Solving Airport Gate Assignment Problem for Boston Airport

Introduction

An airport in any metro city has more than fifty gates and handles hundreds of flight for huge number of passengers on daily basis. Assigning gates to each aircraft for boarding, de-boarding and transferring the passengers is a complex problem as it involves various interdependent resources like- aircrafts, gates, gate facilities, crews, etc. And if this problem is not solved to a certain level, it could result in flight delays, poor customer services and inefficient use of gate facilities. Usually gates are assigned to each arriving and departing flights beforehand to ensure a smooth operation. This problem of assigning gates to each aircraft on any airport is called as Airport Gate Assignment Problem (AGAP).

The methods to solve AGAP can be divided into two major categories: expert system approaches and optimization. Within optimization, there are again two methods- exact methods and heuristics. Expert system consists of software's which stimulates the performance of human experts. Solutions are generated using the database, which contains rules and formulas created by human knowledge is a specific problem domain. An operator can change the rules of system to create better results in different problem situations. Expert system can handle complicated constraints in realistic operations but it lacks the capability of optimization. Exact methods which comes under optimization method also generates optimal solutions, but the only issue with this method is that the computational time increases by a large number if we increase the size of the problem. Here comes the heuristics and metaheuristics approach in the picture. The heuristics methods are fast and give good solutions but this method is problem specific, it may not give good solution if we change the objective or the model of the problem. Hence, we use metaheuristics which is more flexible as compared to other methods and give best results for AGAP's. Metaheuristics can be used to solve different types of problems related to gate assignment problem. [1]

We have tried to solve the AGAP using Simulated Annealing method on data from Boston Logan International Airport. This problem has been solved in the past using various metaheuristics algorithm but the problems involving real data were solved only a couple of times. Here we try to address the problem of Boston Logan Airport by using the real as much as possible, so this specific problem for Boston airport has not been addressed before.

Literature Review

A lot of research has been done and is going on this problem with different objectives and method approaches. Cheng and Ho [1] have used four different methods to get solutions for this problem. The objective of this paper is to minimize the total travelling distance of passengers. They have used Genetic Annealing, Tabu Search, Simulated Annealing and SA+TS (hybrid) methods to solve this problem.

A new greedy method for over-constrained AGAP is used by Ding and Zhu [2] to minimize ungated flights while providing initial feasible solutions. A new gate 'Apron' is added in this paper, where flights will arrive when no gates are available. Modified Tabu Search has been used to find solutions and are compared with unaltered Tabu Search algorithm.

Xu and Bailey [3] have used Tabu Search method to minimize the overall connection times that passenger walk to catch their connection flights. In this paper the problem is formulated as a mixed 0-1 quadratic integer programming problem, they reformulate it as a mixed 0-1 integer problem with a linear objective function and constraints. Then a Tabu Search based heuristic for the AGAP problem has been developed which employs a standard version of short term memory, dynamic Tabu tenure and aspiration criterion.

In another research paper by Bolat [4] there is a use of Genetic Annealing method to solve the problem of minimizing the dispersion of idle time of flights. Here the GA implementation utilizes problem-specific knowledge. Results show that the algorithm works well in realistic data and give some unique alternative solutions.

