties	1600
✓	Silver Loyalties	1500
✓	SSR	200
✓	Passenger Count	50
✓	Business Class	2000
✓	First Class	1800
✓	Premium Economy Class	1650
✓	Economy Class	1500

Flight Scoring

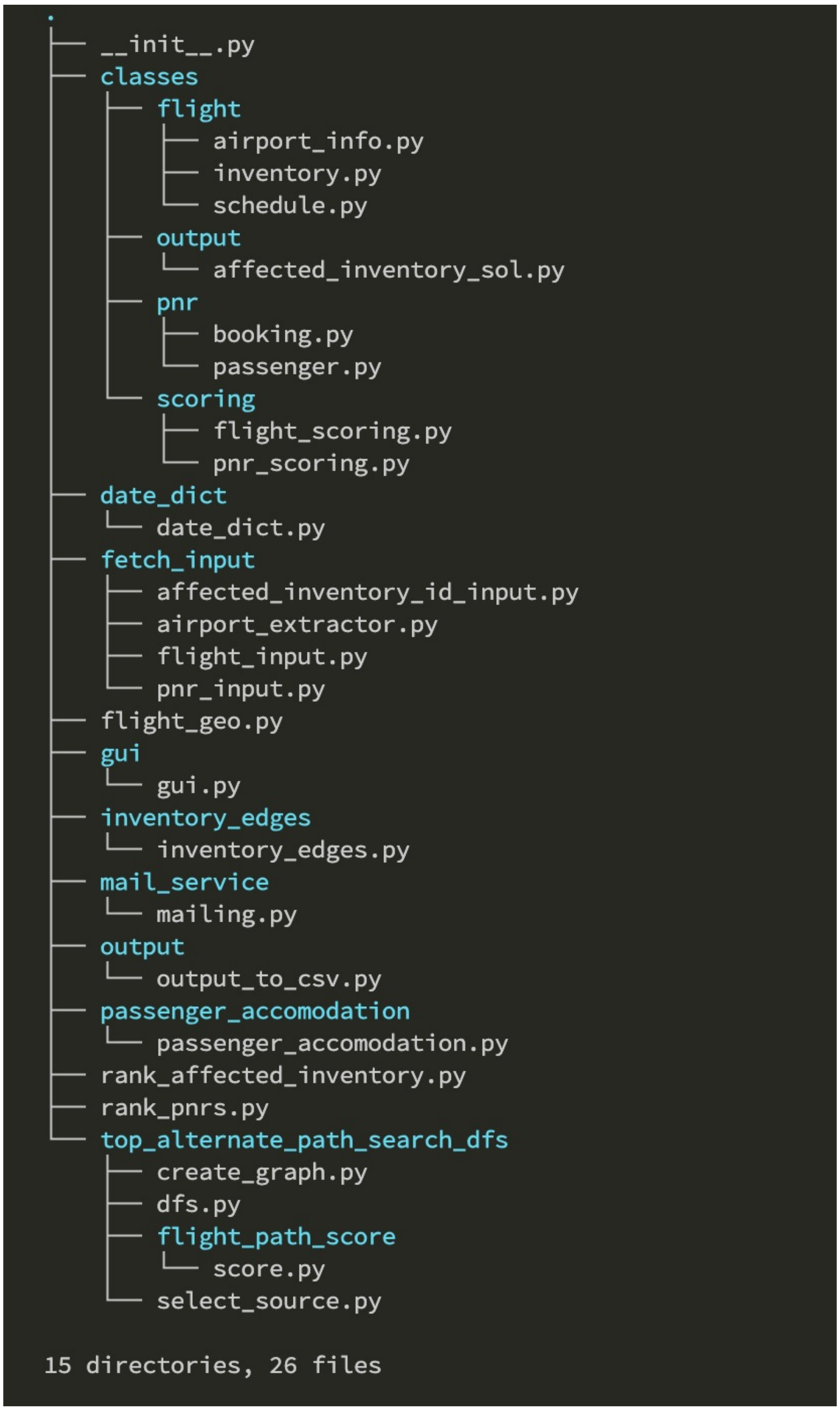
✓	Distance Score Multiplier	100
✓	Arrival Time Score Multiplier	70
✓	Departure Time Score Multiplier	70
✓	Equipment Score Multiplier	50
✓	Stop Over Score	-20

