University of Waterloo
Faculty of Engineering
Department of Electrical and Computer Engineering

Solving Fleet Scheduling Problem using Metaheuristic Algorithms

University of Waterloo
200 University Avenue West
Waterloo, ON, N2L 3G1, Canada

Advisor: Dr. Alaa Khamis
Prepared by: Group - 2015.002
Yash Malik - 20379044 - ygmalik
Satyam Gupta - 20370860 - s52gupta
Piyush Gadigone - 20366910 - pgadigon
Khushi Sohi - 20396220 - ksohi
Sugandha Sharma - 20401693 - s72sharm
4A Computer Engineering
29 July 2014

Table of Contents

List of Figures

List of Tables

Summary

Aircraft Fleet scheduling is at the core of enabling growth, increasing revenue, maintaining customer and employee satisfaction in the airline industry. Every airline thus aims to optimize its airline schedule in a way to make optimal use of resources while minimizing cost and maximizing profit. Many strategies exist to minimize the operational costs, one such strategy is called the fleet assignment problem. The main purpose of the project is to analyse the fleet assignment problem using adaptive algorithms and the knowledge of artificial intelligence. The aim here is to optimize the problem of assigning aircrafts to routes in order to meet with demands and minimize costs.

The major points covered in the report are literature review, problem formulation and modeling which includes how the inputs to the problem are articulated, overview and hand iterations for Tabu Search, Simulated Annealing, Genetic Algorithm, Particle Swarm Optimization and Ant Colony Optimization. In addition to this, a performance evaluation is also done based on evaluation metrics of optimality of solution, iterations required to converge, total time of execution, and time per iteration.

The major conclusions of the report are that Genetic Algorithm gives the most optimal solution closely followed by Particle Swarm Optimization. Tabu Search converges the fastest due to the simplicity of the algorithm. Also, although Genetic algorithm takes a large number of iterations to converge to the most optimal solution, the CPU time that it takes per iteration is much smaller compared to other algorithms. Simulated Annealing was found to lie in different ends of the spectrum across the evaluation metrics. This approach had strength in Optimality of cost and CPU time per iteration, but underperformed in Number of iterations and Average time to converge. Experimental studies showed that ACO was unable to converge to near optimal solutions unlike the other metaheuristic approaches, and faired as one of the least desirable algorithms in most of the other evaluation metrics.

The major recommendations of the report are that the solutions provided by the algorithms can be improved by adding cooperative and adaptive behavior. Additionally, the time taken to converge can be reduced by leveraging the multi-threading libraries in matlab to parallelize the work in an iteration and increase cooperation in the algorithms.

1. Introduction

Air travel remains one of the largest and most rapidly growing industries across the world. Airlines operate in a very competitive market that serves a huge customer base. Airline schedule plays an indispensable role in determining the level/quality of service an airline offers to its customers which further determines their market share. It is thus considered as an area of core focus when formulating an airline's business strategy. The main goal of any airline's business plan to optimize the deployment of the airline's resources in order to meet with demands and to maximize their profit. There can be many strategies to achieve this goal, like keeping large profit margins on travel costs or keeping the airline's operational costs to the minimum. Again, there can be many strategies to minimize the operational costs and one such strategy is Fleet Assignment planning.

In the context of airline scheduling, Fleet Assignment Planning is the assignment of the most appropriate aircraft to each route. However there are many restrictions that need to be considered for this Fleet Assignment Planning, which makes it an extremely challenging task. One such challenge is that the flying performance of every aircraft model is different, for example, voyage range, flying altitude ceiling, maximum takeoff weight, and climbing ability [1]. Thus, one aircraft model may be suitable for a particular route and not others. Furthermore, different aircraft models have different seating layout and operating costs. For example, the A340-200 aircraft can seat up to 380 people, and its operating cost is about 100k RMB per hour, juxtaposed to B737-300, which seats 144 passengers and operates at a cost of about 40k RMB per hour [1]. Keeping the complexity of restrictions in mind, it can be of advantage to come up with a Fleet Assignment plan using adaptive algorithms. This report covers an analysis of the Fleet Assignment problem using five different algorithms. These are Tabu Search, Simulated Annealing, Genetic Algorithms, Particle Swarm Optimization and Ant Colony Optimization. The factors or constraints considered for designing the algorithms are capacity of the aircraft, fuel costs per mile, passenger demands for destinations and distances between origin and destinations.