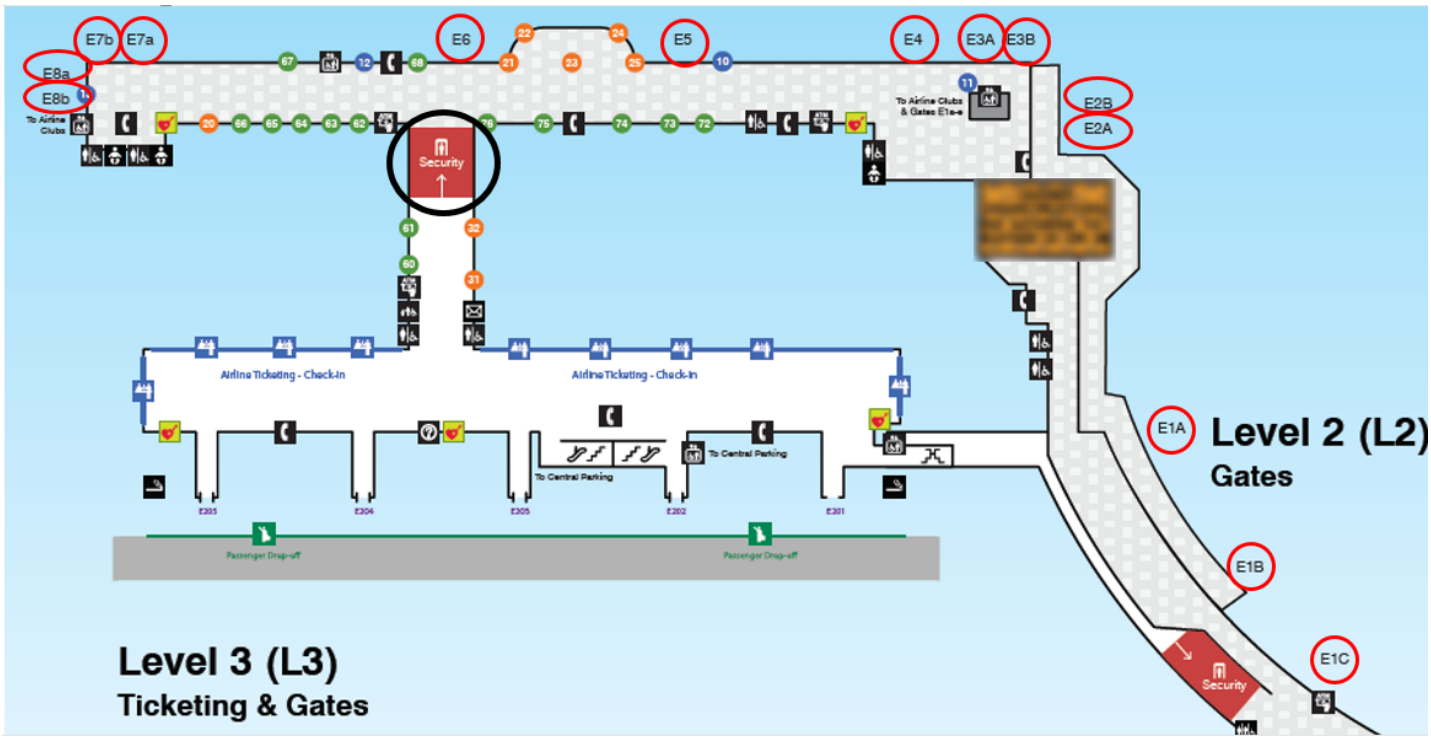
GA is again used by Hu and Di Palo [5] to solve multi objective GAP. In this paper the relative position between aircrafts is used to construct chromosomes, instead of the absolute positions of aircrafts. They implement a new uniform crossover operator which avoids producing infeasible chromosome.

The Model

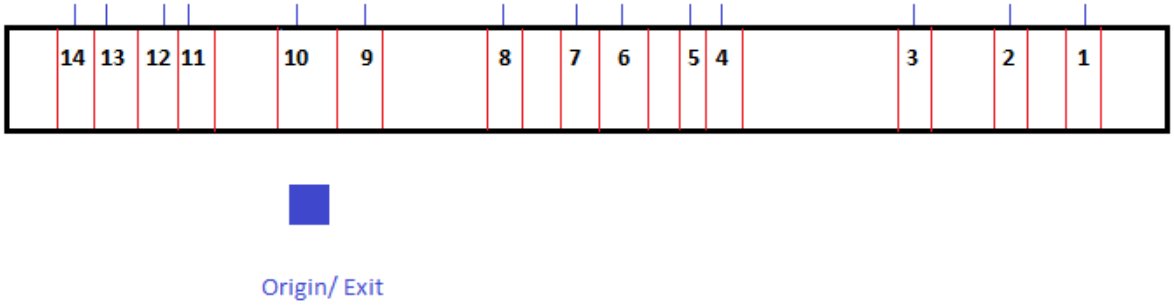
We have used Simulated Annealing to solve this problem. AGAP method can be treated as a scheduling problem subject to constraints and as SA handles scheduling problems better we have used Simulated Annealing over other algorithms.

Target Problem

The target problem for this project is Boston Logan Airport. Within this Airport, we choose to keep our focus on the Terminal E which handles almost all of the arriving and departing flights with a few exceptions [6]. This Terminal has a total of 14 Gates to handle all the flights. Below is a map of terminal E. The red circles indicate the gates and the black circle at security indicates the check in and check out gate.



The layout of gates has been simplified to calculate distances between different gates. The simplified image is shown below.



Assumptions

There were a few assumptions to be considered to model this problem. These are given below:

- All the international flights are handled by Terminal-E
- Passengers are transferred from Terminal E to other international flights
- The buffer time between the change of flights at a gate has been considered in the total time for which a flight is stationed at the gate, hence the arrival time and departure time of a flight 'a' is the time at which it docks and leaves its designated gate respectively
- All the airport gates are big enough to accommodate any size of aircraft
- There are no delays in the arrival or departure of flights
- Passengers only travel in straight lines and hence the distance between different gates has been modelled using L1 norm (Manhattan Distance)
- Distance between all gates has been approximated from the Airport map
- Check- in and check out gate has been considered same, this is treated as the zeroth gate (Origin/Exit) and it acts as the beginning and end point for the passengers

Data Preparation

The real data comprises of the total number of arriving and departing international flights in the month of October 2016. For the scope of this project we will keep our time frame for the first week of October 2016. The above data mentioned is from Massport website. [7] On average everyday 135 flights arrive and depart from Terminal E

Using the above knowledge, the rest of the data had to be simulated as follow:

- The Arriving time of each flight is taken between the interval of $[a_i*11, a_i*11 + 9]$ minutes
- The Departing time of each flight is taken between the interval of $[d_i*110, d_i*130]$ minutes
- Passenger distribution for each flight is taken between the interval of [80,250]

a_i is the arrival time of i^{th} flight; d_i is the departure time of i^{th} flight

This data for both arriving and departing flights is in Arrival.xlsx and Departure.xlsx files respectively.