Mandatory Flight Configurations

Maximum number of dowline flights	3
Number of top flights considered	3
Minimum connection time in hours	1
Maximum connection time in hours	12

Figure 1: GUI for flexibility

5 Tree of Source code files



```
.
├── __init__.py
├── classes
│   ├── flight
│   │   ├── airport_info.py
│   │   ├── inventory.py
│   │   └── schedule.py
│   ├── output
│   │   └── affected_inventory_sol.py
│   ├── pnr
│   │   ├── booking.py
│   │   └── passenger.py
│   └── scoring
│       ├── flight_scoring.py
│       └── pnr_scoring.py
├── date_dict
│   └── date_dict.py
├── fetch_input
│   ├── affected_inventory_id_input.py
│   ├── airport_extractor.py
│   ├── flight_input.py
│   └── pnr_input.py
├── flight_geo.py
├── gui
│   └── gui.py
├── inventory_edges
│   └── inventory_edges.py
├── mail_service
│   └── mailing.py
├── output
│   └── output_to_csv.py
├── passenger_accomodation
│   └── passenger_accomodation.py
├── rank_affected_inventory.py
├── rank_pnrs.py
└── top_alternate_path_search_dfs
    ├── create_graph.py
    ├── dfs.py
    ├── flight_path_score
    │   └── score.py
    └── select_source.py

15 directories, 26 files
```

Figure 2: Tree of source code files

6 Date Dictionary

A system designed to access and retrieve data must be optimized in terms of time as much as possible. This obstacle is faced in the problem statement when we have to retrieve a list of possible alternate flights when we are given the data of a single affected flight.

- The date dictionary is a data structure in which the list of all flight Inventory IDs are stored, split into groups based on their date of departure. The dictionary can be compared to a calendar, where each date stores a list of the Inventory IDs of flights departing on that date. Additionally, the Inventory IDs are stored in the list sorted based on their time of departure on that particular day. This makes retrieving easier when we want to find a flight in a particular time slot.
- As the data is read from the Inventory file, a dictionary is populated with date-list pairs. The Inventory IDs are inserted into their corresponding date lists at the slot corresponding to their date of departure.
- Given an Inventory ID of an affected flight, retrieving the list of alternate flights is just a matter of retrieving the list of flights within 72 hours (3 days) of the affected flight. Sometimes, this can induce errors up to 24 hours as the lists are split based on dates and not time. To fix this, we use simple list-based filtering on the corner dates.

6.1 Time complexity Analysis of this data structure

The dictionary is represented as a hash table, and hence lookup for retrieving the list of flights on any particular day has a time complexity of $\mathbf{O(1)}$. Let $\mathbf{N}$ be the total number of inventories. After taking the input into the inventory dictionary, sorted based on the departure date and time in increasing order and then insert it into the date dictionary.

Inserting the inventory IDs into the date dictionary requires a time complexity of $\mathbf{O(N)}$. Sorting the inventory dictionary based on departure date-time requires a time complexity of $\mathbf{O(Nlog(N))}$

In total, creating the date dictionary can be thought of as an $\mathbf{O(Nlog(N))}$ operation.

7 Ranking PNRs for an Affected Inventory

- **Identification of Affected Bookings**
  - All PNR bookings sharing the same departure key (DEP_KEY) as the affected inventory are collected and stored in a set.
- **Scoring Based on PNR_Ranking Sheet**
  - Each identified booking's PNR is scored according to flight-specific metrics derived from the PNR_Ranking sheet.
  - Metrics include passenger count, class of service (COS_CD), and additional class-specific parameters.
- **Augmentation with SSR and Loyalty Scores**
  - Scores obtained from the PNR_Ranking sheet are complemented by the addition of Passenger Special Service Request (SSR) and Loyalty Program scores.

