# Quantum Computing Challenge

Team ID: 80

December'23


# Passenger Re-accommodation for a planned schedule change

# Documentation

# Contents

# §1 User Manual and Setup

## §1.1 System Requirements

The solution implemented by our team utilizes the resources provided by DWave's Leap Hybrid Solver. The requirements for the system to run the Quantum Solution are:

1. python3.11
2. dwave-ocean-sdk
3. pandas
4. csv
5. DWave Leap Solver API Token

The requirements for the system to deploy and use the GUI are:

1. flask
2. csv

The requirements for the system to run the classical solution are:

1. ortools
2. csv
3. pandas

## §1.2 Demonstration

For ease of use from the user's perspective, we have divided the process of Passenger Re-accommodation for a planned schedule change into a total of 9 steps. The basic GUI from the user end is as follows:

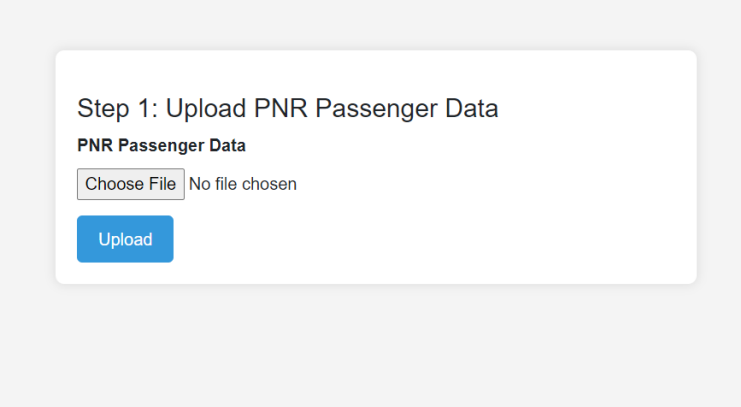



Figure 1

### §1.2.1 Submitting inputs to the GUI

A total of 4 files are needed to be submitted from the user - the PNR Passenger mapping, the PNR to flight booking data, the running flight schedule and the inventory data for the flights. From steps 1 to 4 the GUI will ask for the data in csv format.
Upload the file from the user's device.

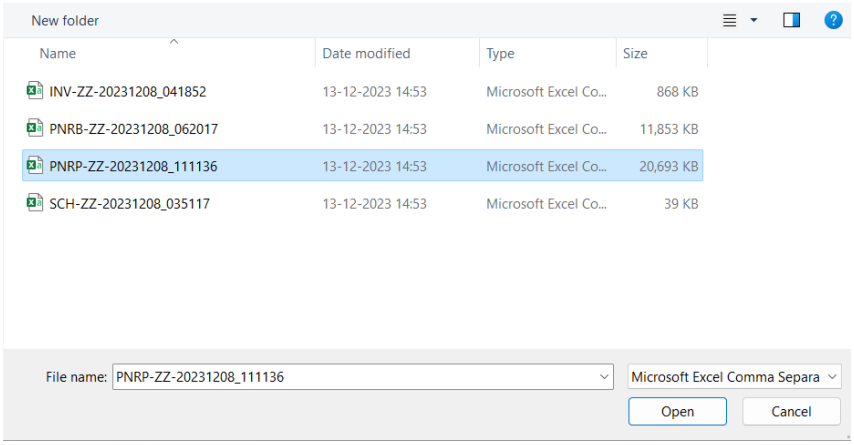



Figure 2

Click on Upload to upload the PNR Passenger mapping file to the server. Make sure that the data is in the IATA data format before proceeding. Similarly, upload the

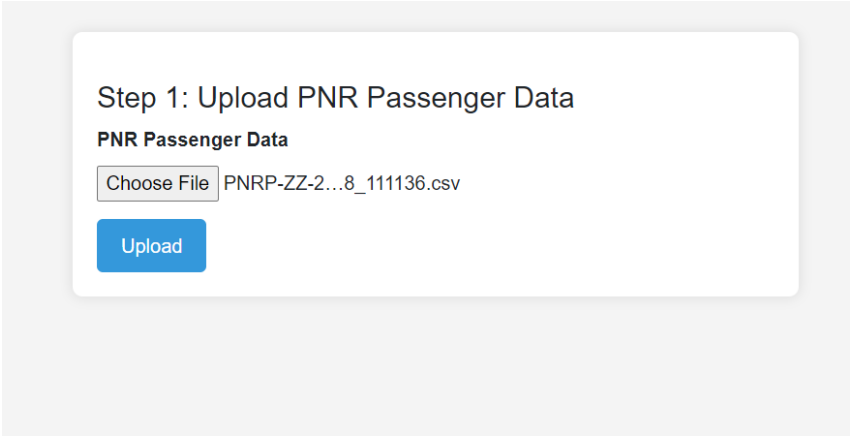



Figure 3

PNR Bookings data, Flight Schedule and Flight Inventory csv files.

### §1.2.2 Cancelling Flights

In step 5, enter the departure keys of the flights you wish to cancel in comma separated format.