Now, every arriving flight has a potential to that some of its passenger will be transferred in to some other departing flight therefore these passengers won't exit the airport rather they will go towards their designated flight gate. However, it is highly unlikely that a passenger will transfer between flights which are more than 8 hrs apart. One last thing to consider for generating transfer passenger data is that a passenger cannot transfer to a flight which has departed and he must have sufficient time if he has to walk to some other gate before the flight departs. This buffer has been considered to be of 40 mins i.e. there should be at least a 40 min gap between arrival time and departure time of two flight for a transfer to occur.

The Transfer passenger data is generated inside the CreateModel() script. The user can select how many passengers will transfer from each arriving flight by giving a percentage value in fractions (0 – 0.99). For example, if a user gives input as 0.25, it means that if an arriving flight has 100 at most 25 passengers will be transferred to randomly to the eligible flights.

Gate distance values are stored in Gates.xlsx file. As seen in the above figure nearest gate will be gate 10 and farthest gate will be gate 1.

Problem Formulation

In this AGAP problem, the goal is to assign gates to the airplanes so that the distance travelled by the passengers on the terminal is minimized. There will be three kinds of passengers and hence distances which will need to be minimized

- Passengers walking from entry gate to a designated flight
- Passengers walking from a flight to the exit gate
- Passengers transferring from one flight to another

The set of variables with which we will deal are given below [1]

G	the set of gates
F	the set of aircrafts
Z	objective function value
x_{ik}	a decision variable which is equal to 1 when aircraft *i* is assigned to gate *k*, and equal to 0 otherwise
a_i	the arrival time of aircraft *i*
d_i	the departure time of aircraft *i*
p_{ij}	the total number of transfer passengers from aircraft *i* to aircraft *j*
p_{i0}	the total number of arriving passengers of aircraft *i*
p_{0i}	the total number of departing passengers of aircraft *i*
w_{kl}	the walking distance between gate *k* and gate *l*
w_{k0}	the walking distance between gate *k* and the arrival hall
w_{0k}	the walking distance between the departure hall and gate *k*

Based on the notation defined above, we formulated the GAP model for our computational study as follows:

$$\text{Minimize } Z = \sum_{i\in F}\sum_{j\in F}\sum_{k\in G}\sum_{l\in G} p_{ij}w_{kl}x_{ik}x_{jl} + \sum_{i\in F}\sum_{k\in G}(p_{i0}w_{k0} + p_{0i}w_{0k})x_{ik}$$

This objective function will be subjected to a few constraints which are given below:

- Every aircraft must be assigned to one and only one gate
- No two aircrafts can be assigned to the same gate at the same time

These constraints are represented below

$$\sum_{k\in G} x_{ik} = 1, \forall i \in F$$

$$x_{ik}x_{jk}(d_j - a_i)(d_i - a_j) \leqslant 0,\ \forall i,j \in F,\quad \forall k \in G$$

Solution Generation

First step in the solution generation process is to make the data ready for algorithm. This is done by the "**CreateModel(Day, Transfer)"** function. Here the user can select which day's data he/she wants to analyse and what percent of passengers will be considered as transfer passenger.

The second step is to create an initial random solution q. The nVar which we use in our algorithm is (I+J-1), here I is the total number of flights and J is the total number of gates. But, as this problem is highly constrained, to generate an initial feasible solution randomly is very unlikely. To counter this a greedy algorithm was employed. First we sorted all the flights by their departure times then flights were assigned to the gates one by one. Say if even 14 flights arrive at the same time, by using this method each of those flights will be assigned to a different gate hence there will be no clash in timings and we can get an initial feasible solution. This process is done by the "**CreateInitialSol()**" function. This function also creates a X_{ij} data frame which will be used in the cost function.

"**CheckInitialSol()**" function makes sure that the initial solution generated is feasible and does not violate any constraints. This is achieved by checking the arrival and departure time of every flight on a gate.

The next step is to calculate the cost of this initial solution, this is done by the "**InitialCost()**" function. This algorithm takes in the passenger data and the data from CreateIntialSol() to calculate the initial cost of the solution. The main objective function which was described in the equations before, is used here, so the total cost output for this function is the cost of arriving passengers, cost of departing passengers and the cost of transfer passengers if there are any.

The above 4 function mentioned are used only once in the algorithm. The rest of the algorithm is described below.