  - SSR scores are computed based on the nature and count of SSR codes associated with the passenger.
  - Loyalty scores are derived from the passenger's loyalty tier level, adding weightage to higher-tier members.

- **Ranking of Passengers**
  - A final scoring system, comprising PNR, SSR, and Loyalty scores, is applied to each affected passenger's booking.
  - Passengers are ranked in descending order based on their cumulative score.
  - Utilization of Python's sorted method results in time complexity of **O(Nlog(N))**, where **N** represents the number of affected passengers.

8 Init DFS

Before starting the init DFS algorithm, the affected inventory IDs are sorted according to the average passenger score (for passengers present in the airplane of the affected inventory ID). Then one by one affected inventory ID is given to init DFS. Given an affected inventory ID, the task of *init_ dfs* methods is:

- Get all the valid inventory IDs that have a departure time difference of at most 72 hours w.r.t to the affected inventory ID's departure time. This list can be fetched very efficiently with the help of date dictionary data type created before.
- Create a graph with airport codes as vertices and valid inventory IDs as directed edges. The vertex set can be formed by taking a union of the arrival and departure airports of each valid inventory in the edges.
- If the departure airport of the affected inventory has unfortunately no outgoing edges (no flights departing), then choose the alternate departure airport that is geographically closest to the former one.
- Once the vertex and edge set for the graph is created, call the **dfs** helper method which performs the dfs on this graph and returns all the valid paths.

8.1 *dfs*

- The stack parameters maintained by this algorithm are:
  - *depth*: Depth of the current airport code we are in, (number of inventories used in sequence to get to the current airport code).
  - *cur_path*: A list passed recursively as reference contains the list of inventory IDs used to get to the current airport code.
  - *all_paths*: A list which stores all the valid sequences of inventory IDs taken from source airport code to any airport code in the vertex set.
  - *current_ airport_ code*: The current airport code to be processed at the top of the stack

  - visited_airport_codes: A set of visited airport codes that helps in visiting only the airports that are not yet visited. (Helps in brute-forcing and generating all paths using the notion of backtracking).

- The constraints to be satisfied for choosing a particular inventory edge from the current airport code are:
  - The departure time difference should be at least 1 hour and at most 12 hours.
  - These values can be customized according to our needs before the process starts executing.

- Base cases:
  - Returns when the depth of the graph reaches some particular value. In real life, this value can be at most 5 because passengers do not like to have too many connecting flights during their journey. This value can be customized before the program executes.
  - If the current airport does not have any valid inventory departing (its adjacency list is empty), then return after appending the current path to the list of all paths.

8.2 *get_ top_ alternate_ paths*

- Arguments of this method:
  - Affected inventory ID
  - *all_ paths*: Output of the *dfs* method
  - Number of top alternate paths: #x of top paths to be returned after sorting it based on scores.

- This method acts as a driver code and uses a helper method *get_ alternate_flight_ path_ score* to calculate net score for a path based on different factors.

- It sorts the *all_ paths* after based on the scores and return top #x paths.

8.3 *get_ alternate_ flight_ path_ score*

- The main task of this method is to calculate an overall score for a path based on distance, time, and other constraints.

- Arguments of this method:
  - Affected inventory ID
  - Path: List of inventory IDs taken for a journey.

Factors considered by this algorithm for overall scoring:

- **Distance score**
  - Let x be the distance between the affected flight destination airport and the final airport reached by following this path in kilometers.

  - The distance score is calculated as follows:

$$DS = \frac{C_d}{1 + (\frac{x}{100})^2}$$

  - C_d is a constant value, which is the max distance score and can be customized before run time. DS is the distance score.
  - This is a continuous evaluating function and gives a more accurate score for filtering the top paths.