2. Literature review

This section gives a brief overview of some research papers in the Flight Scheduling problem that were studied for this report. Fleet Assignment planning and flight scheduling have been a subject of research for decades. Earlier, the models were usually formulated as integer network flow problems aiming at either the minimization of operation costs or at the maximization of profits [2]. One such model was developed at Massachusetts Institute of Technology by Anataram Balakrishnan et al [3]. In their research paper they define the aircraft routing problem as a mixed integer program. The program encapsulates the important profit generating factors in the route selection decision and uses Lagrangian based procedure for finding solutions. The research paper concludes that this algorithm is able to select only few candidate routes with precision [3]. Due to the inefficiency of these algorithms, recently, researchers have been using more adaptive algorithms to study the flight scheduling problem. One such research paper by Xue-Feng Zhang shines light on using a new hybrid swarm optimization algorithm for optimizing any scheduling problem [4].

This hybrid algorithm is based on Particle swarm optimization, Tabu Search and Simulated Annealing. In the paper it is argued that by reasonably combining these three algorithms a robust and fast algorithm can be made, that gives higher quality solutions within lesser computation time.

3. Problem Formulation and Modeling

3.1. Problem statement

This fleet assignment problem deals with the question of how to assign *n* routes to *n* aircrafts of *m* model types. The ultimate objective is to optimize the allocation of aircraft models to routes, in order to minimize the cost, and is proposed as follows:

$$min \sum_{i=1}^{n} \sum_{j=1}^{m} c_{ij} x_{ij}, \quad (1)$$

where,
$i = 1, 2, \ldots, n$; n is the route number,
$j = 1, 2, \ldots, m$; m is the number of aircraft model types,
c_{ij} is the cost associated with assigning aircraft model j to route i.
x_{ij} is 1 if route i is assigned to model j, and 0 otherwise.

The cost, c_{ij} of a route, is dependent on the distance of the route, the fuel consumption of the aircraft and the demand associated with the route.

In addition, the following set of constraints are imposed:

$$\sum_{j=1}^{m} x_{ij} = 1 \quad (2)$$

$$\sum_{i=1}^{n} \sum_{j=1}^{m} x_{ij} = n \quad (3)$$

$$x_{ij}\, capacity_j \geq Demand_i \quad (4)$$

The objective function (1) ensures that once the *n* routes have been assigned to their corresponding aircrafts, the final cost is minimized. Constraint (2) ensures that one route is assigned to exactly one aircraft model type. Constraint (3) ensures a linear assignment, where each route is associated to one aircraft. Lastly, constraint (4) ensures that the route assigned to a particular aircraft is attainable by that aircraft. Here, $capacity_j$ is the passenger and freight capacity of model *j* and $Demand_i$ is the average traffic and freight demand of route i.

An important assumption to note here is that the *n* flights are scheduled concurrently, and to serve the *n* flights, there are *n* aircrafts. Note that this is not a very practical assumption, for example, US Airways typically flies about 2,500 jet flights to over 100 domestic, Caribbean and European markets using more than 400 aircrafts of 14 different types [5].

3.2. Problem modeling

The abstract mathematical modelling technique that is applied to the fleet scheduling problem is motivated from the linear assignment problem. Broadly speaking in the assignment problem there are a number of *agents* and *tasks* such that any *agent* must be assigned to a *task* but incurs a cost in the process on the agent-task assignment. Furthermore, it is known that each *task* may be assigned to at most one *agent* and an *agent* may not be assigned on more than one task [6].

Formally, let us define two sets A and T of equal size such that their bijective mapping *f* from A to T. More concretely, in the fleet assignment problem, elements in set A represent n aircrafts which may be one of m unique aircraft model types. Elements in set T represent one of m pairs of origin-destination cities that needs an assignment of an aircraft to be travelled across. There is a cost c_{ij} associated with assigning an aircraft model to a flight route in T [6].