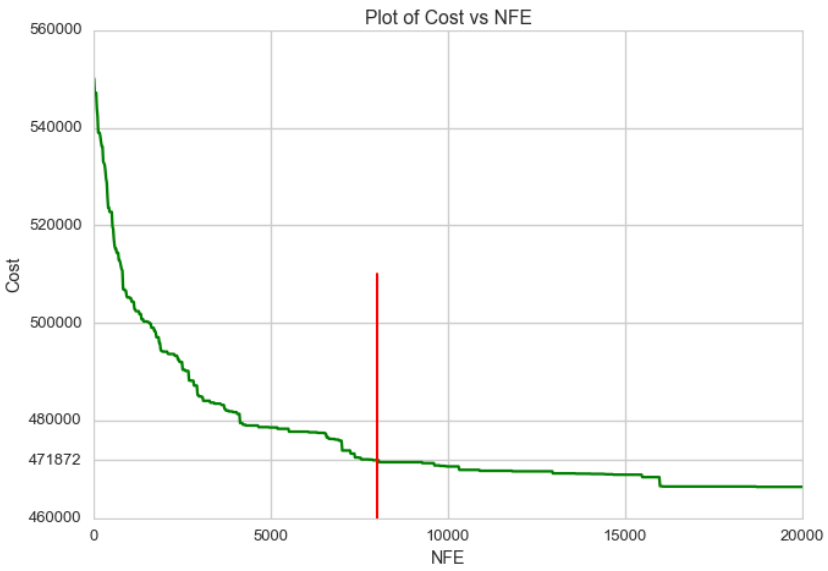
Once in the SA main loop the first function which is triggered is the "**CreateNeighbor()**" function. This function uses Swap, Reversion or Insertion randomly to create a new solution from the initial solution created above. The next step is to check the feasibility of this newly generated solution. "**CheckNewSol()**" function is implemented after each CreateNeighbor() step to check the feasibility, using the constraints described before. If the solution is feasible it goes to the next step in the algorithm otherwise it goes back to the CreateNeighbor step and these two process keeps on running until a new feasible solution is generated.

After a new solution is generated it is passed into "**MyCost()**" function where the cost of this newly created solution is calculated and then the SA continues as usual.

Results and Analysis

The SA algorithm was tested for various conditions for example varying the transfer passengers, changing the SA conditions such as initial temperature and damping rate. The results are summarized below.

First the algorithm was tested for convergence, the first run was for a 1000 iterations and 20000 NFE's we can see that after 8000 or so NFE's the results doesn't improve significantly and hence the stopping criteria for the rest of the tests were set to 8000 NFE's



Conditions:

MaxIt: 1000

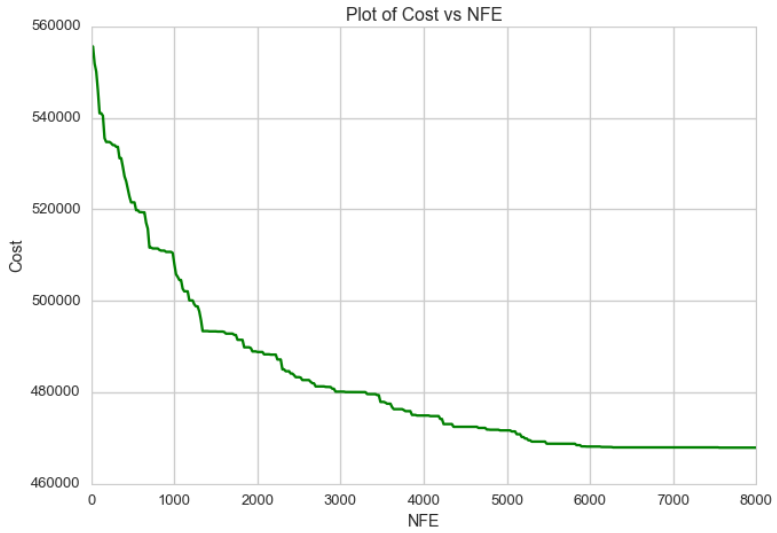
MaxIt2: 20

Transfer Rate: 0%

T0: 100

Damping Rate: 0.99

- **Test on 0% Transfer Passenger**



Conditions:

MaxIt: 400

MaxIt2: 20

Transfer Rate: 0%

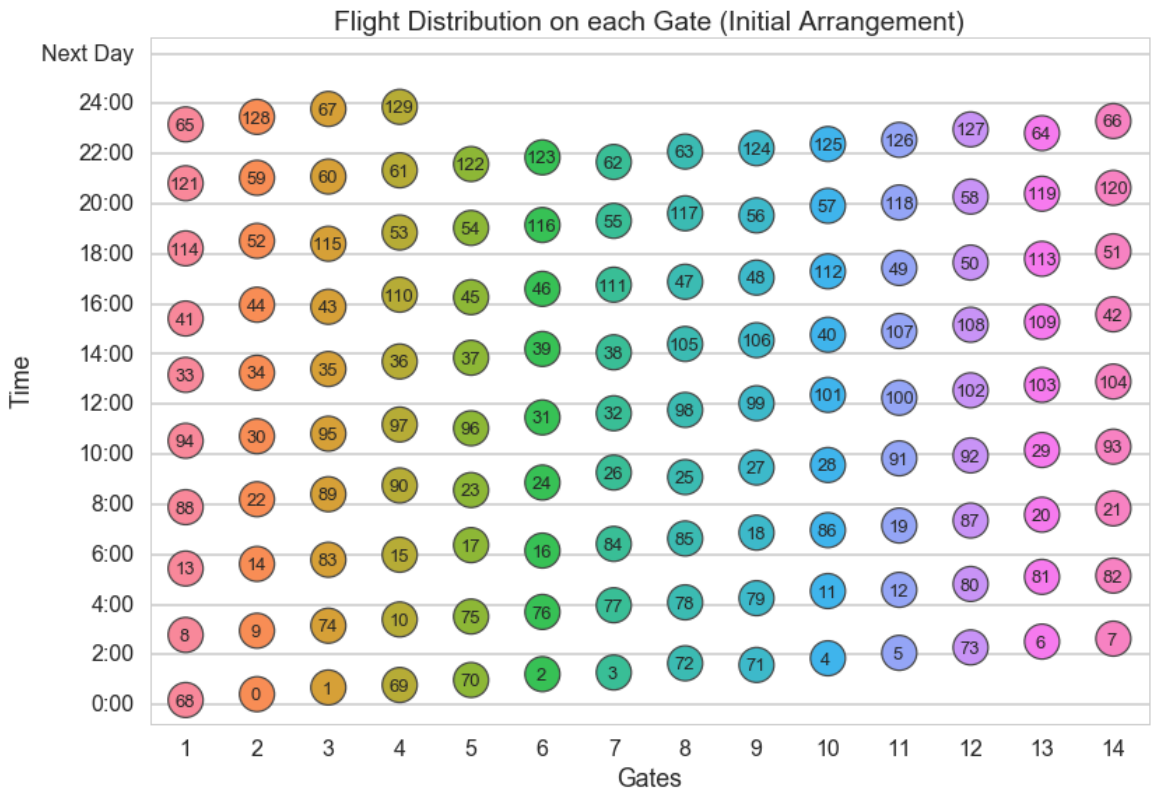
T0: 10

Damping Rate: 0.98

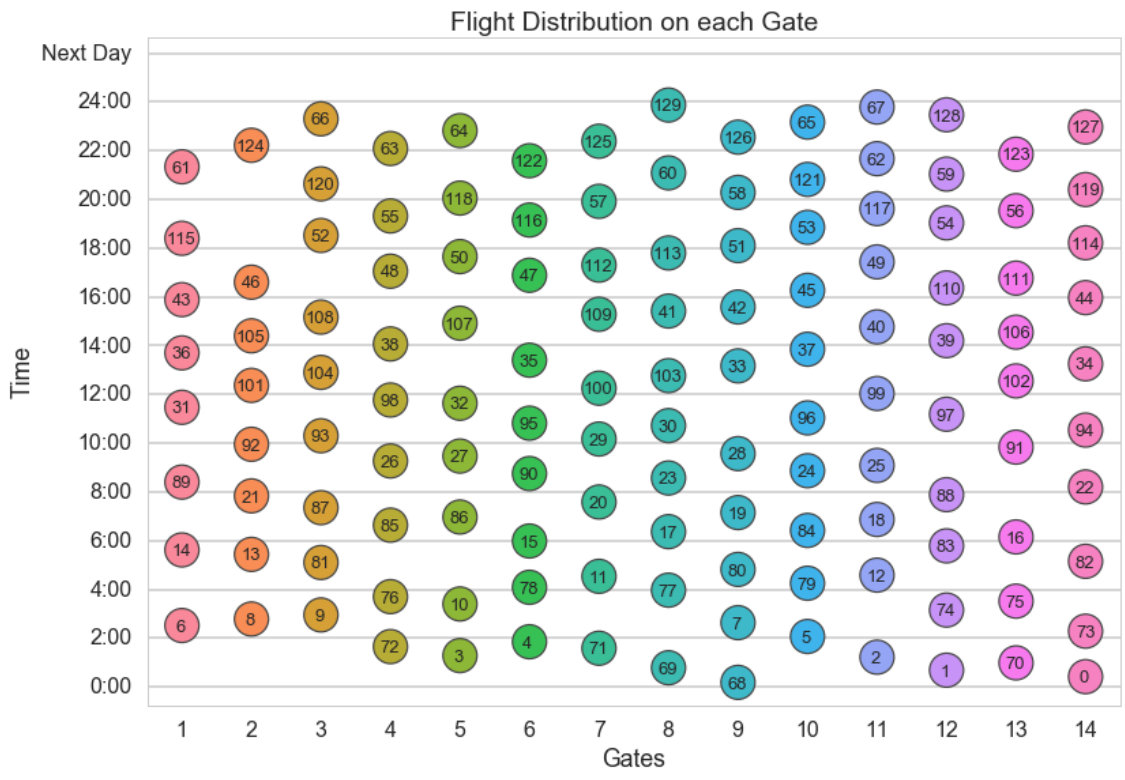
Initial Cost: 555,574
Final Cost: 467,890

We can see that the cost decreased by a factor of 88,000. On the next page the before and after arrangement of flights at their respective gate are shown.

