

QUANTUM COMPUTING

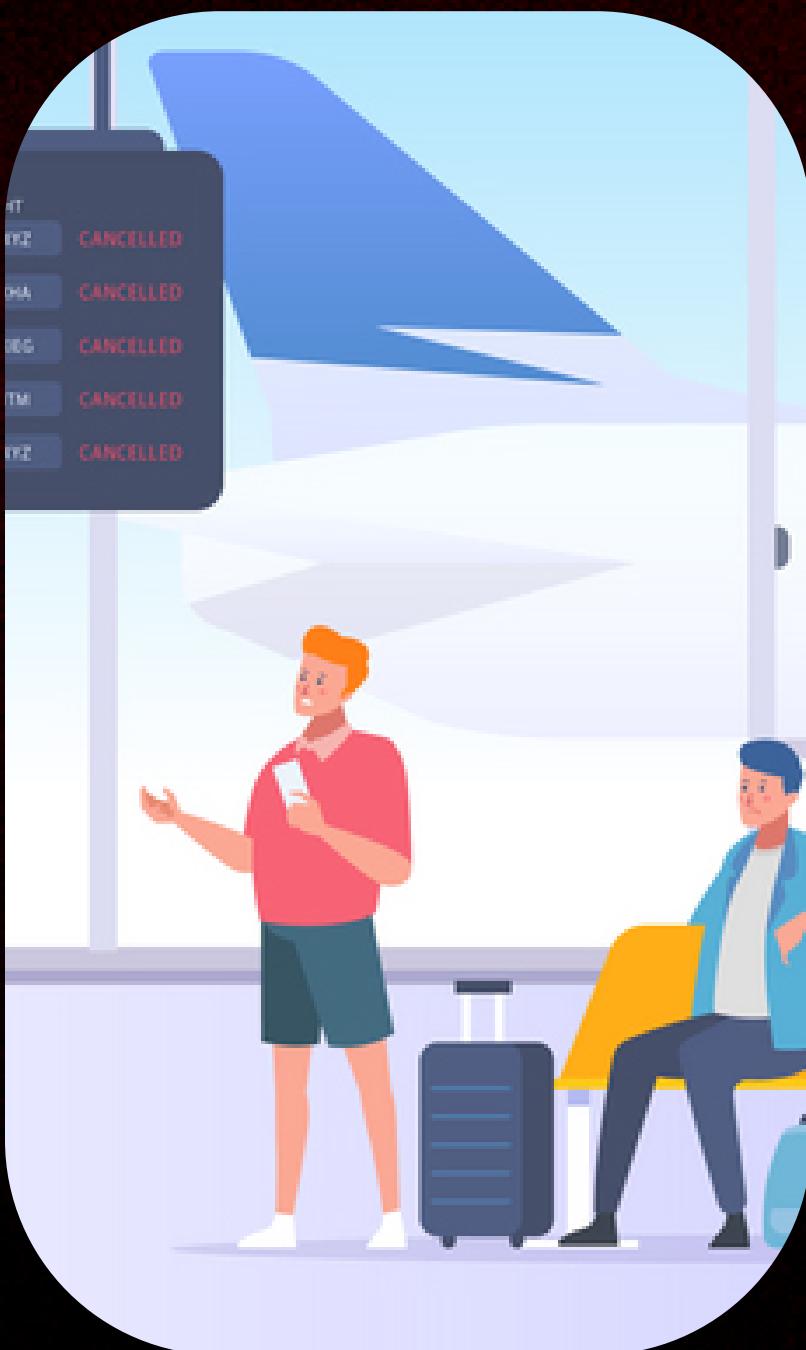
Team 59

- Passenger Recovery during Airline Disruptions

Agenda



Description of the Problem Statement	3
Objectives and Assumptions Used	4
Pipeline of the model	5
Rule Engine	6
The First Stage: Path Finding	7
The Second Stage: Passenger Reallocation	8
The Proposed Methodology	9
Future Scope and Business Proposition	10
Novelty	11
Questions?	13



PROBLEM DESCRIPTION

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

TASK 1

Find Alternate Flight Routes for Impacted PNRs

TASK 2

Reallocate the PNRs to the flight plans found

Our Goals

Expected outcomes of the models



GOAL # 1

Flexible Business Rules



GOAL # 2

Handling Multiple Disruptions while
optimizing the solution



GOAL # 3

Time is Everything. It has to be Fast ...
No Compromise over Correctness though...

Assumptions

Logical Assumptions we took/can take

There is a maximum of 'k' stopovers allowed in any flight plan

Solution Space Reduction from $O(N!)$ $\rightarrow O(k! * N) \rightarrow O(N)$

It is not allowed to visit an already visited airport in a flight plan

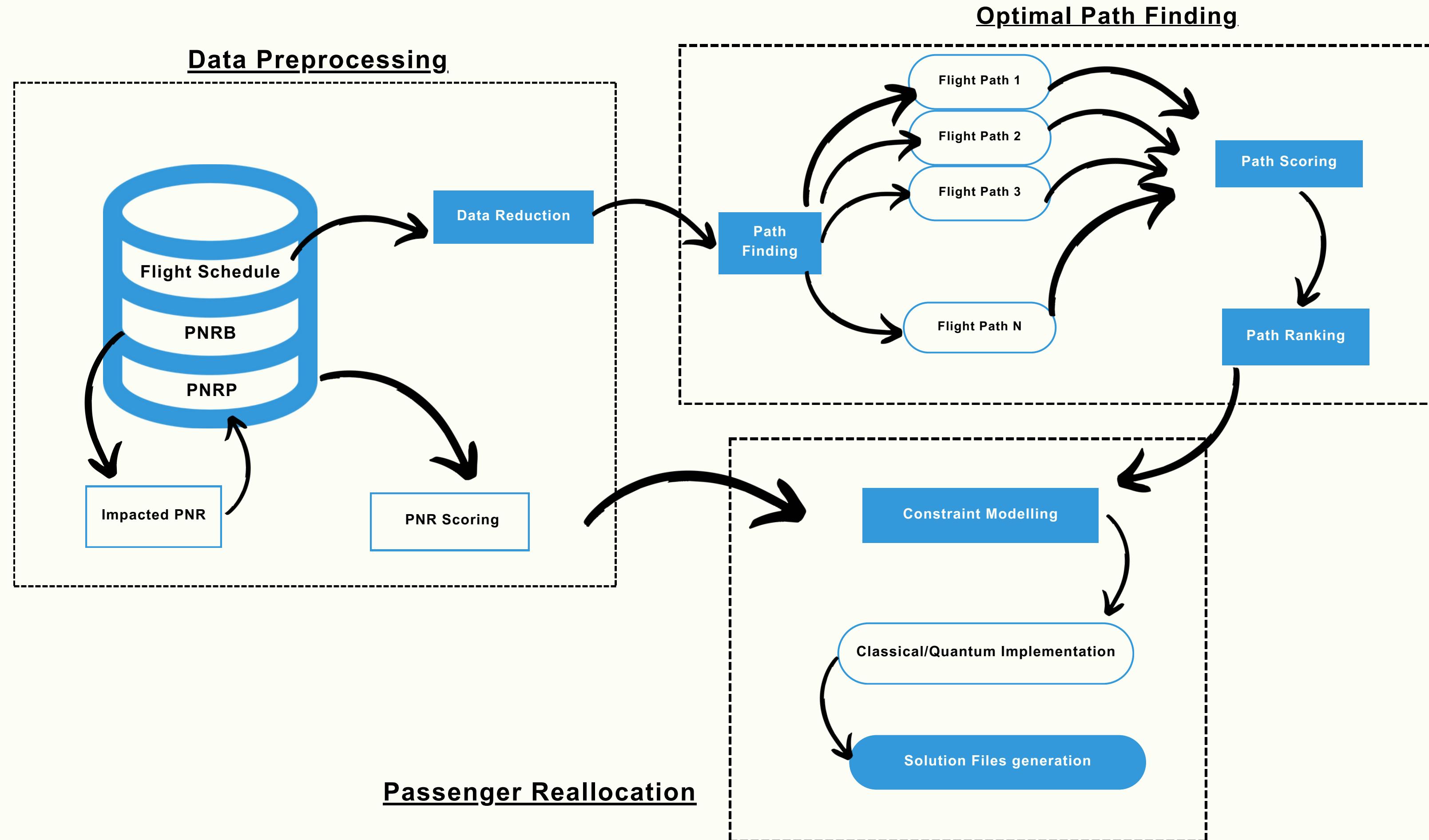
Loop Removal allows for further reduction in complexity

There is a set A of airports where it is allowed for a stopover

Targeted Searches \Leftrightarrow Faster Speed (Not Implemented)

Priority is to get maximum passengers accommodated, then to get most PNRs in default solution, while maximizing PNR scoring

Pipeline for Itinerary Searching



RULE ENGINE



Team 59



POSSIBLE DATA LOSS Some features might be lost if you save this workbook.

A	B	C	D	E
1 Business Rules	On/Off	Score		
2 SSR	1	200		
3 Per_Pax	1	50		
4 Loyalty_Silver	1	1500		
5 Loyalty_Gold	1	1600		
6 Loyalty_Platinum	1	1800		
7 Loyalty_PPlatinum	1	2000		
8 Booking_Group	1	500		
9 Paid_Service	1	200		
10 Downline_Connection	1	100		
11 EconomyClass	1	1500		
12 BusinessClass	1	1850		
13 PremiumEconomyClass	1	1650		
14 FirstClass	1	2000		
15 Upgrade_allow	1	0		
16 Downgrade_allow	1	0		
17				
18				
19				
20				
21				

POSSIBLE DATA LOSS Some features might be lost if you save this workbook.