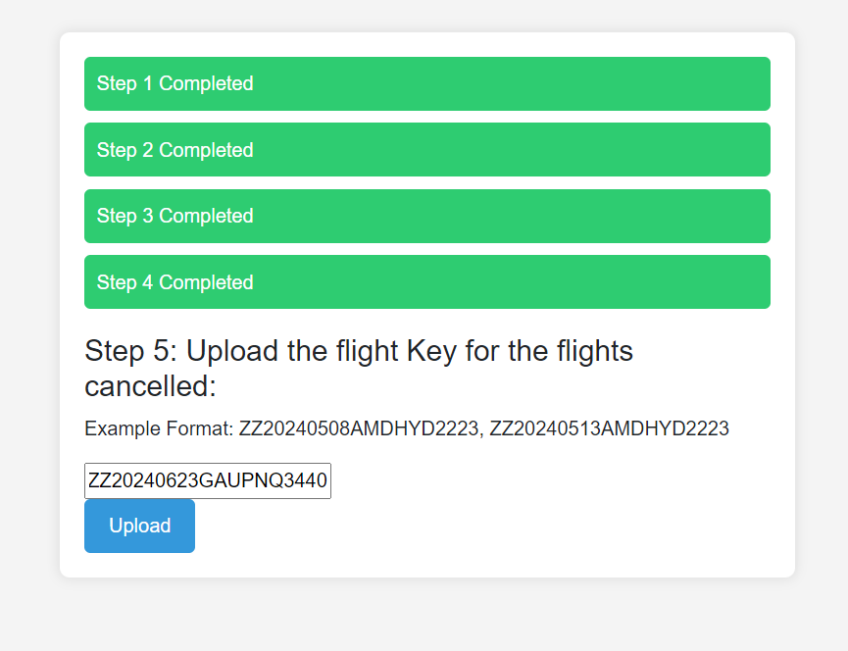



Figure 4

### §1.2.3 Enabling rules and weights

In step 6, choose the rules you wish to enable according to the business policy, and choose weights. Defaults are already pre-filled.

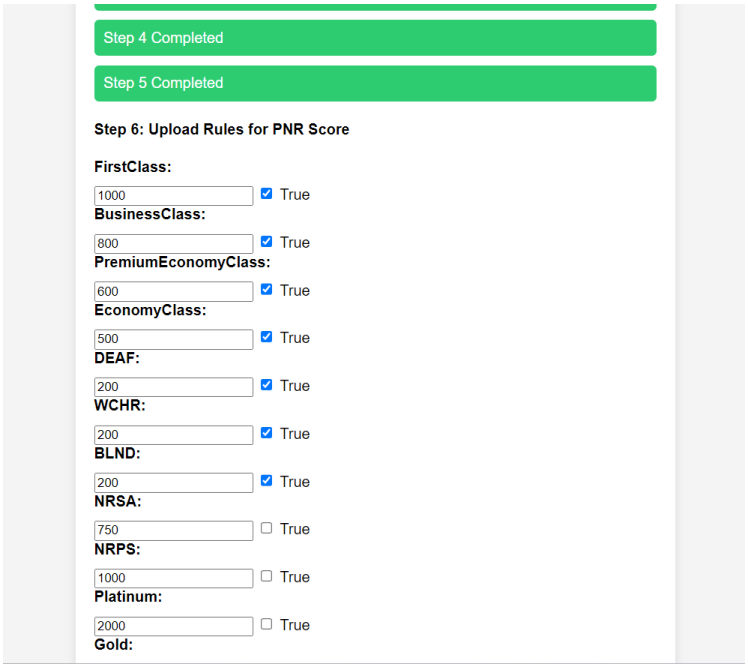



Figure 5

In step 7 you can choose the weights regarding the viable replacement routes which will be generated by the solution.

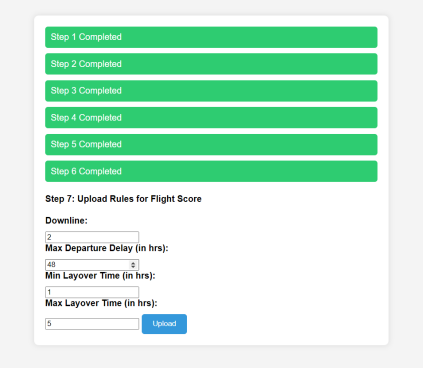



Figure 6

### §1.2.4 Connecting the GUI to DWave's Solver

For step 8, you need to input the DWave Solver API Token from DWave's official website. Copy your Solver API Key and paste here.

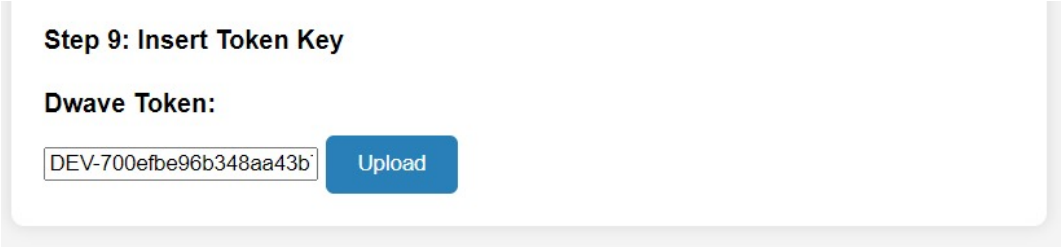
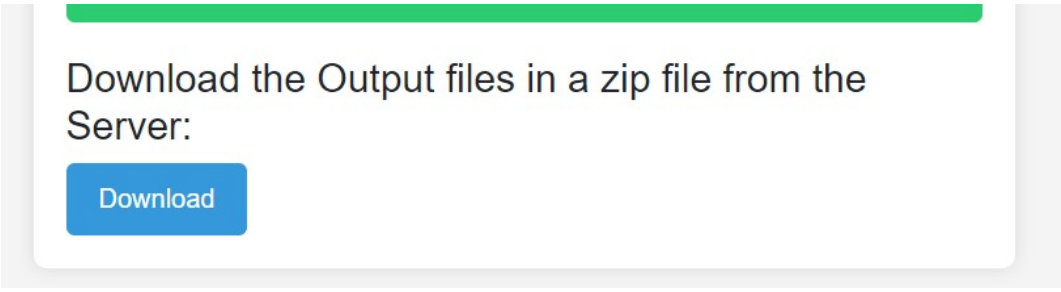



Figure 7

### §1.2.5 Retrieving Outputs

After the solution files have been generated on the backend, you get a prompt to download the files. The files are downloaded as 'output.zip' and contains $2 * N$ files per cancelled flight corresponding to the default and exception allotments for each generated solution. The 10 best solutions are generated by default. Files are also generated for overbooked flights which are detected by the program automatically.