Start of Algorithm (Intial Distributions of Flights)

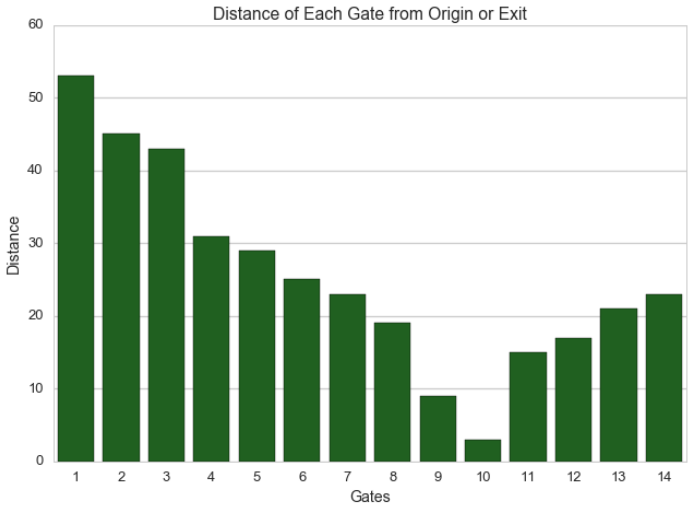


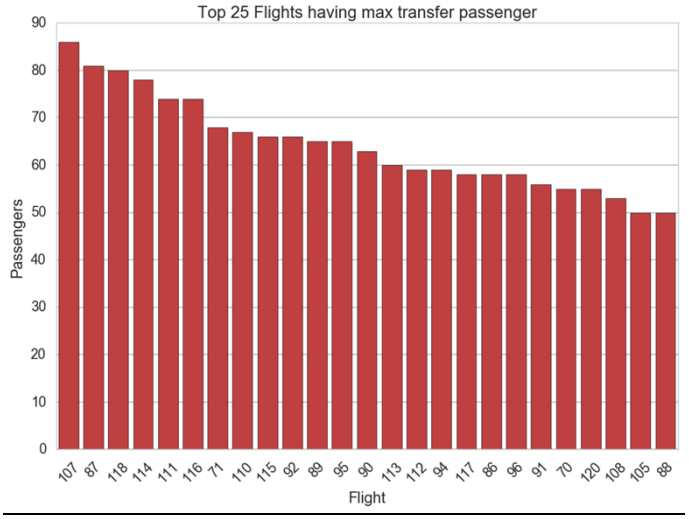
After Completion of Algorithm (Final Distribution of Flights)



(The numbers in circles are the flight numbers. On the Y-axis the arrival time of each corresponding flight is shown and on the X-axis the gate number is shown at which the flight will be stationed)

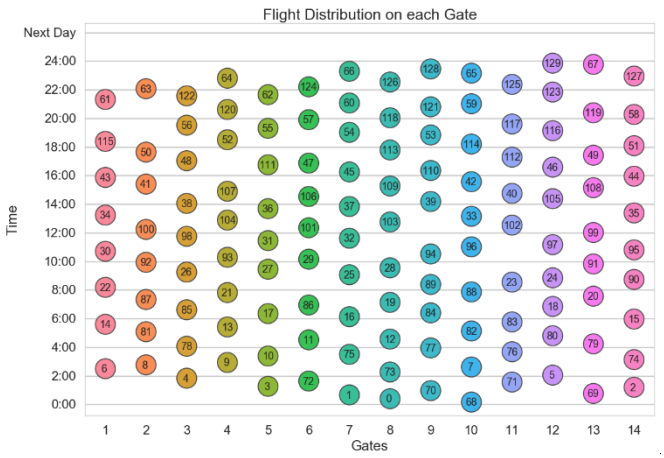
The next two plots are, the distance of each gate from Origin/Exit and top 25 flights which are receiving the highest number of transfer passengers from different flights. In the first plot we can see that Gate 10 has the minimum distance from origin/exit point and Gate 1 has the maximum distance. This is one of the reasons we see that most of the flight were assigned near Gate 10 and flight near Gate 1 were less in number.