Flight Rules

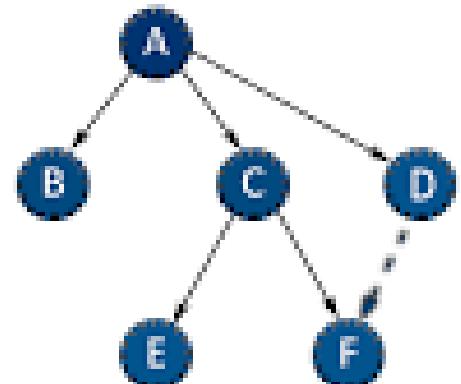
A	B	C	D	E
1 Flight Rules	On/Off	Score		
2 Arr_LT_6hrs	1	70		
3 Arr_LT_12hrs	1	50		
4 Arr_LT_24hrs	1	40		
5 Arr_LT_48hrs	1	30		
6 Equipment	1	50		
7 Same_Citipairs	1	40		
8 DiffSame_citipairs	1	30		
9 Different_citipairs	1	20		
10 SPF_LT_6hrs	1	70		
11 SPF_LT_12hrs	1	50		
12 SPF_LT_24hrs	1	40		
13 SPF_LT_48hrs	1	30		
14 IsStopover	1	-20		
15 A_Grade	1	200		
16 B_Grade	1	180		
17 C_Grade	1	150		
18 D_Grade	1	150		
19				
20				
21				

STAGE 1: PATH FINDING

REJECTION LIST



BFS



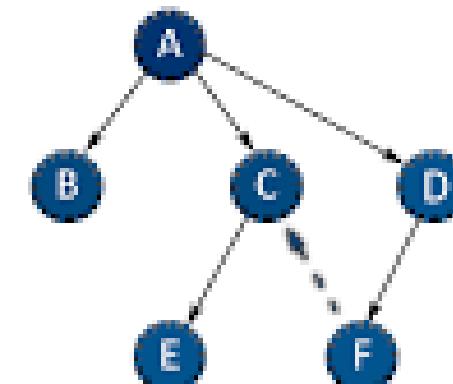
A B C D E F

Breadth First Search

Disadvantage

- Creating a Graph of possible paths while considering constraints burns through the resources
- Searching takes $O(N \log N)$

DFS

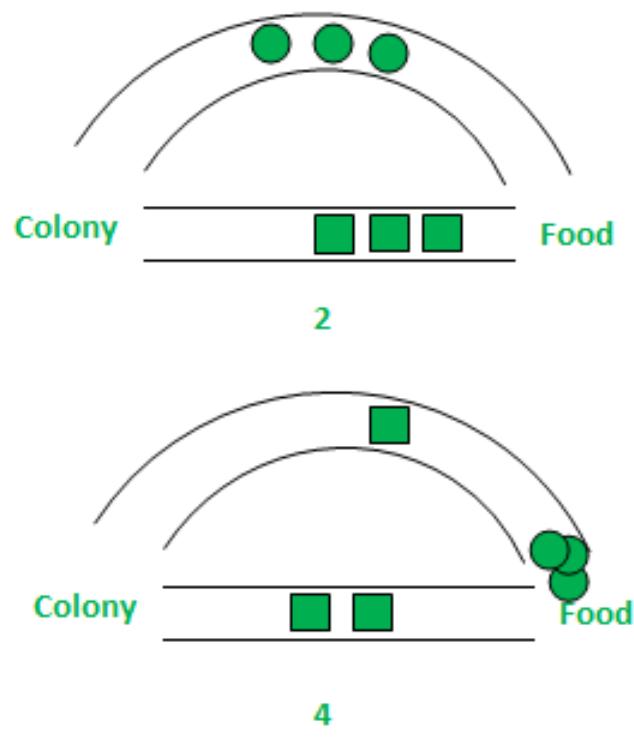


A D F C E B

Depth First Search

Disadvantage

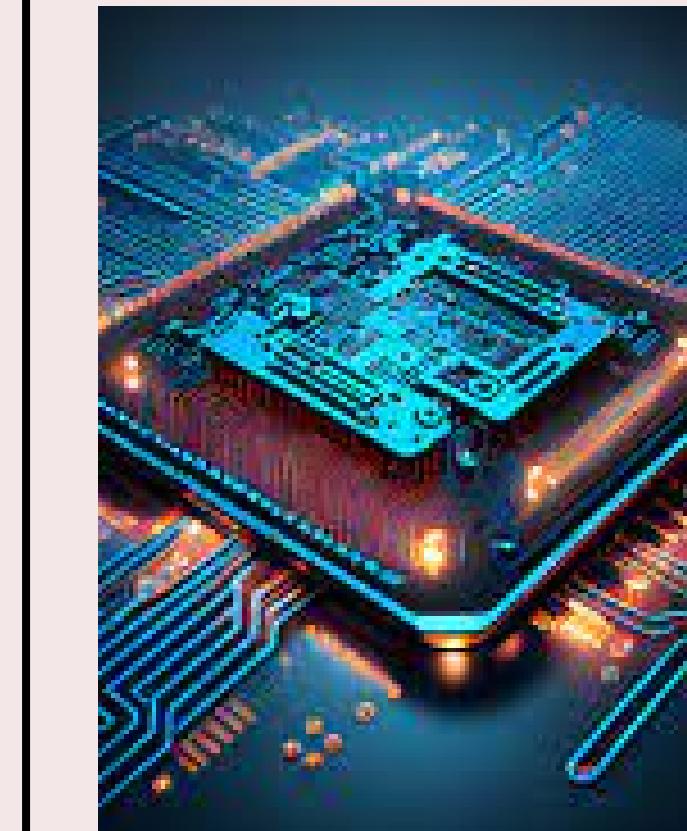
- Creating a Graph of possible paths while considering constraints burns through the resources
- Searching takes $O(N \log N)$



Ant Colony Optimization

Disadvantage

- Slow Convergence to Optimal Solution
- Pheromone Deposition can miss global Optima
- High number of ant makes the runtime blow up quadratically



QACO and Quantum Walk

Disadvantage

- Theoretically it was the most promising with time complexity $O(\sqrt{N})$. After actual implementation, we found out a different issue !!!

IMPLEMENTATION STAGE 1: PATH FINDING

The methods we tried but rejected

```
PS H:\My Drive\QC Inter IIT> & "C:/Users/Sukhvansh Jain/AppData/Local/Programs/Python/Python310/python.exe" "h:/My Drive/QC Inter IIT/QACO_updated.py"
{'1110': 36, '0111': 43, '1111': 253, '0010': 33, '1000': 42, '0000': 233, '0011': 30, '1010': 37, '1101': 35
, '0110': 45, '0101': 31, '1100': 33, '1001': 39, '0100': 47, '0001': 40, '1011': 47}
PS H:\My Drive\QC Inter IIT>
```

Quantum Ant Colony Optimization

Rejection reason: Unstable with current quantum systems, Need more Qubits

Classical Ant Colony Optimization

Rejection reason: Runtime is too high

SIMULATION STATISTICS:

Airports Count: 30

Time Span: 101

Flight ID of Disruption: 13485

Source Node of Disruption: 14

Destination Node of Disruption: 16

Departure Time of Disruption: 4 // 1

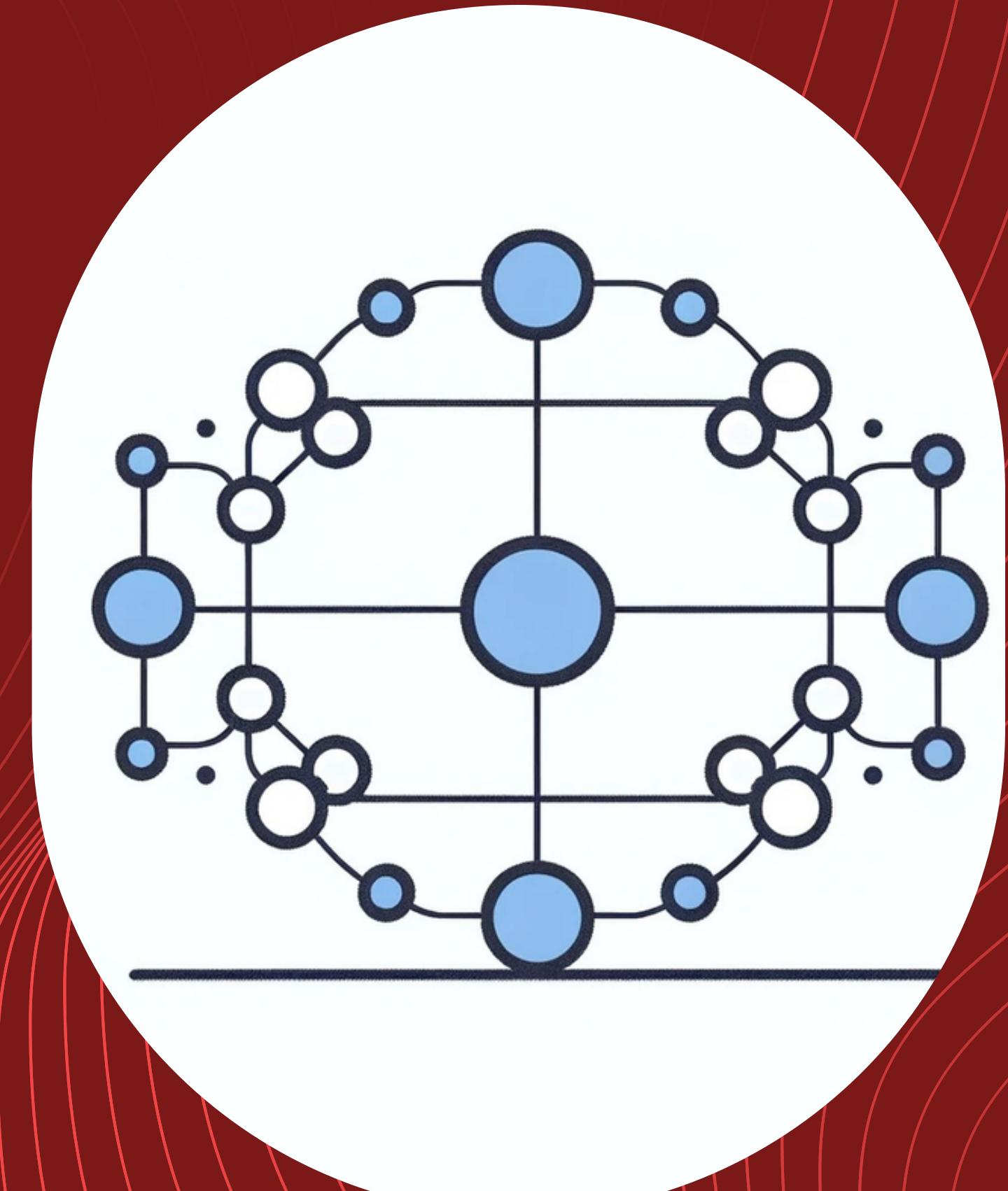
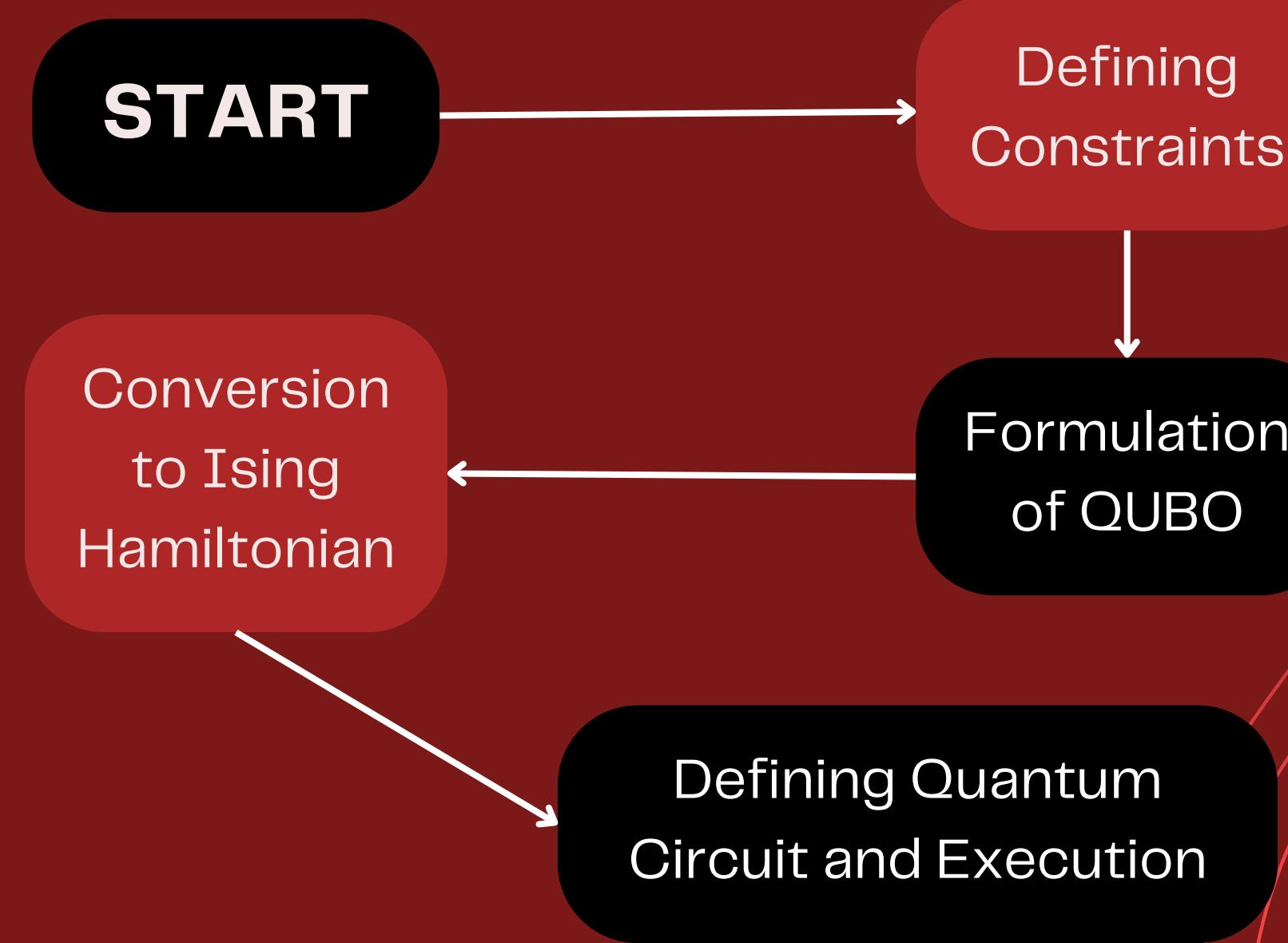
Arrival Time of Disruption: 3 // 2

```
{'Path': None, 'Path_cost': inf, 'Epoch': 1, 'Clock': 1.017286200003582}
{'Path': None, 'Path_cost': inf, 'Epoch': 2, 'Clock': 2.7474938000086695}
{'Path': None, 'Path_cost': inf, 'Epoch': 3, 'Clock': 5.15698010000051}
{'Path': None, 'Path_cost': inf, 'Epoch': 4, 'Clock': 7.3786158000002615}
{'Path': [14, 16.0], 'Path_cost': 25, 'Epoch': 5, 'Clock': 8.947287200004212}
{'Path': [14, 16.0], 'Path_cost': 25, 'Epoch': 6, 'Clock': 10.719987500007846}
{'Path': [14, 16.0], 'Path_cost': 25, 'Epoch': 7, 'Clock': 12.7915151000052}
{'Path': [14, 16.0], 'Path_cost': 25, 'Epoch': 8, 'Clock': 14.521899100000155}
{'Path': [14, 16.0], 'Path_cost': 25, 'Epoch': 9, 'Clock': 16.948991199999}
{'Path': [14, 16.0], 'Path_cost': 25, 'Epoch': 10, 'Clock': 18.764366200004588}
{'Path': [14, 16.0], 'Path_cost': 25, 'Epoch': 11, 'Clock': 21.050808400003007}
{'Path': [14, 16.0], 'Path_cost': 25, 'Epoch': 12, 'Clock': 23.126063900010195}
{'Path': [14, 16.0], 'Path_cost': 25, 'Epoch': 13, 'Clock': 24.864134600007674}
{'Path': [14, 16.0], 'Path_cost': 25, 'Epoch': 14, 'Clock': 26.586843099998077}
{'Path': [14, 16.0], 'Path_cost': 25, 'Epoch': 15, 'Clock': 28.319043299998157}
```

For quantum Reference: Mrityunjay Ghosh, Nivedita Dey, Debdeep Mitra, Amlan Chakrabarti. A novel quantum algorithm for ant colony optimization

QAOA

Optimization Algoirthm



Reference: M Alpuente, D Ballis, S Escobar, and J Sapi~na. Journal of logical and algebraic methods in programming. 2021



Quantum Annealing

Benefits of Quantum Annealing

Efficiency: For optimization tasks, quantum annealing can outperform Gate based approach

Rapid Convergence: For its target problem areas, quantum annealing can converge to a solution more quickly than other methods

Error Tolerance: Quantum annealers typically have a higher tolerance for operational errors

Optimization Efficiency: It's specifically designed for optimization problems

Noise tolerance: as quantum annealing approach focus on finding minima through tunneling, small noise wont make any impact

Decoherence Resilience: quantum annealers leverage quantum tunneling, which can be effective even in the presence of some level of decoherence.

OPTIMIZATION USING QUANTUM ANNEALING

BQM

- D waves' Annealing Version of QUBO
- Solved Using Leap's Hybrid Solver

$$\sum_{k'} \sum_{m'} \sum_{l'} T_{jk'l'm'} q_{jk'l'm'} - \sum_i \sum_k \sum_l q_{ijkl} \left(T'_{ijkl} + \lambda_2 \sum_{k'} \sum_{m'} \sum_{l'} q_{jk'l'm'} \right) \leq 0 \quad \forall j \notin S, E$$

OBJECTIVE FUNCTION

$$\min \sum_e^E \sum_j^{S_2} \sum_k^{A_2} \sum_l^{D_T} |T'_{jekl} - \beta| q_{jekl}$$

$$\min \sum_i^{S_1} \sum_k^{A_1} \sum_l^{D_T} |T_{Sikl} - \alpha| q_{Sikl}$$

CONSTRAINTS

$$\sum_i^{S_1} \sum_k^{A_1} \sum_l^{D_T} q_{Sikl} = 1$$

$$\sum_e^E \sum_j^{S_2} \sum_k^{A_2} \sum_l^{D_T} q_{jekl} = 1$$

PATH FINDING

$$\sum_o \sum_k \sum_l q_{ojkl} - \sum_j \sum_k \sum_l q_{jqrs} = 0 \quad \forall j \notin S, E$$

MIN/MAX TIME DELAY

$$\sum_{k'} \sum_{m'} \sum_{l'} T_{jk'l'm'} q_{jk'l'm'} - \sum_i \sum_k \sum_l q_{ijkl} \left(T'_{ijkl} + \lambda_1 \sum_{k'} \sum_{m'} \sum_{l'} q_{jk'l'm'} \right) \geq 0 \quad \forall j \notin S, E$$

CQM



Team 59

Same constraints as BQM (NOTE: not QUBO)

Does not form a penalty function to be added in the objective function

Provides Efficiency and Speed over BQM

NOTE: CQM is more than 250% faster than BQM !!!

Can provide solutions with no violation of constraints (NOTE: Not the case with BQM)

Classical Approach

Path Finding Using Classical Recursion

The exploration of alternative flight routes is facilitated through a recursive search mechanism.