- **Arrival time score**
  - Let t_a be the difference in the arrival time of the affected flight and the arrival time of the solution path in hours.
  - The arrival time score is calculated as follows:

$$ATS = \frac{C_{t_a}}{1 + (\frac{t_a}{10})^2}$$

  - C_{t_a} is a constant value, which is the max arrival time score and can be customized before run time. ATS is the arrival time score.

- **Departure time score**
  - Let t_d be the difference in the departure time of the affected flight and the departure time of the solution path in hours.
  - The departure time score is calculated as follows:

$$DTS = \frac{C_{t_d}}{1 + (\frac{t_d}{10})^2}$$

  - C_{t_d} is a constant value, which is the max departure time score and can be customized before run time. DTS is the departure time score.

- **Equipment score**
  - Let p be the length of the solution path of the inventory IDs and let x be the number of inventory IDs in this path having the same aircraft type.
  - Equipment score is calculated as follows:

$$ES = \frac{x}{p} * C_e$$

  - C_e is a constant value, which is the equipment score and can be customized before run time. ES is the equipment score.

- **Stop over score** C_{so} score will be deducted from the net score if the solution path contains more connection flights.

9 Passenger Accommodation

The Passenger Accommodation Algorithm is a crucial component of the overall system, designed to efficiently assign seats to passengers in the alternate flight paths based on a predefined rule set specified by the company.

This algorithm ensures that the reallocation process adheres to the company's policies and maximizes the satisfaction of passengers by considering factors such as cabin class, seating preferences, and any other relevant guidelines. The rules are applied to connecting flights in the path.

For all top alternate paths, the algorithm tries to accommodate passengers of the affected flight. The path that accommodates a large amount of passengers that path is taken as a default solution and other paths are taken as alternate solutions for not accommodated passengers.

9.1 Input

For accommodating passengers

- Inventory ID of the affected flight
- A list of Affected Passengers ranked in descending order according to their cabin, preferences, etc.
- A tuple of the suggested alternate flight paths

9.2 Output

The algorithm returns the list of the recommended Accommodations of the passengers in the flight along with the inventory IDs of the flights.

10 Note on Quantum Computing

- **Grover's** algorithm is a quantum algorithm that finds a specific item in an unsorted database or list. Linear search has a worst-case time complexity of $\mathbf{O(N)}$, but grovers search improves it to $\mathbf{O(\sqrt{N})}$.
- Where can we possibly use this algorithm in our system?
  - To search for an inventory, schedule, passenger, or booking based on their IDs. Since, all these data are naturally unordered, using Grover's algorithm makes sense. But, here we have dictionary to store this which has inserting and access time complexity as $\mathbf{O(1)}$.
  - Grover's search can also be used to find the affected inventory based on inventory ID and may be used to fetch all the valid inventories for the graph. But instead, we use a more optimized data structure **date dictionary** for the same task which has a query time of $\mathbf{O(1)}$.
- The quantum query complexity of DFS is $\mathrm{O}(\mathbf{N}^{\frac{3}{2}} * log(\mathbf{N}))$ in the adjacency matrix model and $\mathrm{O}(\sqrt{\mathbf{N} * \mathbf{M}} * log(\mathbf{N})$ in the adjacency list model.
- Above, **N** denotes the number of vertices and **M** denotes the number of edges.

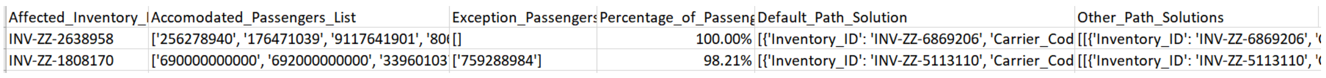
- Since the DFS used in the system has a fixed depth value which is usually at most 5, therefore the algorithm has an average time complexity based on depth $\mathbf{O(1)}$.