## §2 Abstract

The aviation industry is an extremely dynamic sector, with the airlines trying to constantly optimize their costs and increase profits by making schedule changes to their published schedule. Subsequently, passengers are affected due to these changes, and currently, airlines need 2 to 3 days of manual work to confirm the changed schedule. Our solution proposes a simple and efficient solution to the problem, reducing the required time from 2 to 3 days to a few seconds.
The problem can effectively be modeled in terms of several binary variables and weighted constraints on them expressed in terms of a quadratic model. There exist classical methods to solve quadratic problems with libraries such as CVXOPT, etc., which can be used to solve quadratic programming problems. The problem with the existing classical methods is that the time taken to solve a system of equations increases very quickly, making it infeasible to use them for the problem at hand because of the exceptionally large search space. Any change to even a single flight affects around 200 passengers, and there are multiple ways of re-accommodating each passenger, which results in a large search space, and, consequently a large number of variables and constraints.
We decided to exploit the properties of quantum mechanics to overcome this barrier of an incredibly large search space and use Quantum Annealing to find a viable solution to the described problem. We see an immediate performance improvement, with better time scaling in the quantum solution and an ability to handle large amounts of changes.

## §3 Introduction and Problem Formulation

In the aviation industry, flight schedules undergo frequent changes due to seasonal demands and operational adjustments. This poses challenges for airline companies to re-accommodate passengers affected by flight cancellations.
Airline companies aim to map most affected passengers to another flight option with minimal time delays and inter-class mappings.
Our Flight Scheduling and Passenger Assignment Optimization system utilizes the strength of Quantum Annealing to provide a comprehensive solution. It identifies and optimally re-accommodates impacted flights and passengers, considering factors such as travel time and ancillary service impact. The system prioritizes passengers based on factors like passenger type and loyalty status, offering flexibility in rule adjustments. It generates insightful reports for airline decision-making, along with detailed re-accommodation files. Ground rules, including expiration dates, ensure a structured and time-sensitive schedule adjustment process. Above all, the solution is truly flexible to the airline's business needs, and with extremely minor changes to the GUI / file setup, can be modified to include or exclude any of the airline's interests.

The problem formulation is completely independent of the rule set, and it simply takes in the data generated in the data pre-processing step, where all the airline's business rules can be accommodated easily. Additionally, there are constant factors in the main solution, which can be changed as described below in the Methodology section, making our entire solution extremely versatile for any airline to adopt, rather than it being tailored to the needs of any singular airline.

# §4 Data Preprocessing

## §4.1 Description

The provided data contains 4 individual datasets corresponding to the Schedule of flights, the individual inventory of available flights, the mapping of PNRs with their booked flights, and finally, the data of each passenger associated with the PNR.
There are 90 distinct flights, totaling to about 2000 flights throughout the dataset.
The total number of passengers is about 150000.
Out of all the data, we identified the following fields as relevant to our solution and came up with the following data model:

1. rules.csv: This is the file that will be generated by the GUI interface where the airline will input its business rules with respect to the score it wishes to allocate to each condition associated with a particular PNR as well as the force kickout condition.

2. pnr_score.csv: This file will be generated by the GUI using the PNRP and PNRB csv files uploaded to the interface. It contains 5 columns, the unique ID of a PNR, flight pairing, the PNR itself, the departure key of the flight that was booked, the number of passengers booked under this PNR, the score calculated for the PNR using the above file, and whether or not this PNR can be force kicked out.

3. flights.csv: The GUI will generate this file using the SCH and INV csv files uploaded to the interface. It contains 8 columns, the UNIQUE ID assigned to a departure key, class pair, the departure key, the departure airport, the arrival airport, the class, the total capacity of the class on the flight, the departure time in epoch and the arrival time in epoch

## §4.2 Passenger Name Record (PNR) Ranking

The PNR ranking is done according to the score calculated from the business rules provided by the airline. The airline can choose the points they wish to allocate to each category and add or remove categories as they wish to with a little modification to the GUI. The PNR sore is the sum of scores of all individual passengers linked to that PNR. However, if the airline wishes to change that score, it can be easily done solely via modifying the GUI code, and our solution remains unaffected.

## §4.3 Flights

For the flights, all the data that we need is present in the INV database. We split each flight into 4 different options to make it easier to deal with the different classes present on every flight. Thus the capacity column contains solely the capacity of the class that is referred to by the respective option.

The integrity of the data is maintained still by the fact that we store the departure key as well, which links all 4 options together.

# §5 Methodology

We present a quantum-based Flight Re-Accommodation optimization approach to address the challenges posed by schedule modifications. Leveraging the Leap Hybrid CQM Solver by D-Wave, our methodology aims to efficiently re-accommodate passengers affected by schedule changes, considering various constraints and objectives.