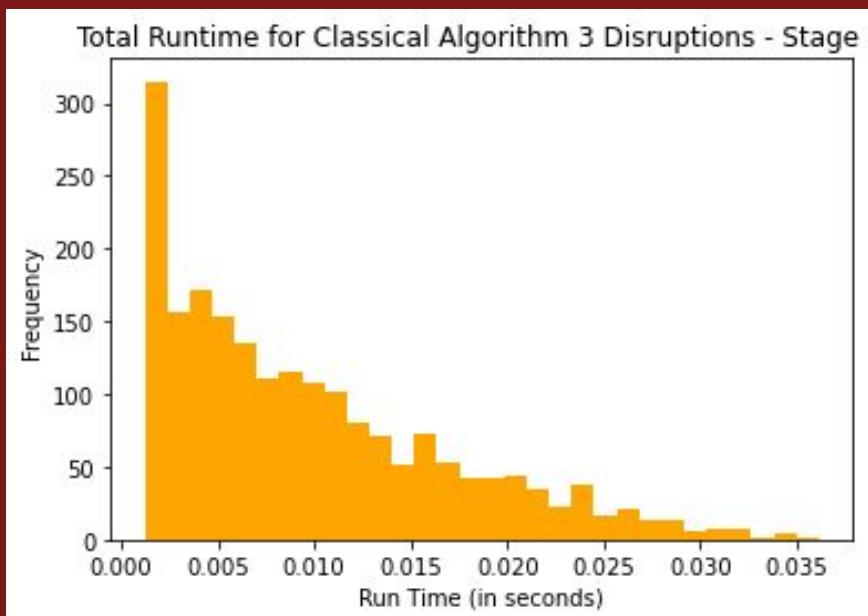
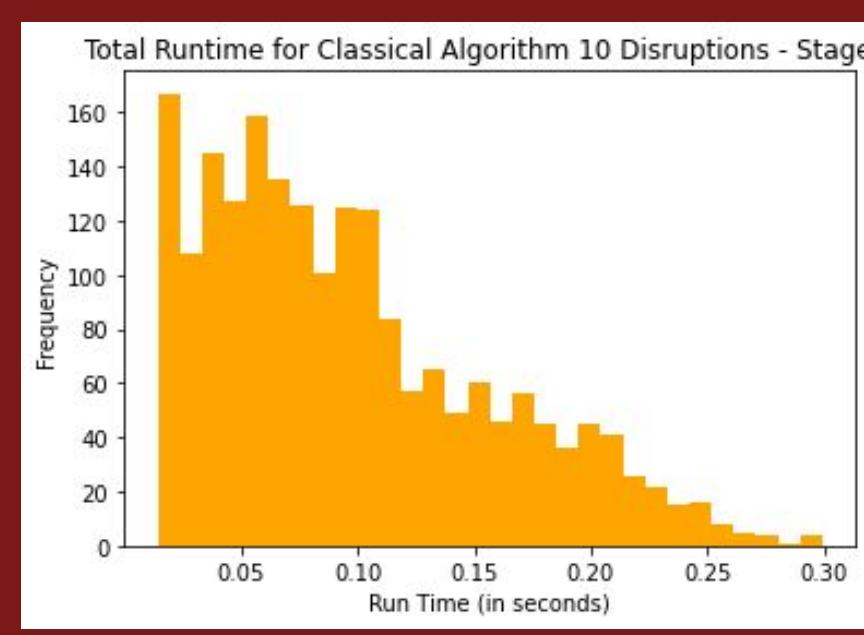
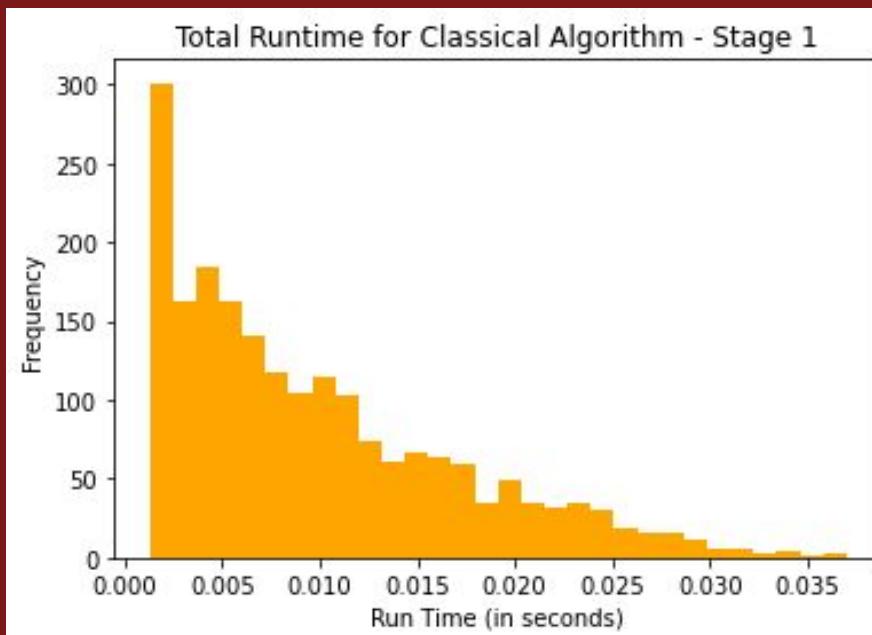
Despite the theoretical concern of poor time complexity associated with recursion, real-world constraints dictate that stopovers rarely exceed three flights

This limitation ensures a low recursion depth, contributing to the code's remarkable speed.

We used Pandas as our data handler, with Numpy under the hood. The Original creator of pandas, Wes McKinney, was overly obsessed with efficiency and speed.

Stage 1: Runtime Comparison

BEST CLASSICAL APPROACH



Average Runtime:
0.029935202146299557 seconds

Accuracy: 100%
(Since a deterministic classical approach is used with no approximations with all constraints satisfied)