11 Solution Output

After finding alternate paths and accommodating passengers in the default solution. Accommodated passengers, not accommodated passengers, default solution and other solutions are stored in the AffectedInventorySolution object.

Once all affected flight passengers are accommodated and stored in a list of Affected-InventroySolution objects. We write solutions into the final_solutions csv file. The CSV file contains:

- affected inventory IDs
- corresponding accommodated passengers doc ids
- un-accommodated passengers doc ids
- percentage of accommodated passengers
- default solution
- other solutions



Affected_Inventory_I	Accomodated_Passengers_List	Exception_Passengers	Percentage_of_Passeng	Default_Path_Solution	Other_Path_Solutions
INV-ZZ-2638958	['256278940', '176471039', '9117641901', '80	[]	100.00%	[{'Inventory_ID': 'INV-ZZ-6869206', 'Carrier_Cod	[[{'Inventory_ID': 'INV-ZZ-6869206', '
INV-ZZ-1808170	['690000000000', '692000000000', '33960103	['759288984']	98.21%	[{'Inventory_ID': 'INV-ZZ-5113110', 'Carrier_Cod	[[{'Inventory_ID': 'INV-ZZ-5113110', '

Figure 3: output csv file example

12 Mailing the affected Passengers

In response to the need for a streamlined and efficient communication process with passengers affected by flight changes, our team has successfully developed a Mailing System. This system automates the process of sending personalized emails to affected passengers, providing them with details about their alternative flight paths.