## §5.1 General Goals

1. Minimize number of un-accommodated passengers.
2. Minimize mean arrival time delay.
3. Prioritise PNRs with higher score to be assigned better flights (lesser time delay, better class, etc.).
4. Prioritise replacement journeys with fewer legs over those with more legs/layovers.
5. Maximize the percentage of PNRs under the default solution.

## §5.2 Some Definitions

A *Flight Option* is defined to be the union of any given flight along with a specific class (First Class, Business Class, Premium Economy, Economy Class). Briefly, $f : A \to B$ is a flight option departing from $A$ and arriving at $B$, and is associated with a specific class. The set of all flight options in the network is denoted by $F$.

A *Route* with $n$ legs departing from $A$ and arriving at $B$, $r : A \to B$ is defined to be a sequence of $n$ flight options $f_i$ where $f_1$ departs from $A$, $f_n$ arrives at $B$, and $f_i$ departs from the airport where $f_{i-1}$ arrives.

An *Impacted PNR* is defined to be any PNR whose original flight has been cancelled, or has been kicked out of an overbooked plane.

We represent the classes numerically for the purposes of calculation, with first class as 1, business class as 2, premium economy as 3 and economy class as 4.

## §5.3 Viable Routes and Mapping

We explore viable routes for each impacted flight from departure to arrival locations. A graph-based representation facilitates route discovery, utilizing recursive calls to identify multiple routes between airports. This comprehensive exploration enhances the solution space, enabling more diverse and efficient re-accommodation alternatives.

We find a set of viable replacement routes for every impacted flight using a DFS-based algorithm. This is a simple algorithm that starts DFS at the departure airport and tries to find routes ending at the arrival airport with a few constraints:

1. Maximum layover time ($T_L$ in seconds).
2. Minimum layover time ($T_{mL}$ in seconds).
3. Maximum departure delay ($T_D$ in seconds).

4. Maximum number of legs in the replacement route ($L_m$).

Subject to the above constraints, for each impacted flight $f : A \to B$ we generate a set of routes $r : A \to B$. We denote this set of viable routes by a set $R(f)$. The array $V$ is defined to be of the size of the set $R$, with element $V[i]$ of the array representing the $i$'th viable route.

## §5.4 Quantum-Based Binary Optimization

We formulate our problem in terms of a **Constrained Quadratic Model (CQM)** as explained in the following sections. The constraints, objectives and decision variables are encoded into binary variables for quantum processing. The Leap Hybrid CQM Solver is then used to arrive at various optimal solutions of the problem.

## §5.5 Formulation of CQM

For each impacted flight $k$, we define the following:

1. Matrix $X[k]$, a binary matrix signifying the assignment of PNRs to viable routes.
2. Matrix $M[k]$, a real matrix signifying the cost of assignment of every possible PNR-route pairing.
3. An array $D[k]$, signifying which of the routes is the default solution.
4. An array $Y[k]$, signifying the number of passengers re-accommodated on a specific viable route.

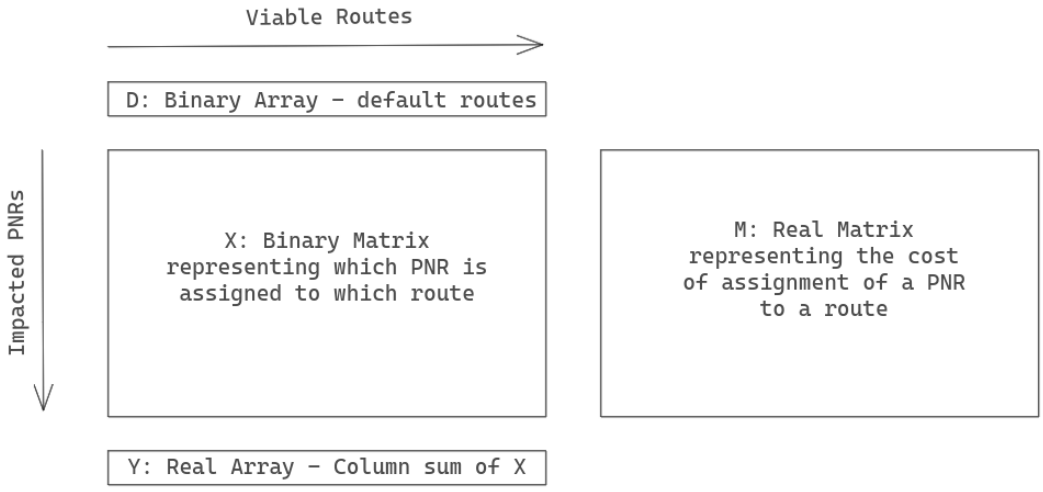



Figure 8: A visualization of the data structures created for every impacted flight (overbooked, cancelled etc.)

A detailed definition of each of the above is as follows:

**Matrix** $X$ $(X[k])$: This matrix signifies the assignment of passengers to viable routes. It is of size $|P[k]| \times (|V[k]| + 1)$, where $|P[k]|$ is the number of impacted passengers originally in the $k^{th}$ flight, and $|V[k]|$ is the number of viable routes for this flight. Each element $X[k][i][j]$ is a binary decision variable indicating whether $i^{th}$ impacted passenger of $k^{th}$ flight $P[k][i]$ is re-accommodated on route $V[k][j]$. The last "dummy" column corresponds to a passenger not being re-accommodated.

