Approach to the Problem

First I tried to visualize this as a graph theory problem. We can model the flight network as a graph G(V,E) where airports are nodes and directed edges represent the respective flights from one airport to another. We need to find the optimal paths that satisfy the list of given conditions.

For that, I created a dictionary flight_info which stores detailed flight data, indexed by flight IDs, and another dictionary called flight_graph, which stores the adjacency list of airports, where each key is an airport, and the value is a list of outgoing flight IDs. Next I used BFS here since it helps to find the shortest path in an unweighted graph, which is the case here.

BFS Algorithm

Firstly, we start from the departure airport of the original flights that got cancelled. I used a queue to explore paths, as is typical of BFS. Here each element in the queue contains the current airport, the path taken to reach there, the number of flights taken till now, and the current arrival time. Also, to avoid revisiting nodes with the same number of flights taken, I maintained a set. Secondly, for path exploration, for each node I checked if it reaches the destination within the allowed number of planes. Moreover, for each outgoing flight from the current airport, I check if the flight has available capacity. If the path is valid, we update it, increment the number of flights taken till now, and finally add the new state to the queue. Lastly and obviously, the search will terminate when the destination airport is reached with the permissible number of layovers. If no valid path is found, we return an empty path and None for arrival time difference.

I have implemented the above algorithm only in my code.

Some Roadblocks and Solutions I devised

i) My first solution did not take into account the max capacity of each plane, and ended up returning outputs like allocating much total 300 passengers to a plane that could seat only 100! Hence I had to ensure that the BFS function checks the remaining capacity of the plane in each iteration – I did this by using a dictionary to keep track of the passenger count per flight and checking the capacities before confirming reallocations.

ii) BFS can become very time-consuming with such a large number of nodes and edges. So I tried to reduce the time complexity by limiting the search space - avoiding revisits to the same node with the same number of layovers, and terminating early if it found an optimal path.

Towards the end of the code I have also created a metric which can be used to rank the solutions. I have taken into account the percentage of passengers placed, the average number of layovers, and lastly the average time difference, and given it corresponding weights in the decreasing order, to arrive at a score which can be used to compare different solutions.