Let us define G_ϕ = (A, T, E): bipartite graph, where the vertex sets A and T both have n elements (Section 1.2). There is an edge (i, j) ε E iff = $\phi(i)$. Now, ϕ: every permutation of the set N = {1, . . . , n}. For example, Figure 1 illustrates a possible ϕ where there are 3 flights to be scheduled.

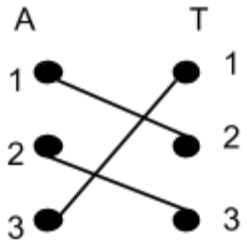

Figure 1: Bipartite Graph

The pictorial representation in Figure 1, is denoted as follows - (1, 2; 2, 3; 3, 1).

It is also possible to create an adjacency matrix for the graph G_ϕ. This adjacency matrix is denoted by X_ϕ. $X_\phi = (x_{ij})$ with $x_{ij} = 1$ for $j = \phi(i)$ and $x_{ij} = 0$ for $j \neq \phi(i)$. X_ϕ is in matrix form shown below:

$$\begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{pmatrix}$$

4. Proposed Solutions

4.1. Simple problem description

For the sake of discussion, assume the following scenario for the remainder of section 4:

At a local airport of Cairo, Egypt, in a given time interval, there are 7 flights that need to be scheduled. The 7 flights include 2 Boeing 747 models, 3 Boeing 703 models, and 2 Boeing T-43 models. Table 1 summarizes the attributes related to each aircraft model type, and Table 2 summarizes the attributes related to each route.

Table 1: Attributes of different aircraft models.

Model Type	Aircraft Numbers	Seating capacity	Cost per mile
747	1, 2	450	10
703	3, 4, 5	150	4
T-43	6, 7	240	6

Table 2: Summarizes the attributes related to each route.

Destination	Route number	Distance (mi)	Demand (passengers)
Sydney, Australia	1	8,950	541
Istanbul, Turkey	2	1,700	201
Athens, Greece	3	2,100	193
Aswan, Egypt	4	400	328
Alexandria, Egypt	5	100	73
Amman, Jordan	6	308	196
New Delhi, India	7	2,751	447

4.2. Inputs

Given the information in section 4.1, the cost for a particular route *i* being assigned to aircraft of model type *j* is given as follows:

$$Cost_{ij} = Demand / Capacity\ of\ aircraft\ type_j \times Cost\ per\ mile_j \times Distance_j$$

The cost matrix in Table 3 can be generated from the simple problem description described in Section 4.1. Each row represents an aircraft model type, and each column represents a destination route.

Table 3: Sample cost matrix

Type \ Route	1	2	3	4	5	6	7
Airbus 747	179000	17000	21000	4000	1000	3080	27510
Airbus 703	143200	13600	16800	1600	400	2464	33012
Airbus T-42	161100	10200	12600	2400	600	1848	33012

4.3. Initial Solution

An initial solution can be generated by computing a random permutation of the *n* routes.

1	2	3	4	5	6	7
[3	2	4	5	6	7	1]

This row vector represents a valid permutation of the route numbers. The index of each element in this initial solution vector is a flight number, which maps to a possible aircraft model type. The total cost of the above initial solution can be calculated by using the information from Table 1 and Table 3 as follows:

$$Cost = Route\ 3\ cost\ with\ Aircraft\ number\ 1 + Route\ 2\ cost\ with\ Aircraft\ number\ 2 + \ldots$$
$$+ Route\ 1\ cost\ with\ Aircraft\ number\ 7$$
$$Cost = 21000 + 17000 + 1600 + 400 + 2464 + 33012 + 161100 = 236576$$

A neighbouring solution can be obtained by performing a swap of route numbers in the solution vector. For example, if route 2 and 5 are swapped a new neighbouring solution is obtained as follows:

[3 5 4 2 6 7 1]

A suitable strategy to get an optimal solution to this problem is to try all permutations of flight numbers and find the minimum cost from all the permutations. This however, is an NP-hard problem as the computational time increases exponentially with the number of routes. For *n* routes, the runtime to calculate all possible permutations is $O(2^n)$.

4.4. Tabu Search

4.4.1. Overview

The overview of the implementation for Tabu Search is as follows:

Generate an initial solution which is a random permutations of all *n* route numbers.
Initialize a *n* by *n* matrix with zeros. This matrix represents the Tabu Table.