Paths Found: All possible paths given a constraint

BEST QUANTUM APPROACH

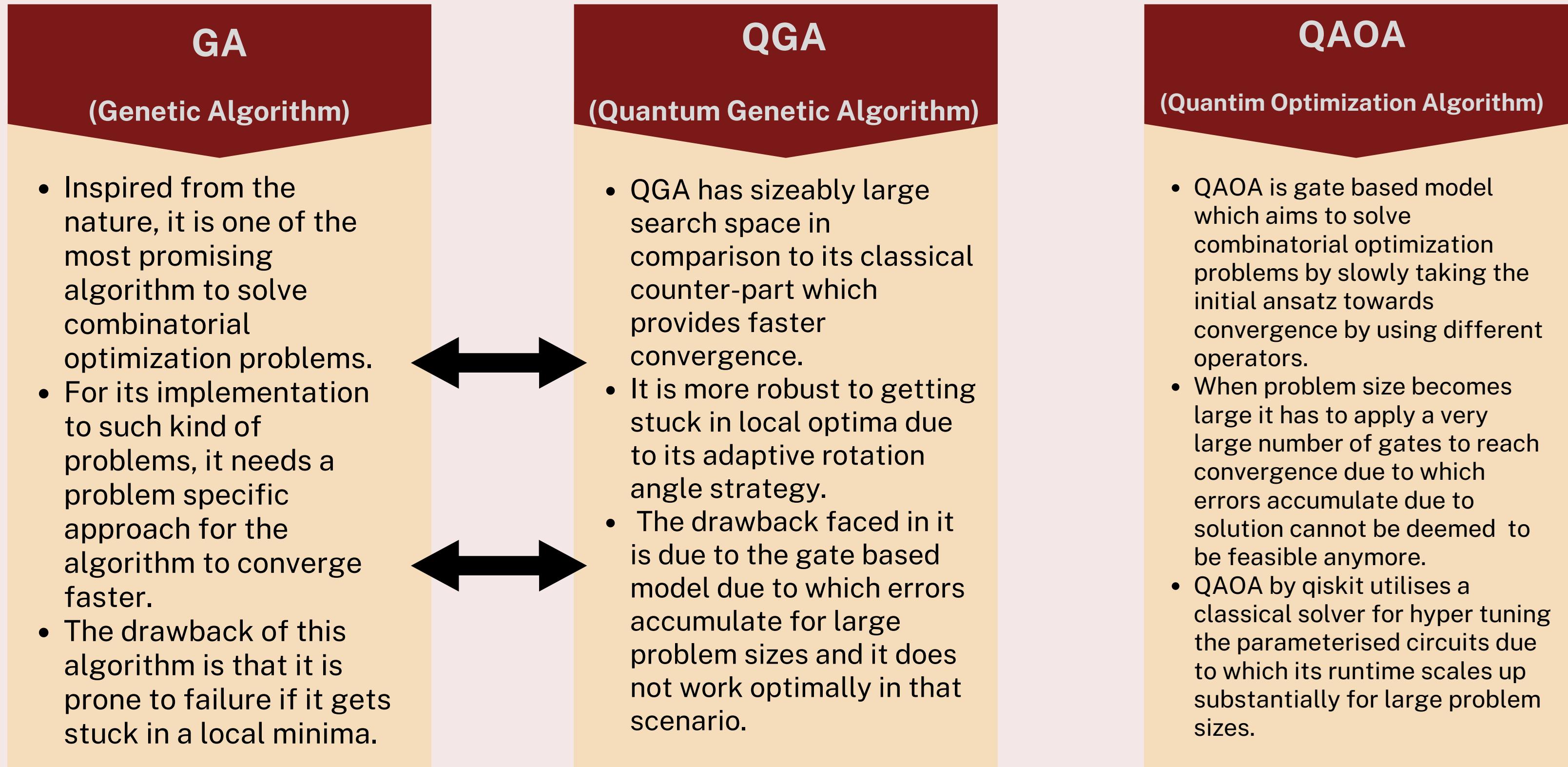
For "INV-ZZ-6029799" as cancelled

```
Top 5 unique solutions:

Unique Solution 1:
Energy: 0.0 Occurrences: 1
Sample: {'1': 0.0, '5': 0.0, '6': 0.0, '7': 0.0, 'q(1,3,1,5)': 0.0, 'q(2,0,0,4)': 1.0, 'q(2,1,2,0)': 0.0, 'q(2,1,2,2)': 0.0, 'q(2,1,2,7)': 0.0, 'q(3,0,3,1)': 0.0, 'q(3,0,3,3)': 0.0, 'q(3,0,4,6)': 0.0, 'q(3,0,4,8)': 0.0, 'q(3,1,3,5)': 0.0, 'q(3,2,1,5)': 0.0, 'q(3,2,2,5)': 0.0, 'q(3,2,3,5)': 0.0, 'q(3,3,2,5)': 0.0, 'q(3,3,3,5)': 0.0, 'q(3,3,4,5)': 0.0, 'q(3,4,3,5)': 0.0, 'q(3,4,4,5)': 0.0, 'q(3,5,3,5)': 0.0, 'q(3,5,4,5)': 0.0, 'q(3,6,3,5)': 0.0, 'q(3,6,4,5)': 0.0, 'q(3,7,3,5)': 0.0, 'q(3,7,4,5)': 0.0, 'q(3,8,3,5)': 0.0, 'q(3,8,4,5)': 0.0, 'q(3,9,3,5)': 0.0, 'q(3,9,4,5)': 0.0, 'q(3,10,3,5)': 0.0, 'q(3,10,4,5)': 0.0, 'q(3,11,3,5)': 0.0, 'q(3,11,4,5)': 0.0, 'q(3,12,3,5)': 0.0, 'q(3,12,4,5)': 0.0, 'q(3,13,3,5)': 0.0, 'q(3,13,4,5)': 0.0, 'q(3,14,3,5)': 0.0, 'q(3,14,4,5)': 0.0, 'q(3,15,3,5)': 0.0, 'q(3,15,4,5)': 0.0, 'q(3,16,3,5)': 0.0, 'q(3,16,4,5)': 0.0, 'q(3,17,3,5)': 0.0, 'q(3,17,4,5)': 0.0, 'q(3,18,3,5)': 0.0, 'q(3,18,4,5)': 0.0, 'q(3,19,3,5)': 0.0, 'q(3,19,4,5)': 0.0, 'q(3,20,3,5)': 0.0, 'q(3,20,4,5)': 0.0, 'q(3,21,3,5)': 0.0, 'q(3,21,4,5)': 0.0, 'q(3,22,3,5)': 0.0, 'q(3,22,4,5)': 0.0, 'q(3,23,3,5)': 0.0, 'q(3,23,4,5)': 0.0, 'q(3,24,3,5)': 0.0, 'q(3,24,4,5)': 0.0, 'q(3,25,3,5)': 0.0, 'q(3,25,4,5)': 0.0, 'q(3,26,3,5)': 0.0, 'q(3,26,4,5)': 0.0, 'q(3,27,3,5)': 0.0, 'q(3,27,4,5)': 0.0, 'q(3,28,3,5)': 0.0, 'q(3,28,4,5)': 0.0, 'q(3,29,3,5)': 0.0, 'q(3,29,4,5)': 0.0, 'q(3,30,3,5)': 0.0, 'q(3,30,4,5)': 0.0, 'q(3,31,3,5)': 0.0, 'q(3,31,4,5)': 0.0, 'q(3,32,3,5)': 0.0, 'q(3,32,4,5)': 0.0, 'q(3,33,3,5)': 0.0, 'q(3,33,4,5)': 0.0, 'q(3,34,3,5)': 0.0, 'q(3,34,4,5)': 0.0, 'q(3,35,3,5)': 0.0, 'q(3,35,4,5)': 0.0, 'q(3,36,3,5)': 0.0, 'q(3,36,4,5)': 0.0, 'q(3,37,3,5)': 0.0, 'q(3,37,4,5)': 0.0, 'q(3,38,3,5)': 0.0, 'q(3,38,4,5)': 0.0, 'q(3,39,3,5)': 0.0, 'q(3,39,4,5)': 0.0, 'q(3,40,3,5)': 0.0, 'q(3,40,4,5)': 0.0, 'q(3,41,3,5)': 0.0, 'q(3,41,4,5)': 0.0, 'q(3,42,3,5)': 0.0, 'q(3,42,4,5)': 0.0, 'q(3,43,3,5)': 0.0, 'q(3,43,4,5)': 0.0, 'q(3,44,3,5)': 0.0, 'q(3,44,4,5)': 0.0, 'q(3,45,3,5)': 0.0, 'q(3,45,4,5)': 0.0, 'q(3,46,3,5)': 0.0, 'q(3,46,4,5)': 0.0, 'q(3,47,3,5)': 0.0, 'q(3,47,4,5)': 0.0, 'q(3,48,3,5)': 0.0, 'q(3,48,4,5)': 0.0, 'q(3,49,3,5)': 0.0, 'q(3,49,4,5)': 0.0, 'q(3,50,3,5)': 0.0, 'q(3,50,4,5)': 0.0, 'q(3,51,3,5)': 0.0, 'q(3,51,4,5)': 0.0, 'q(3,52,3,5)': 0.0, 'q(3,52,4,5)': 0.0, 'q(3,53,3,5)': 0.0, 'q(3,53,4,5)': 0.0, 'q(3,54,3,5)': 0.0, 'q(3,54,4,5)': 0.0, 'q(3,55,3,5)': 0.0, 'q(3,55,4,5)': 0.0, 'q(3,56,3,5)': 0.0, 'q(3,56,4,5)': 0.0, 'q(3,57,3,5)': 0.0, 'q(3,57,4,5)': 0.0, 'q(3,58,3,5)': 0.0, 'q(3,58,4,5)': 0.0, 'q(3,59,3,5)': 0.0, 'q(3,59,4,5)': 0.0, 'q(3,60,3,5)': 0.0, 'q(3,60,4,5)': 0.0, 'q(3,61,3,5)': 0.0, 'q(3,61,4,5)': 0.0, 'q(3,62,3,5)': 0.0, 'q(3,62,4,5)': 0.0, 'q(3,63,3,5)': 0.0, 'q(3,63,4,5)': 0.0, 'q(3,64,3,5)': 0.0, 'q(3,64,4,5)': 0.0, 'q(3,65,3,5)': 0.0, 'q(3,65,4,5)': 0.0, 'q(3,66,3,5)': 0.0, 'q(3,66,4,5)': 0.0, 'q(3,67,3,5)': 0.0, 'q(3,67,4,5)': 0.0, 'q(3,68,3,5)': 0.0, 'q(3,68,4,5)': 0.0, 'q(3,69,3,5)': 0.0, 'q(3,69,4,5)': 0.0, 'q(3,70,3,5)': 0.0, 'q(3,70,4,5)': 0.0, 'q(3,71,3,5)': 0.0, 'q(3,71,4,5)': 0.0, 'q(3,72,3,5)': 0.0, 'q(3,72,4,5)': 0.0, 'q(3,73,3,5)': 0.0, 'q(3,73,4,5)': 0.0, 'q(3,74,3,5)': 0.0, 'q(3,74,4,5)': 0.0, 'q(3,75,3,5)': 0.0, 'q(3,75,4,5)': 0.0, 'q(3,76,3,5)': 0.0, 'q(3,76,4,5)': 0.0, 'q(3,77,3,5)': 0.0, 'q(3,77,4,5)': 0.0, 'q(3,78,3,5)': 0.0, 'q(3,78,4,5)': 0.0, 'q(3,79,3,5)': 0.0, 'q(3,79,4,5)': 0.0, 'q(3,80,3,5)': 0.0, 'q(3,80,4,5)': 0.0, 'q(3,81,3,5)': 0.0, 'q(3,81,4,5)': 0.0, 'q(3,82,3,5)': 0.0, 'q(3,82,4,5)': 0.0, 'q(3,83,3,5)': 0.0, 'q(3,83,4,5)': 0.0, 'q(3,84,3,5)': 0.0, 'q(3,84,4,5)': 0.0, 'q(3,85,3,5)': 0.0, 'q(3,85,4,5)': 0.0, 'q(3,86,3,5)': 0.0, 'q(3,86,4,5)': 0.0, 'q(3,87,3,5)': 0.0, 'q(3,87,4,5)': 0.0, 'q(3,88,3,5)': 0.0, 'q(3,88,4,5)': 0.0, 'q(3,89,3,5)': 0.0, 'q(3,89,4,5)': 0.0, 'q(3,90,3,5)': 0.0, 'q(3,90,4,5)': 0.0, 'q(3,91,3,5)': 0.0, 'q(3,91,4,5)': 0.0, 'q(3,92,3,5)': 0.0, 'q(3,92,4,5)': 0.0, 'q(3,93,3,5)': 0.0, 'q(3,93,4,5)': 0.0, 'q(3,94,3,5)': 0.0, 'q(3,94,4,5)': 0.0, 'q(3,95,3,5)': 0.0, 'q(3,95,4,5)': 0.0, 'q(3,96,3,5)': 0.0, 'q(3,96,4,5)': 0.0, 'q(3,97,3,5)': 0.0, 'q(3,97,4,5)': 0.0, 'q(3,98,3,5)': 0.0, 'q(3,98,4,5)': 0.0, 'q(3,99,3,5)': 0.0, 'q(3,99,4,5)': 0.0, 'q(3,100,3,5)': 0.0, 'q(3,100,4,5)': 0.0, 'q(3,101,3,5)': 0.0, 'q(3,101,4,5)': 0.0, 'q(3,102,3,5)': 0.0, 'q(3,102,4,5)': 0.0, 'q(3,103,3,5)': 0.0, 'q(3,103,4,5)': 0.0, 'q(3,104,3,5)': 0.0, 'q(3,104,4,5)': 0.0, 'q(3,105,3,5)': 0.0, 'q(3,105,4,5)': 0.0, 'q(3,106,3,5)': 0.0, 'q(3,106,4,5)': 0.0, 'q(3,107,3,5)': 0.0, 'q(3,107,4,5)': 0.0, 'q(3,108,3,5)': 0.0, 'q(3,108,4,5)': 0.0, 'q(3,109,3,5)': 0.0, 'q(3,109,4,5)': 0.0, 'q(3,110,3,5)': 0.0, 'q(3,110,4,5)': 0.0, 'q(3,111,3,5)': 0.0, 'q(3,111,4,5)': 0.0, 'q(3,112,3,5)': 0.0, 'q(3,112,4,5)': 0.0, 'q(3,113,3,5)': 0.0, 'q(3,113,4,5)': 0.0, 'q(3,114,3,5)': 0.0, 'q(3,114,4,5)': 0.0, 'q(3,115,3,5)': 0.0, 'q(3,115,4,5)': 0.0, 'q(3,116,3,5)': 0.0, 'q(3,116,4,5)': 0.0, 'q(3,117,3,5)': 0.0, 'q(3,117,4,5)': 0.0, 'q(3,118,3,5)': 0.0, 'q(3,118,4,5)': 0.0, 'q(3,119,3,5)': 0.0, 'q(3,119,4,5)': 0.0, 'q(3,120,3,5)': 0.0, 'q(3,120,4,5)': 0.0, 'q(3,121,3,5)': 0.0, 'q(3,121,4,5)': 0.0, 'q(3,122,3,5)': 0.0, 'q(3,122,4,5)': 0.0, 'q(3,123,3,5)': 0.0, 'q(3,123,4,5)': 0.0, 'q(3,124,3,5)': 0.0, 'q(3,124,4,5)': 0.0, 'q(3,125,3,5)': 0.0, 'q(3,125,4,5)': 0.0, 'q(3,126,3,5)': 0.0, 'q(3,126,4,5)': 0.0, 'q(3,127,3,5)': 0.0, 'q(3,127,4,5)': 0.0, 'q(3,128,3,5)': 0.0, 'q(3,128,4,5)': 0.0, 'q(3,129,3,5)': 0.0, 'q(3,129,4,5)': 0.0, 'q(3,130,3,5)': 0.0, 'q(3,130,4,5)': 0.0, 'q(3,131,3,5)': 0.0, 'q(3,131,4,5)': 0.0, 'q(3,132,3,5)': 0.0, 'q(3,132,4,5)': 0.0, 'q(3,133,3,5)': 0.0, 'q(3,133,4,5)': 0.0, 'q(3,134,3,5)': 0.0, 'q(3,134,4,5)': 0.0, 'q(3,135,3,5)': 0.0, 'q(3,135,4,5)': 0.0, 'q(3,136,3,5)': 0.0, 'q(3,136,4,5)': 0.0, 'q(3,137,3,5)': 0.0, 'q(3,137,4,5)': 0.0, 'q(3,138,3,5)': 0.0, 'q(3,138,4,5)': 0.0, 'q(3,139,3,5)': 0.0, 'q(3,139,4,5)': 0.0, 'q(3,140,3,5)': 0.0, 'q(3,140,4,5)': 0.0, 'q(3,141,3,5)': 0.0, 'q(3,141,4,5)': 0.0, 'q(3,142,3,5)': 0.0, 'q(3,142,4,5)': 0.0, 'q(3,143,3,5)': 0.0, 'q(3,143,4,5)': 0.0, 'q(3,144,3,5)': 0.0, 'q(3,144,4,5)': 0.0, 'q(3,145,3,5)': 0.0, 'q(3,145,4,5)': 0.0, 'q(3,146,3,5)': 0.0, 'q(3,146,4,5)': 0.0, 'q(3,147,3,5)': 0.0, 'q(3,147,4,5)': 0.0, 'q(3,148,3,5)': 0.0, 'q(3,148,4,5)': 0.0, 'q(3,149,3,5)': 0.0, 'q(3,149,4,5)': 0.0, 'q(3,150,3,5)': 0.0, 'q(3,150,4,5)': 0.0, 'q(3,151,3,5)': 0.0, 'q(3,151,4,5)': 0.0, 'q(3,152,3,5)': 0.0, 'q(3,152,4,5)': 0.0, 'q(3,153,3,5)': 0.0, 'q(3,153,4,5)': 0.0, 'q(3,154,3,5)': 0.0, 'q(3,154,4,5)': 0.0, 'q(3,155,3,5)': 0.0, 'q(3,155,4,5)': 0.0, 'q(3,156,3,5)': 0.0, 'q(3,156,4,5)': 0.0, 'q(3,157,3,5)': 0.0, 'q(3,157,4,5)': 0.0, 'q(3,158,3,5)': 0.0, 'q(3,158,4,5)': 0.0, 'q(3,159,3,5)': 0.0, 'q(3,159,4,5)': 0.0, 'q(3,160,3,5)': 0.0, 'q(3,160,4,5)': 0.0, 'q(3,161,3,5)': 0.0, 'q(3,161,4,5)': 0.0, 'q(3,162,3,5)': 0.0, 'q(3,162,4,5)': 0.0, 'q(3,163,3,5)': 0.0, 'q(3,163,4,5)': 0.0, 'q(3,164,3,5)': 0.0, 'q(3,164,4,5)': 0.0, 'q(3,165,3,5)': 0.0, 'q(3,165,4,5)': 0.0, 'q(3,166,3,5)': 0.0, 'q(3,166,4,5)': 0.0, 'q(3,167,3,5)': 0.0, 'q(3,167,4,5)': 0.0, 'q(3,168,3,5)': 0.0, 'q(3,168,4,5)': 0.0, 'q(3,169,3,5)': 0.0, 'q(3,169,4,5)': 0.0, 'q(3,170,3,5)': 0.0, 'q(3,170,4,5)': 0.0, 'q(3,171,3,5)': 0.0, 'q(3,171,4,5)': 0.0, 'q(3,172,3,5)': 0.0, 'q(3,172,4,5)': 0.0, 'q(3,173,3,5)': 0.0, 'q(3,173,4,5)': 0.0, 'q(3,174,3,5)': 0.0, 'q(3,174,4,5)': 0.0, 'q(3,175,3,5)': 0.0, 'q(3,175,4,5)': 0.0, 'q(3,176,3,5)': 0.0, 'q(3,176,4,5)': 0.0, 'q(3,177,3,5)': 0.0, 'q(3,177,4,5)': 0.0, 'q(3,178,3,5)': 0.0, 'q(3,178,4,5)': 0.0, 'q(3,179,3,5)': 0.0, 'q(3,179,4,5)': 0.0, 'q(3,180,3,5)': 0.0, 'q(3,180,4,5)': 0.0, 'q(3,181,3,5)': 0.0, 'q(3,181,4,5)': 0.0, 'q(3,182,3,5)': 0.0, 'q(3,182,4,5)': 0.0, 'q(3,183,3,5)': 0.0, 'q(3,183,4,5)': 0.0, 'q(3,184,3,5)': 0.0, 'q(3,184,4,5)': 0.0, 'q(3,185,3,5)': 0.0, 'q(3,185,4,5)': 0.0, 'q(3,186,3,5)': 0.0, 'q(3,186,4,5)': 0.0, 'q(3,187,3,5)': 0.0, 'q(3,187,4,5)': 0.0, 'q(3,188,3,5)': 0.0, 'q(3,188,4,5)': 0.0, 'q(3,189,3,5)': 0.0, 'q(3,189,4,5)': 0.0, 'q(3,190,3,5)': 0.0, 'q(3,190,4,5)': 0.0, 'q(3,191,3,5)': 0.0, 'q(3,191,4,5)': 0.0, 'q(3,192,3,5)': 0.0, 'q(3,192,4,5)': 0.0, 'q(3,193,3
```