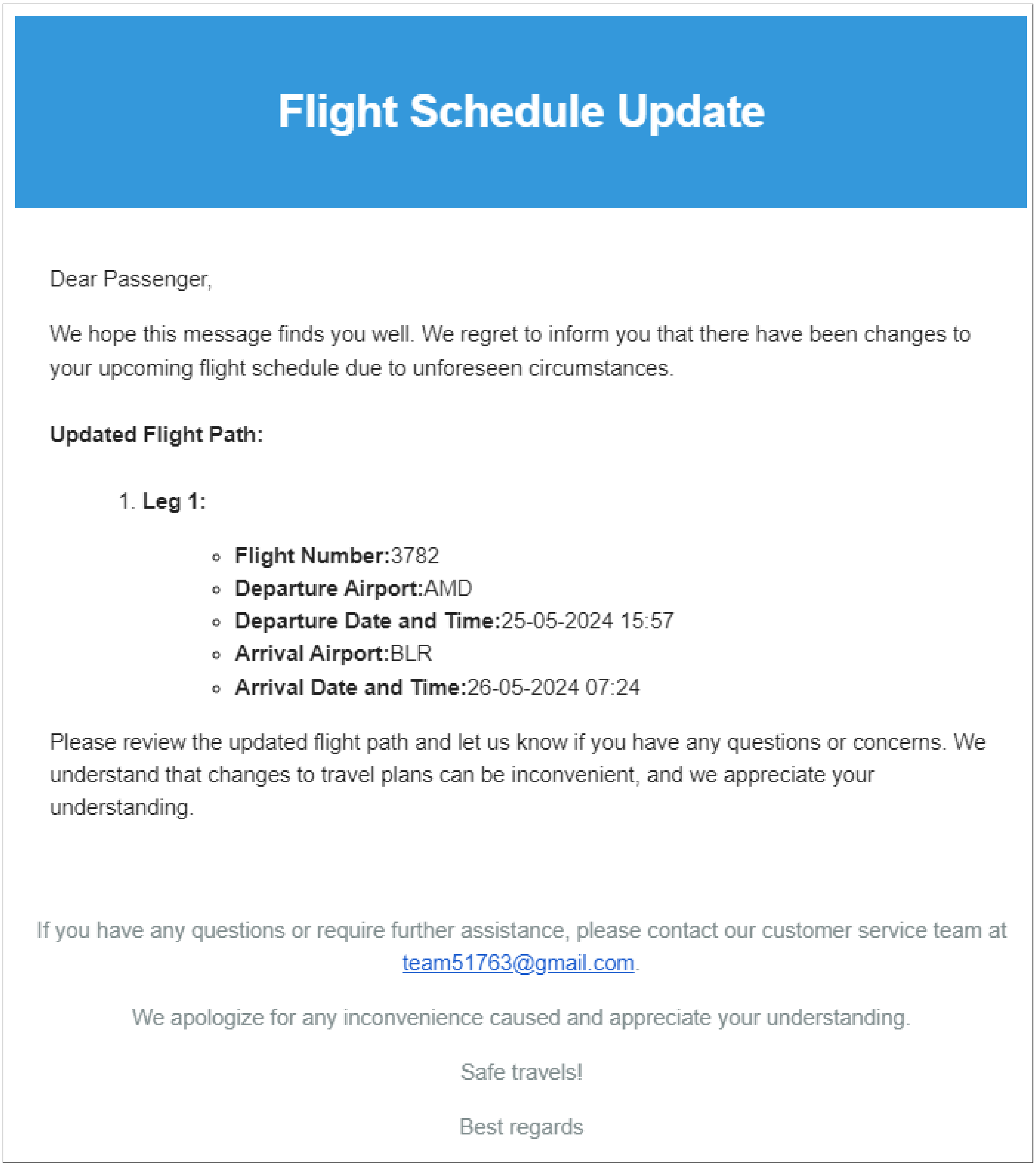
12.1 Modules used

- smtplib
- email for email message and MIMEText objects

12.2 Working

- After getting accommodated passengers and un-accommodated passengers for each inventory id affected, Mail service will send mails about default solution and other solutions respectively.

- Default Solution for accommodated passengers



Flight Schedule Update

Dear Passenger,

We hope this message finds you well. We regret to inform you that there have been changes to your upcoming flight schedule due to unforeseen circumstances.

Updated Flight Path:

1. **Leg 1:**
   - **Flight Number:**3782
   - **Departure Airport:**AMD
   - **Departure Date and Time:**25-05-2024 15:57
   - **Arrival Airport:**BLR
   - **Arrival Date and Time:**26-05-2024 07:24

Please review the updated flight path and let us know if you have any questions or concerns. We understand that changes to travel plans can be inconvenient, and we appreciate your understanding.

If you have any questions or require further assistance, please contact our customer service team at team51763@gmail.com.