**Matrix** $M$ $(M[k])$: This matrix encapsulates the scores and weights associated with re-accommodating each passenger on a specific route. It is also of size $|P[k]| \times (|V[k]| + 1)$, with $M[k][i][j]$ representing the score associated with re-accommodating passenger $P[k][i]$ on route $V[k][j]$.

Let

- $m = |P[k]|$, $n = |V[k]| + 1$.
- $t_o$ be the arrival time of the original flight $k$, $t_i$ be the arrival time of route $V[i]$.
- $O_i$ be the original class allotted to PNR $i$.
- $C_i$ be the average of the classes of the flight options present in route $i$.
- $S_i$ be the net score of the PNR $i$.
- $l_i$ be the number of legs in route $i$.
- $\beta$ be a constant, termed as **Class Constant**.
- $\gamma$ be a constant, termed as **Multi Leg Constant**.
- $CNT[i]$ represents the number of passengers in PNR $i$.

Then, $M[k]$ is an $m \times n$ real matrix, which is populated as follows:

$$M[k][i][j] = \begin{cases} S_i \times (t_j - t_o) \times \left(1 + \frac{C_j}{\beta O_i}\right) \times \left(1 + \frac{l_j}{\gamma}\right) \times CNT[i] & j < n \\ S_i \times \infty & j = n \end{cases}$$

where $\infty$ is a sufficiently large finite number. The logic behind defining $M$ like this is to minimize the time delay, prioritize the PNR keeping the same class or being upgraded, reducing the number of legs in the replacement route, and to ensure that every passenger is accommodated if possible.

**Array** $Y[k]$ It is an array of size $n$, storing the number of PNRs reassigned to a specific route. It is defined as follows:

```
for all i, j, k:
    Y[k][j] += (X[k][i][j] * CNT[i])
```

where $CNT[i]$ is the passenger count of PNR $i$.

**Array** $D[k]$ This is a binary array of size $n-1$, with $D[k][i] = 1$ if the $i$'th route is the default solution for the $k$'th impacted flight, else it is zero.

Finally, we also define a mapping $Z$ with size $|F|$. It stores the number of passengers reallocated in a specific flight option $f$. $Z$ is populated as follows:

```
for all i, j, k:
    for all f in R(j):
        Z[f] += (X[k][i][j] * CNT[i])
```

## §5.6 Constraints

1. Constraint 1: One PNR is to be accommodated in one route only.

$$\sum_j X[k][i][j] = 1 \ \forall \ k, i$$

2. Constraint 2: The number of passengers accommodated on a flight option $f$ must not exceed its capacity to carry $C_f$.

$$Z[f] \leq C_f \ \forall \ f \in F$$

3. Constraint 3: There is exactly one default solution for each impacted flight.

$$\sum_i D[k][i] = 1 \ \forall \ k$$

## §5.7 Objective

We formulate our objective so as to achieve the goals mentioned in section 5.1. Let $S$ be our objective which will be **minimized**, then we have defined it as follows:

```
for all i, j, k:
    S += X[k][i][j] * M[k][i][j]

for all j, k:
    S -= D[k][j] * Y[k][j] * alpha
```

where $\alpha$ is a constant termed as **Default Constant**. The objective defined above is **quadratic** as both $D$ and $Y$ are decision variables.

The first part of the objective is motivated by the **Rearrangement Inequality**, as we want passengers with higher scores to be assigned better routes (with lower value in the $M$ matrix).

The second part of the objective is motivated by goal (5), but as goal (5) can conflict with goals (2), (3), it's weightage is defined by $\alpha$ which can be tuned as required.

The solution with the minimized objective is the most optimal solution.

## §5.8 Summary of CQM Formulation

Minimizing the objective successfully deals with all the goals mentioned in section 5.1. As certain goals can be conflicting, and certain decisions depend on the business policy of the airlines, we have summarized the constants below in Table 1 which can be tuned as required.

| Symbol | Range | Name | Description |
|---|---|---|---|
| $\alpha$ | $10^6 - 10^{15}$ | Default Constant | Higher value prioritizes more % of PNRs under default solution |
| $\beta$ | $1 - 100$ | Class Constant | Higher value means less of a priority to maintaining /upgrading class |
| $\gamma$ | $1 - 100$ | Multi Leg Constant | Higher value means multiple-leg journeys are penalised less |
| $T_{mL}$ | $3600 - 72000$ | Min Layover Time | Value in seconds |
| $T_L$ | $3600 - 72000$ | Max Layover Time | Value in seconds |
| $T_D$ | $3600 - 259200$ | Max Departure Delay | Value in seconds |
| $L_m$ | $1 - 5$ | Max No. of Legs | Number of legs in replacement route |
| $\infty$ | $10^9 - 10^{18}$ | Infinity | Sufficiently large finite value |

Table 1: Input Constants and their recommended ranges

## §5.9 Dealing with overbooked flights

Overbooked flights can be dealt with in a greedy manner - kick out those passengers with the lowest scores which can be force kicked out based on the given rule set. Then, we treat these kicked out passengers in the same way we treat PNRs whose flight has been cancelled; by creating a matrix and solving the CQM.

In our implementation we have directly added all PNRs of an overbooked flight to the matrix created for this impacted flight. Overbooked flights are **automatically detected** and appropriate re-accommodation of PNRs is performed by the program.

## §5.10 Dealing with different kinds of schedule changes

Every schedule change as mentioned in the problem statement can essentially be modeled as the cancellation of some flights and addition of some other flights. In our solution, if the new flights are added to the SCH and INV files, the solution will automatically consider allocating passengers to the new flights if feasible.