STAGE 2: PASSENGER REALLOCATION

REJECTION LIST



IMPLEMENTATION - STAGE 2: RELOCATION

The methods we tried but rejected

```
Weights: [95, 4, 60, 32, 23, 72, 89, 62, 65, 46]
Values: [55, 10, 47, 5, 4, 50, 8, 61, 85, 87]
Knapsack Size: 269

[10.775767588452917, 31.69073243226771, 9.597849187232892, 18.325826107939626, 26.89366051874746, 15.323100008233538, 8.548038078105542, 5.845498199398683, 19.580448863
248538, 8.325104282202659]
Best String: 1000000111
Best Value: 288
```

Genetic Algorithm

Rejection reason: Runtime too high, slow Convergence, High Number of Generations Needed

Quantum Genetic Algorithm

Rejection reason: Gate Based model accumulates errors for large sizes, Runtime too high

Knapsack Problem

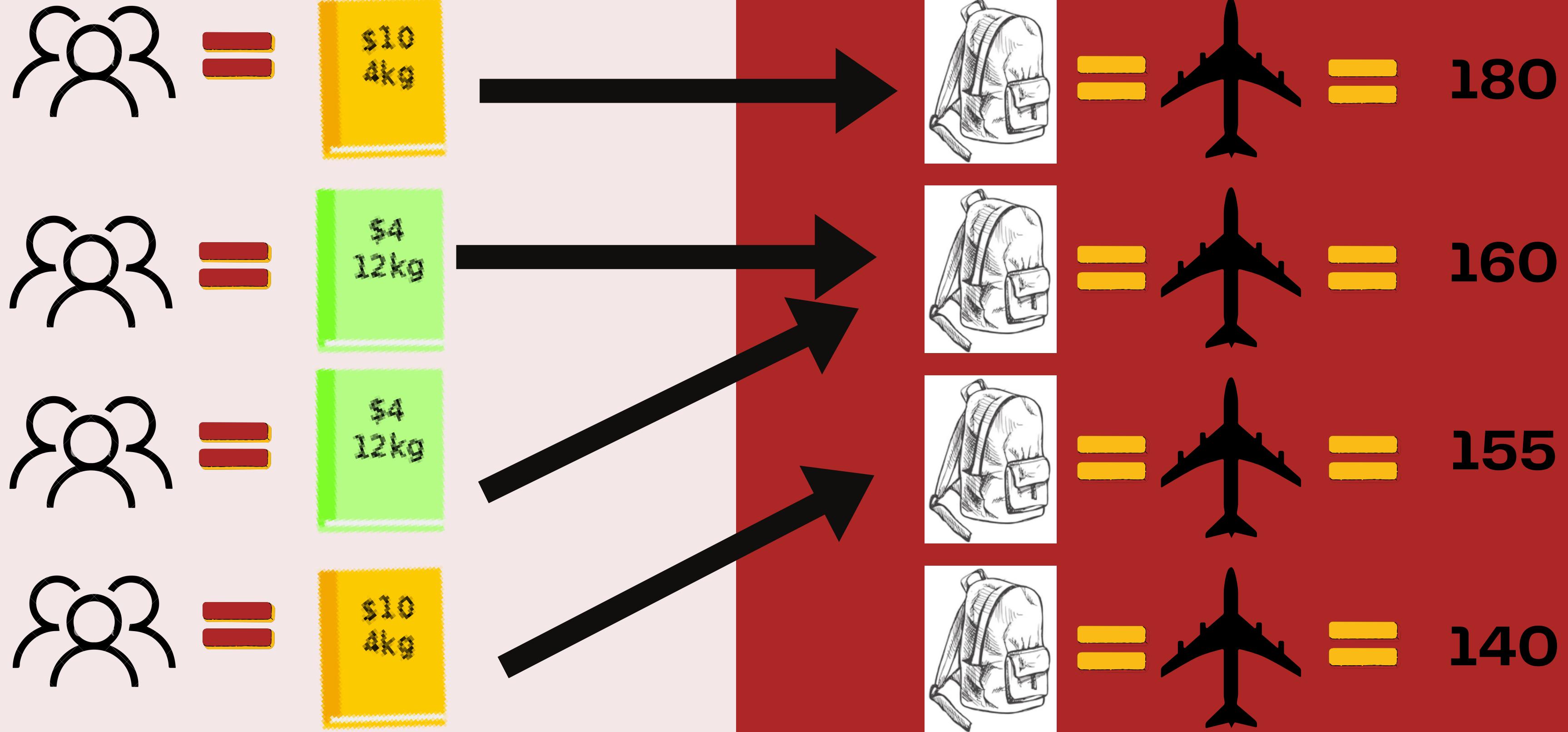
The art of Linear Programming helps to refine this problem as a multiknapsack problem with different classes.

$$\max \sum_{j=1}^n p_j x_j$$

$$\max \sum_{j=1}^n w_{ij} x_j \leq c_i$$



MODIFIED MULTIPLE KNAPSACKING



Constraints

$$\max \sum_{c \in S_p} \sum_{i \in c} \alpha_i \sum_j^K \sum_k p_{jk} x_{cijk}$$

$$\sum_j^N \sum_{c \in S_f} x_{cijk} \cdot w_j \leq C_{ik} \quad \forall k \in K, \quad \forall i \in c$$

$$\sum_{i \in S_1} \sum_k^K x_{cijk} \leq 1 \quad \forall j \in (1, 2, \dots, N), \quad \forall c \in S_p$$

$$\sum_{i \in S_2} \sum_k^K x_{cijk} \leq 1 \quad \forall j \in (1, 2, \dots, N), \quad \forall c \in S_p$$

$$\sum_{c \in S_p} \left(\sum_{i \in F_I} \sum_k^K x_{cijk} - \sum_{i \in F_O} \sum_k^K x_{cijk} \right) = 0 \quad \forall j \in 1, 2, \dots, N$$

$$\sum_{c \in S_p} \left(\sum_{i \in F_c} \sum_k^K x_{cijk} - N_c \cdot \sum_k^K x_{cfjk} \right) = 0 \quad \forall j \in 1, 2, \dots, N$$

HYPERPARAMETER NOTE:

How did we choose our alpha parameter?