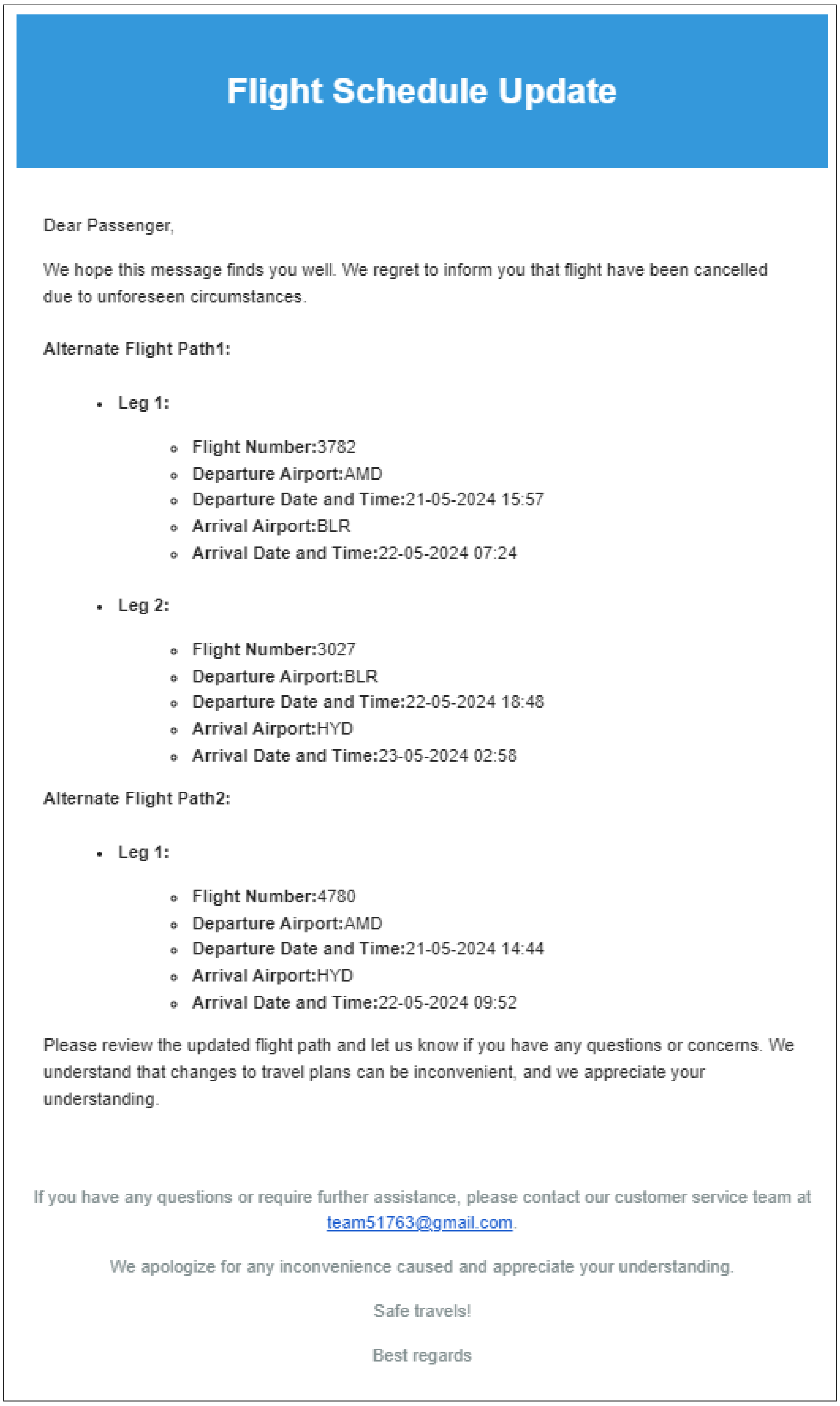
We apologize for any inconvenience caused and appreciate your understanding.

Safe travels!

Best regards

Figure 4: Default solution mail

- Other Solutions for Un-accommodated passengers



Flight Schedule Update

Dear Passenger,

We hope this message finds you well. We regret to inform you that flight have been cancelled due to unforeseen circumstances.

Alternate Flight Path1:

- **Leg 1:**
  - **Flight Number:**3782
  - **Departure Airport:**AMD
  - **Departure Date and Time:**21-05-2024 15:57
  - **Arrival Airport:**BLR
  - **Arrival Date and Time:**22-05-2024 07:24
- **Leg 2:**
  - **Flight Number:**3027
  - **Departure Airport:**BLR
  - **Departure Date and Time:**22-05-2024 18:48
  - **Arrival Airport:**HYD
  - **Arrival Date and Time:**23-05-2024 02:58

Alternate Flight Path2:

- **Leg 1:**
  - **Flight Number:**4780
  - **Departure Airport:**AMD
  - **Departure Date and Time:**21-05-2024 14:44
  - **Arrival Airport:**HYD
  - **Arrival Date and Time:**22-05-2024 09:52

Please review the updated flight path and let us know if you have any questions or concerns. We understand that changes to travel plans can be inconvenient, and we appreciate your understanding.

If you have any questions or require further assistance, please contact our customer service team at team51763@gmail.com.

We apologize for any inconvenience caused and appreciate your understanding.

Safe travels!

Best regards

Figure 5: Other solutions mail