Consider the following scenarios

1. Schedule duration change(extend schedule / or compress schedule)
   In the case of schedule compression, several flights that were supposed to take

off during the compressed duration are canceled and flights with new take-off times are added to the schedule. If the entries for the new flights are added to SCH and INV files, our solution will accommodate affected passengers, provided the new flights are viable under the airline's business rules.

2. Schedule deletion
   This corresponds to straightforward cancelation of flights, and no additional steps are required to process these changes.

3. Frequency of the schedule change
   Again, this can be modeled as the cancelation of flights from some days and adding new flights to another day. As long as these changes are properly reflected in the input files and fields, our solution will do that.

4. Aircraft Type change (which will change the Capacity of the available seat in the flight)
   With this change, if the flight does not become overbooked, no passengers need to be moved or accommodated. However, it might result in the flight becoming overbooked, and some passengers may become overbooked. The change needs to be reflected in the SCH and INV files, and we take care of overbooking as explained in the above section.

5. Possible Schedule Time change
   This means that the flight is canceled at the original time, and then new flights are added at the new time. Suppose this new time is within acceptable limits for the airline's business rules. In that case, passengers will mostly be allocated to this new flight, as this flight is empty, and the new flights allocated to the passengers on the canceled flight will be allocated flights according to the prioritization criteria defined above.

## §6 Results

After the airline chooses various business rules, sets various weights and inputs the various files as required by the GUI, the algorithm will generate many various solutions. This gives the airline a variety of options to choose from. The most optimal solution from the perspective of the algorithm is that with the least objective, but the airline is free to choose from any of the given options.

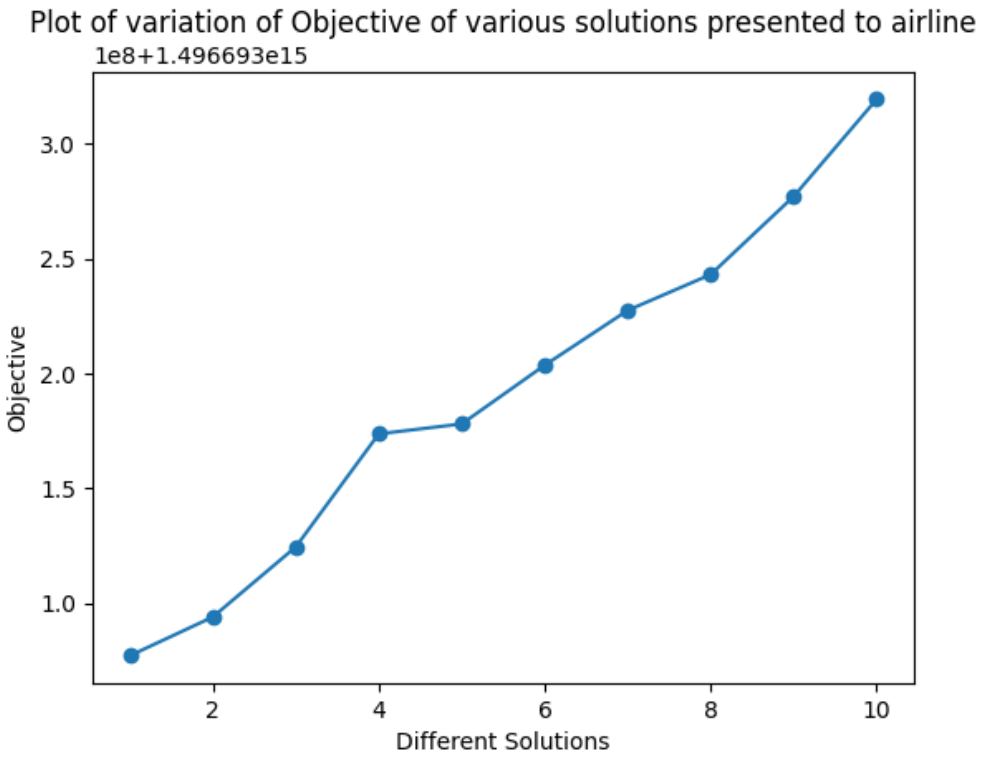



Figure 9

The output files are generated in the form of two csv files for each impacted flight; one for the default solution case, and the other for the exception solution case.

For example, we cancel two flights in the given dataset, those with departure keys *ZZ20240505AMDHYD2223* and *ZZ20240623GAUPNQ3440*; the solution files generated will look like those in the following image. The files with prefix 0 correspond to the most optimal solution, with prefix 1 correspond to the second most optimal solution, and so on. The number of alternative solutions required can be configured easily.

The files describe the PNR re-accommodation. Some statistics about the most optimal solution (with prefix 0) are as follows:

1. **362/362** impacted passengers are re-accommodated
2. Mean Arrival Time Delay: 29356.9 seconds which is approximately 8 hours
3. Objective = 102551032000.00

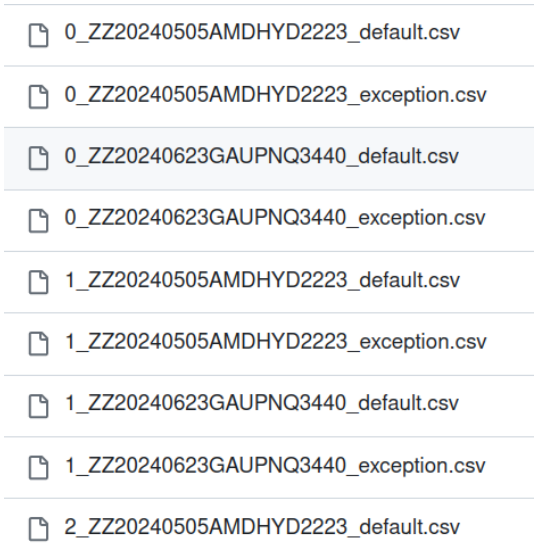



Figure 10: Output files and their file names

4. For the flight from AMD to HYD, 23/77 PNRs are under the default solution.
5. For the flight from GAU to PNQ, 19/31 PNRs are under the default solution.