Objective function which aims to maximize total PNR scoring while taking into account flight scoring

Capacity constraint for the class k of flight i

A passenger at best can be accommodated to one of the flights leaving from the departure airport

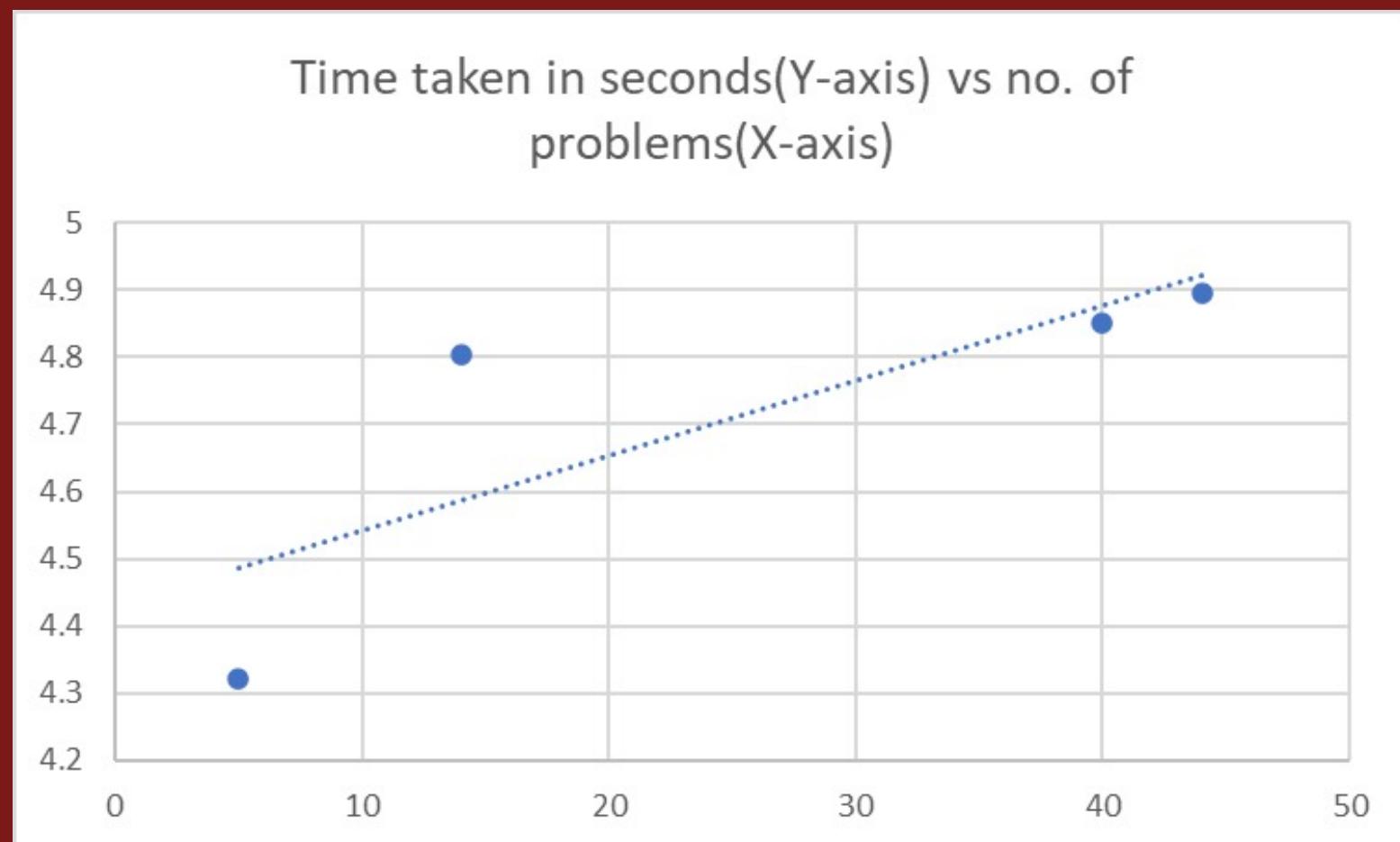
A passenger at best can be accommodated to one of the flights reaching the destination airport.

Ensures if a PNR reaches an intermediate airport then it should leave it as well

Deals with preserving the path of the PNR on which it travels

RESULTS

Quantum based knapsacking



If Problem size is scaled by 8 times, then the increase in QPU runtime is 3.3% !!!

QPU_ACCESS_TIME

00 h : 00 m : 00.016 s

CHARGE_TIME

00 h : 00 m : 05.000 s

RUN_TIME

00 h : 00 m : 05.091 s

QPU_ACCESS_TIME

00 h : 00 m : 00.032 s

CHARGE_TIME

00 h : 00 m : 04.759 s

RUN_TIME

00 h : 00 m : 04.759 s

QPU_ACCESS_TIME

00 h : 00 m : 00.016 s

CHARGE_TIME

00 h : 00 m : 04.320 s

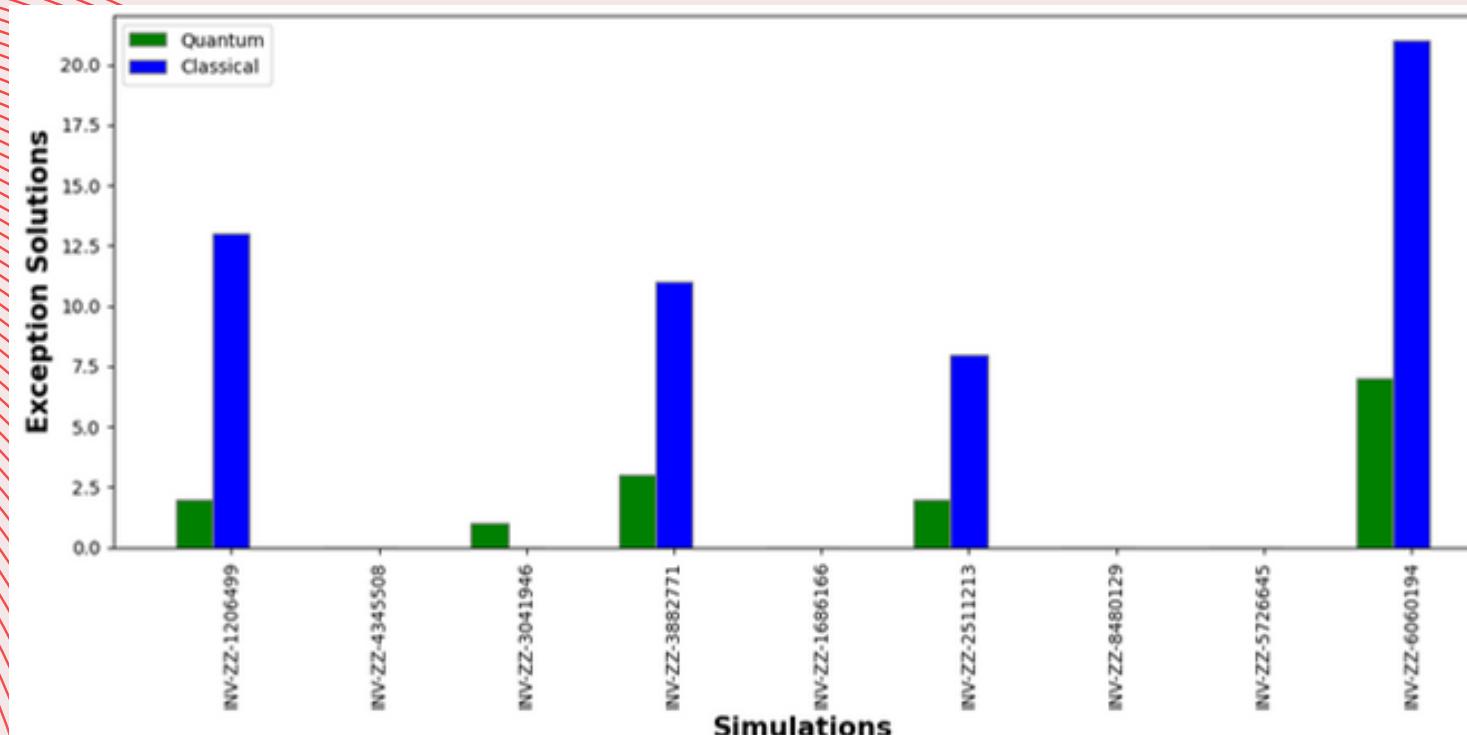
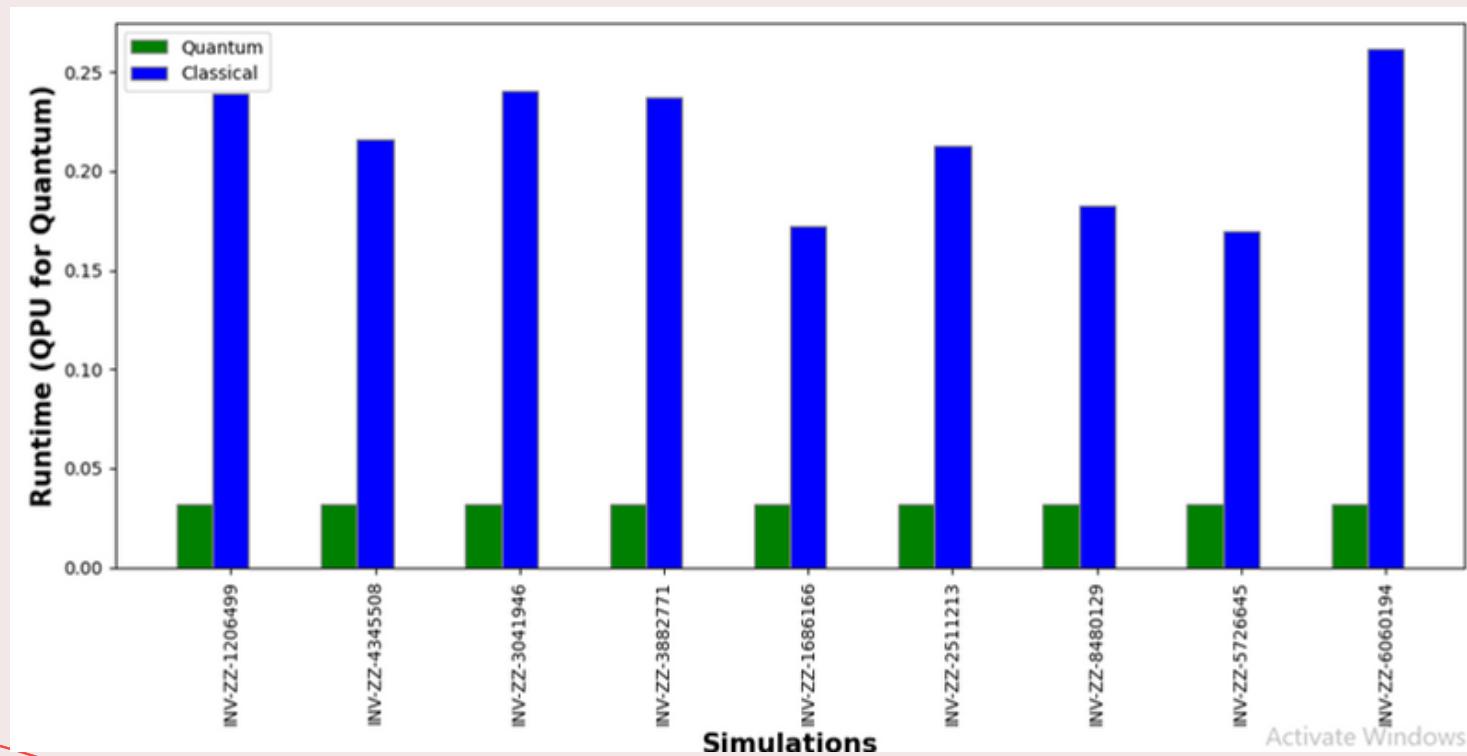
RUN_TIME