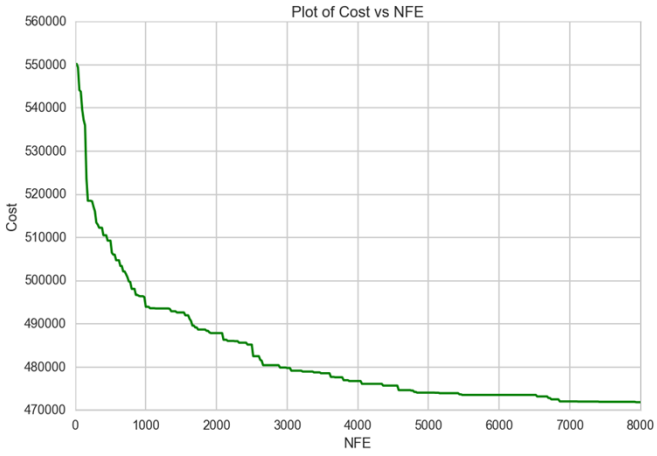




- **<u>Test on 25% Transfer Passengers</u>**

  To test the results, similar conditions for SA were kept, but this time the initial cost of solution was Cost: 557,975 which dropped to - Cost: 471,876. Below are the results of in a graphical format.





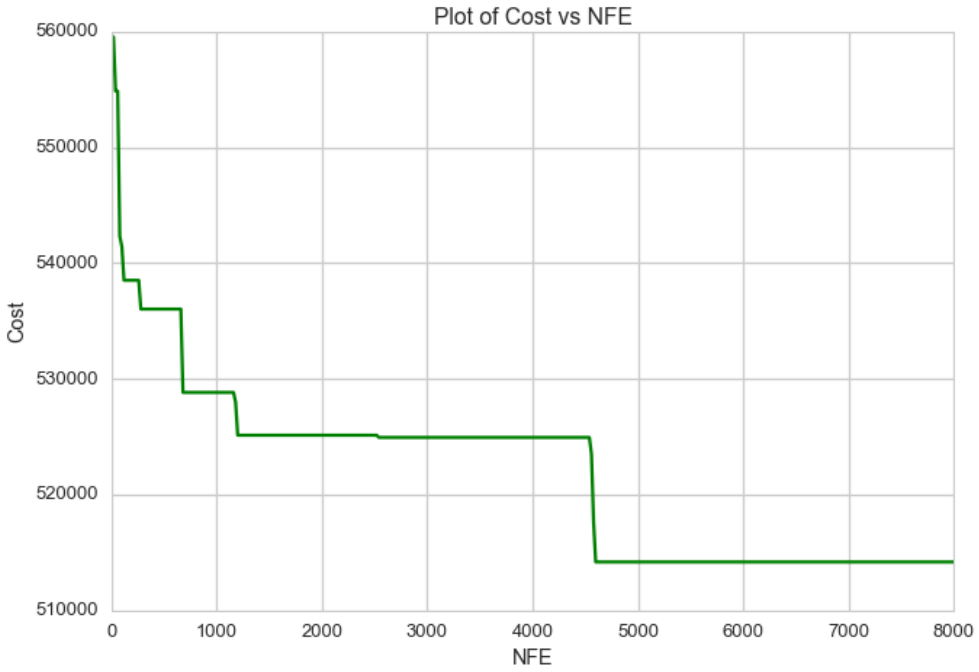
- **<u>Test on 75% Transfer Passengers</u>**

Similar results were generated for the transfer passenger corresponding to at most 75% for each flight. The initial cost was dropped from 554,901 to 460,111.

Next the algorithm was tested for different SA conditions like increasing/decreasing the initial temperature and the damping rate. Case1 and Case2 shows the effect of changing the SA parameters.

Case 1

The initial temperature was raised to 100,000 and the damping rate was kept at 0.9999999. This means the temperature will be decreased extremely slowly and hence the algorithm will be in an exploration phase for very long time. Doing this gave poor results as we kept the NFE to 8000. The Cost vs NFE graph is shown below.



Conditions:

MaxIt: 400

MaxIt2: 20

Transfer Rate: 25%

T0: 100,000

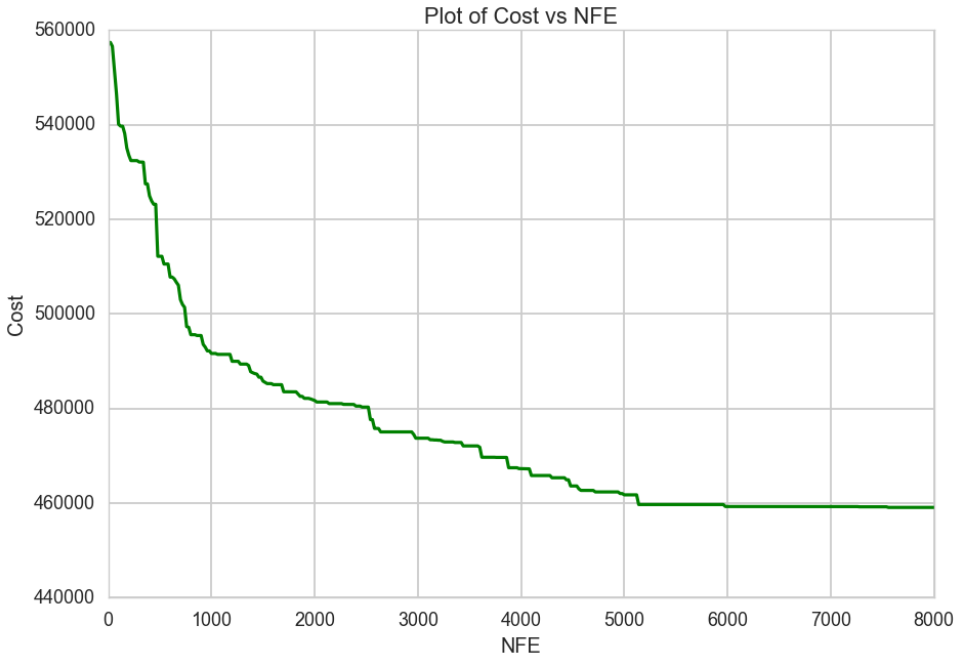
Damping Rate: 0.9999999

Initial Cost: 557,975

Final Cost: 514,181

Case 2

This time the initial temperature was raised to 10,000 but the damping rate was decreased and kept at 0.0009. This means that now temperature will be decreased extremely fast and hence the algorithm will be in exploitation phase very early. Doing this gave better results than Case1. The Cost vs NFE graph is shown below.



Conditions:

MaxIt: 400

MaxIt2: 20

Transfer Rate: 25%

T0: 10,000

Damping Rate: 0.0009

Initial Cost: 557,975

Final Cost: 459,101

Time Complexity

All the solutions generated above are time intensive. The time for which the algorithm run ranges from 2.5 hours to 10 hours, provided the NFE is kept at 8000. This high runtime is depended upon the following reasons.

1. After creating a new neighbor, the solution is checked for feasibility and if the constraints are not satisfied then again a new neighbor is created. The process is repeated until a feasible solution can be generated. This can take a significant amount of time.
2. The time for which the algorithm runs very much depends upon the number of transfer passengers. As the number of transfer passenger increases the cost function goes through additional loops to calculate the cost of the passengers being transferred from each flight. This also added up to the high runtime of the algorithm.

All the above results are summarized in this table

Number	*Difference between Initial and Final Cost*	*Transfer Passengers*	*Time (hours)*	*Initial Temperature for SA*	*Damping Rate for Temperature*
1	87,700	0%	2.5	10	0.98
2	95,000	75%	10	10	0.98
3	86,000	25 %	8.2	10	0.98
4	43,800	25%	8.2	100,000	0.9999999
5	98,300	25%	8.2	10,000	0.0009

NFE was kept 8000 for all the tests

The best results were generated when the initial temperature for SA was kept high and then the temperature was decreased very fast.

Future works

This project was based on minimizing the passenger walking distance to assign a gate to flights, which was a single objective meta-heuristics in itself. This problem can be transformed into a multi objective heuristics problem by accommodating more objective and constraints. For example, a gate can be said to handle only a specific number of flights or type of flight. As in the real world there are different size of aircraft and hence have different requirements at the time of gate assignment.

As we could not get our hands on the actual data, we had to simulate some of the data to be used in the algorithm. In the future this algorithm can be tested on a more realistic data which might have variations in number of passengers, arriving and departing time of aircrafts, etc. This may give an interesting result and the results might be useful in the industry.

Last but not least, a few assumptions might be removed and actual conditions may be incorporated in the algorithm such as the delay in a flights arrival or departure time may be considered, the buffer time between the aircraft's departure and the next aircraft's arrival at the same gate may be added as a constraint in the algorithm.

References

[1] Chun-Hung Cheng, Sin C.Ho & Cheuk-Lam Kwan (2012). The use of meta-heuristics for airport gate assignment. *In Expert Systems with Applications 39 (2012) 12430-12437.*

[2] H.Ding, A.Lim, B.Rodrigues & Y.Zhu (2004). Aircraft and Gate Scheduling Optimization at Airports. *In Proceedings of the 37th Hawaii International Conference on System Sciences – 2004*.

[3] Jiefeng Xu & Glenn Bailey (2001). The Airport Gate Assignment Problem: Mathematical Model and a Tabu Search Algorithm. *In Proceedings of the 34th Hawaii International Conference on System Sciences – 2001*.

[4] Bolat, A. (2001). Models and a genetic algorithm for static aircraft-gate assignment problem. *Journal of Operations Research Society, 52(10), 1107 – 1120.*

[5] Hu, X. B., & Di Paolo, E. (2009). An efficient genetic algorithm with uniform crossover for the multi-objective airport gate assignment problem. In C. K. Goh, Y. S. Ong, & K.C. Tan (Eds.). *Multi-objective memetic algorithms (vol. 171, pp. 71-89). Berlin Heidelberg: Springer-Verlag.*

[6] https://en.wikipedia.org/wiki/Logan_International_Airport - Boston Logan Airport Wikipedia page

[7] https://www.massport.com/media/414776/1016-avstats-airport-traffic-summary.pdf – Data for Boston Logan Airport