Overall, we see that provided that appropriate constants are chosen, the algorithm performs very well. Mean time delays are kept to a minimum as far as possible, and all passengers are re-accommodated. This is also observed with various other flight cancellations and schedule changes.

In some runs, the selection of canceled flights resulted in some passengers not being allocated any flight. In this case, it was observed that these were the passengers with the lowest PNR scores, and no other seats could be allocated to these passengers because every seat on every possible route satisfying the assumed variable constants was occupied. These passengers were later accommodated when the variable constants were relaxed to increase the number of routes available.

Also, the given dataset has *no* overbooked flights. When run on a dataset containing overbooked flights, our solution **automatically identifies overbooked flights and re-accomodates passengers optimally**. The output files generated are similar to the ones above; 2 files are generated per solution per overbooked flight, one for the default and one for the exception case.

```
NQZY82,1,"['ZZ20240527AMDHYD2223', 'EC']"
DJAS36,2,"['ZZ20240526AMDMAA4012', 'EC']","['ZZ20240527MAAHYD4704', 'PC']"
RYQN10,1,"['ZZ20240527AMDHYD2223', 'BC']"
ECGC84,1,"['ZZ20240527AMDHYD2223', 'FC']"
```

Figure 11: Snippet of an example output file of an overbooked flight.

As shown above, each line of the output files corresponds to the assignment of a PNR to a route; a route can consist of one or more legs as can be seen above. Each leg will consist of a departure key along with the class assigned in that flight.

# §7 Comparison with Non Quantum Solutions

The classical version of our solution uses Google's OR Tools to solve the constrained problem. A major limitation of the OR Tools library is that it supports only linear programming problems and cannot handle quadratic programming problems.

For our solution, this meant that we had to forego Default Solution prioritization, as that was the only part of the solution that is quadratic in nature, with the rest of the solution being linear in nature.

Despite this relaxation in the number of variables and constraints, the linear version is much slower than the quadratic version.

In a stress test where we kept the number of canceled flights equal to 2 only but varied the search space size $\sum_{i=1}^{k} n_i \times m_i$, we observed a generally increasing trend of the measured **process time**, with the solution taking approximately 5ms for small search spaces and going up to 7200s for a search space of approximately 1500.

Additionally, as is visible in the graph, the time taken by the solver for any search space is highly variable and based on how accurate its initial few steps are, with some searches for large search spaces taking as less as 5ms and others taking 7200s.

Rerunning the same test cases gives highly variable times as well.

On the other hand, for quantum solutions, the solution is much more stable with

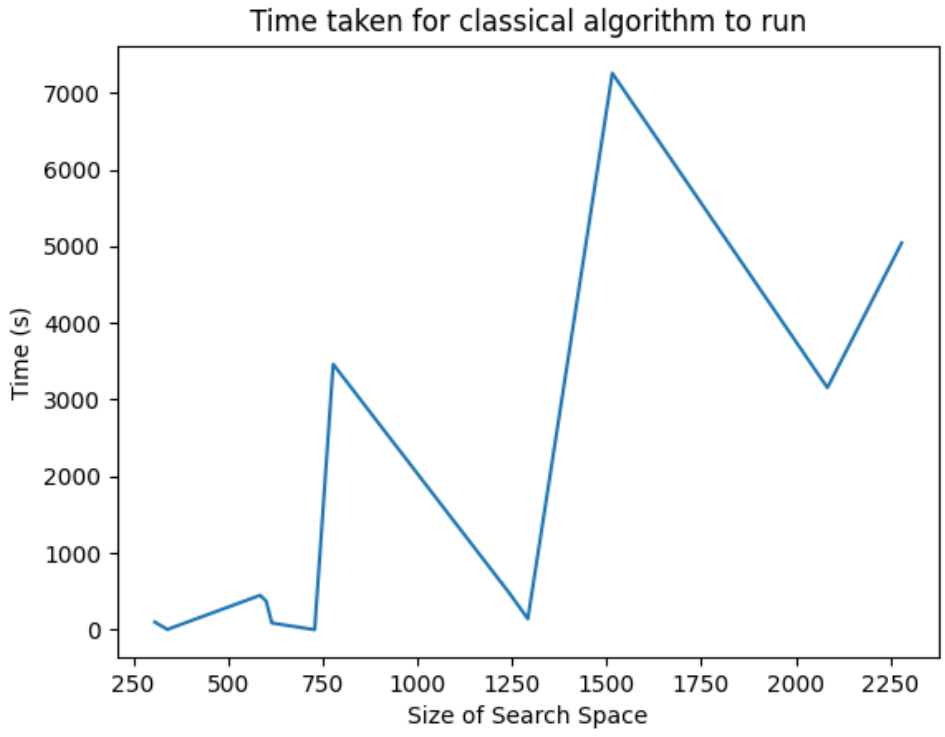



Figure 12: Run on 11th Gen Intel® Core™ i7-11800H, 16 threads, 16GB RAM. Process time is measured.

similar solutions each time.