00 h : 00 m : 04.320 s

BEST CLASSICAL APPROACH

- For the classical approach to this problem we were unable to use class constrained multi - knapsack analog (NP - HARD problem)
- Instead we used a simple greedy approach to allocate highest ranking PNRs to the highest ranking flights

Stage 2 (Reallocation): Runtime Comparison



Classical Quantum

Average Number of Paths:
4.89 Paths

Average Ratio of Default to Exception:
87.0778698 %

Average Runtime:
0.27394458 seconds

Class Violations:
Nil

Average Number of Paths:
4.89 Paths

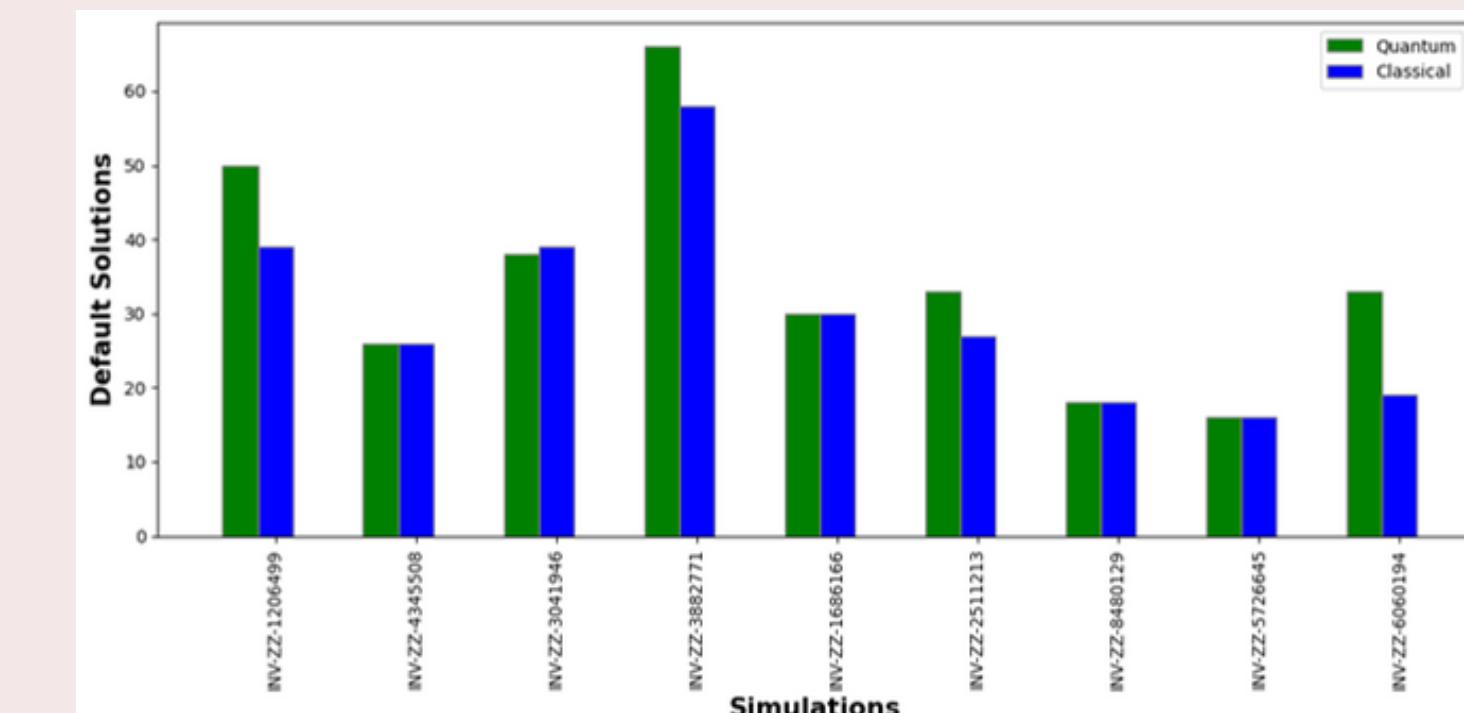
Average Ratio of Default to Exception:
96.22529242 %

Average Runtime:
0.032 seconds

Class Violations:
Nil

**8.5 Times
Faster !!!**

**Higher Overall
Score with
better
Optimizations !!!**



Overall Comparison

Classical – Classical

- Paths Found: All possible Paths found 100% of the time
- Solution Correctness/ Efficiency: Fails to allocate efficiently. Not all passengers are allocated in some cases
- Scaling of Problem Size: Capable of finding paths efficiently but runtime blows up exponentially for reallocation
- Total Average Runtime: 0.30394458 seconds

Classical – Quantum

- Paths Found: All possible Paths found 100% of the time
- Solution Correctness/ Efficiency: Efficiently Allocates the passengers while maximizing scores
- Scaling of Problem Size: Grows roughly linearly with increase in problem size (low value of coefficient)
- Total Average Runtime: $0.0299+0.032 = 0.0619$ seconds (Excluding Server Access Time)

Quantum – Quantum

- Paths Found: Top K Paths found with accuracy approx. 95%
- Solution Correctness/ Efficiency: Efficiently Allocates the passengers while maximizing scores
- Scaling of Problem Size: Grows roughly linearly with increase in problem size (low value of coefficient)
- Total Average Runtime: $0.016+0.032 = 0.048$ seconds (Excluding Server Access Time)

FINDING NEARBY FLIGHTS

Unique Selling Point

- Implemented Code to automatically change the search radius for city pairs as a Business Rule
- This can also be manually setup for faster search times
- Similar method can be used for finding connection flights used in real life

```
def nearby_airport(cancelled_flight_departure_airport, radius):  
    df = pd.read_csv('GlobalAirportDatabase.csv')  
  
    reference_airport_iata = cancelled_flight_departure_airport  
    search_radius_km_D = radius  
  
    df_airports = df  
  
    if reference_airport_iata in df_airports['IATA'].values:  
        reference_airport = df_airports[df_airports['IATA'] == reference_airport_iata].iloc[0]  
        ref_lat, ref_lon = reference_airport['Decimal Latitude'], reference_airport['Decimal Longitude']  
  
        df_airports['Distance_km'] = df_airports.apply(  
            lambda row: haversine(ref_lon, ref_lat, row['Decimal Longitude'], row['Decimal Latitude']),  
            axis=1  
        )  
        df_nearby_airports = df_airports[df_airports['Distance_km'] <= search_radius_km_D]  
  
        df_nearby_airports = df_nearby_airports[df_nearby_airports['IATA'] != reference_airport_iata]  
  
        df_nearby_airports.dropna(subset=['IATA'], inplace=True)  
        E_prime_d = df_nearby_airports['IATA'].tolist()  
    else:  
        return []  
        pass  
    return E_prime_d  
  
print(nearby_airport('DEL', 150))  
print(nearby_airport('DEL', 750))  
print(nearby_airport('DEL', 1000))
```

```
[]  
['LYP', 'LHE', 'MFG', 'MUX', 'RYK', 'ISB', 'RAZ', 'BHO', 'IDR', 'JLR', 'HJR', 'UDR', 'YOP', 'AGR', 'IXD', 'ATQ', 'VNS', 'KUU', 'IXC', 'DED', 'GWL', 'JOH', 'JAI', 'JSA', 'IXJ', 'KNU', 'KTU', 'LUH', 'IXL', 'LKO', 'IXP', 'PGH', 'SXR', 'BWA', 'PKR']  
['JAA', 'LYP', 'GIL', 'HDD', 'LHE', 'MFG', 'MOD', 'MUX', 'WNS', 'PEW', 'RYK', 'ISB', 'RAZ', 'SKZ', 'SDT', 'SUL', 'BDN', 'PZH', 'AMD', 'AKD', 'IXU', 'PAB', 'BHO', 'BDQ', 'BHO', 'BHU', 'IDR', 'JLR', 'JGA', 'IXY', 'HJR', 'NAG', 'RAJ', 'RPR', 'STV', 'UDR', 'YOP', 'GAY', 'PAT', 'AGR', 'IXD', 'ATQ', 'VNS', 'KUU', 'IXC', 'DED', 'GWL', 'JOH', 'JAI', 'JSA', 'IXJ', 'KNU', 'KTU', 'LUH', 'IXL', 'LKO', 'IXP', 'PGH', 'SXR', 'BWA', 'KTM', 'PKR', 'SIF', 'HTN']
```

U

Business Proposal and USPs

S

Solved Class Constrained Multiple Knapsack Problem with Quantum Annealing (Never solved before in Literature)

P

New Business Rule to Automatically Solve for City Pairs

Solves for multiple solution files using a single iteration (rather than looping repeatedly for different solutions)



**WE ARE NOW OPEN TO ANY
AND ALL QUESTIONS**