Note that despite the **quantum algorithm being run on a search space 50**

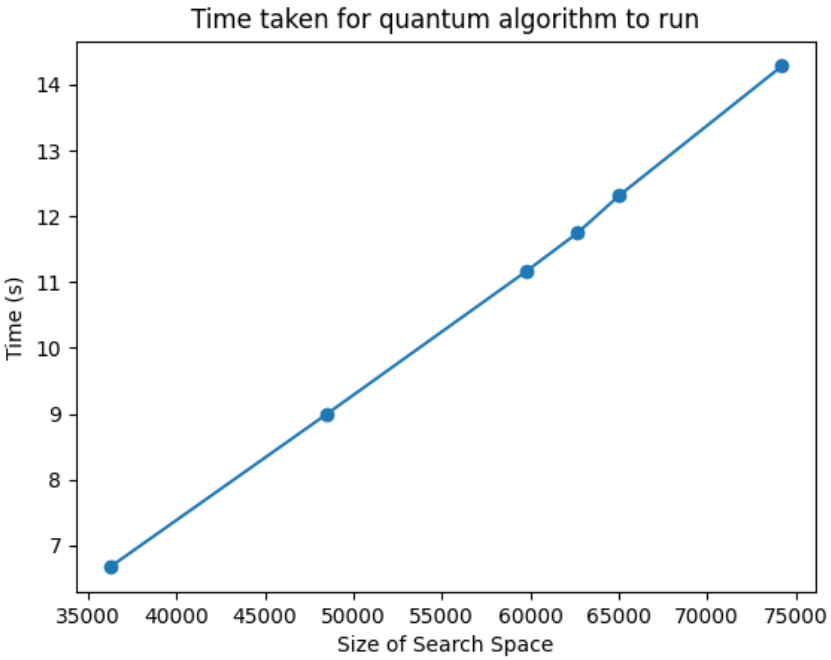



Figure 13

**times the search space of the classical algorithm, the classical algorithm takes about 1000 times the run-time of the quantum algorithm**.

The slope in the time taken vs search space graph of the quantum algorithm is almost zero, showing that quantum algorithms scale extremely well for such optimization problems.

Next, we compare the objective values obtained from the classical vs the quantum algorithm. We randomly cancelled two flights in the given dataset, and have plotted the objective produced from both the classical and quantum algorithms.

As can be seen from the graph plotted in log scale, the objectives produced are very similar. This shows that not only is the quantum algorithm extremely fast, but it also produces quality solutions.

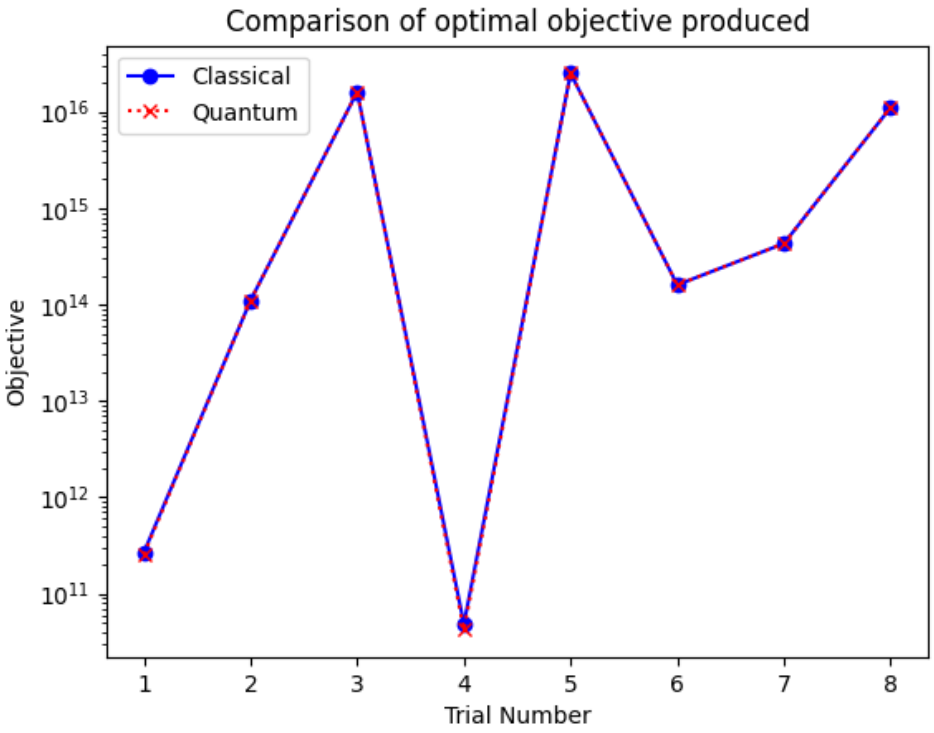



Figure 14: Plot (log scale) showing the quality of quantum vs classical solution

## §8 Conclusion

Framing the flight rescheduling problem in terms of a **Constrained Quadratic Model** and using a **Quantum Annealer** to solve the same is an extremely effective approach to solve the rescheduling problem. Such a method permits us to reach all the required goals, while being flexible and efficient.

From the data points observed, we can conclude that a quantum algorithm is much faster than a classical one, while it does not compromise on the solution quality. A quantum annealer scales very well with an increasing search space size, to which the classical algorithms employed by Google's OR Tools do not even come close.

As such, we can conclude the method of reduction of the rescheduling problem to a CQM described in above sections paired with quantum algorithms is the ideal way to solve such large optimization problems.