Initialize the number of iterations and the Tabu Length.
For each iteration

Swap the pair that yields the lowest cost by looking at all possible swaps. Perform this swap only if it is not already Tabued.
Update the current solution to this new solution and mark it in the Tabu Table.
Decrement the positive values in the Tabu Table.
Update the global best cost if the new solution has a lower cost.
Adaptively decrease the Tabu Length if the cost from the previous iterations does not have any improvements.

End loop

4.4.2. Hand Iterations

Iteration 0

Current Solution: [3 2 4 5 6 7 1]
Current Cost: 236576 (using the method described in Section 4.3)
Global Best Solution: [3 2 4 5 6 7 1]
Global Best Cost: 236576
Tabu Length: 5 (chosen randomly and performs well)

Tabu table

0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0

Iteration 1

Looking at all possible swaps, the swap that gives the least cost is calculated to be a swap between route 5 and 7. The variables are now updated as follows:

Current Solution: [3 2 4 7 6 5 1]
Current Cost: 218060
Global Best Solution: [3 2 4 7 6 5 1]
Global Best Cost: 218060
Tabu Length: 5

Tabu table

0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	5
0	0	0	0	0	0	0
0	0	0	0	5	0	0

Iteration 2

Looking at all possible swaps, the swap that gives the least cost is calculated to be a swap between route 3 and 6. The variables are now updated as follows:

Current Solution: [3 2 4 7 1 5 6]
Current Cost: 204158
Global Best Solution: [3 2 4 7 1 5 6]
Global Best Cost: 204158
Tabu Length: 5

Tabu table

0	0	0	0	4	0	0
0	0	0	0	0	0	0
0	0	0	0	0	5	0
0	0	0	0	0	0	0
4	0	0	0	0	0	0
0	0	5	0	0	0	0
0	0	0	0	0	0	0

4.5. Simulated Annealing

4.5.1 Overview

The overview of the implementation for Simulated Annealing for the linear fleet assignment is as follows:

Initialize initial temperature T_o and an initial solution (random permutation of routes).
Set global best solution to initial solution
Set final temperature T_o and max number of iterations N
While ($T > T_f$ and $n < N$)
    for iterations at each temperature
        if (no change in best solution observed over several iterations)
            break;
        end if
        Swap two routes randomly to generate a new solution
        Calculate the new cost
        Accept the new solution if lower cost
        if cost is higher
            Generate a random number r
            Accept if $p = \exp(-\Delta \textit{ new solution } / T) > r$
        end if
        Update global best solution
        Update iteration: n = n + 1
    end
    Update temperature: temperature = temperature * alpha
End while

4.5.2. Hand iterations

The following parameters are used for the hand iteration: T = 110 and $T_f = 0.5$, which is a reasonable value for a problem of this scale N = 1000, and alpha = 0.99.

Iteration 0

From section 4.3, let the initial solution be:

x_0 = [3 2 4 5 6 7 1]
c_0 = 236576

Using the technique defined in section 4.3, the cost can be calculated to be $C_0 = 212774$.

Initially,

$x_{best} = x_0 =$ [3 2 4 5 6 7 1]
$c_{best} = c_0 = 236576$

Iteration 1

Randomly select two numbers r_1 and r_2 between 1 and 7 to swap. Let $r_1 = 4$ and $r_2 = 6$. The following is the effect of the swap:

x_1 = [3 2 4 7 6 5 1]
$c_1 = 218060$

$c_{best} < C_1$, accept solution and update C_{best}

$c_{best} = 218060$
x_{best} = [3 2 4 7 6 5 1]

Update temperature: $T = T \times \alpha = 110 * 0.99 = 108.9$
Move onto iteration 2.

Iteration 2

Randomly select two numbers r_1 and r_2 between 1 and 7 to swap. Let $r_1 = 5$ and $r_2 = 7$.
The following is the effect of the swap:

x_2 = [3 2 4 7 1 5 6]
$c_2 = 204158$

Similar to iteration 1, because this solution is an improving solution, it is accepted and the global best is updated.

x_{best} = [3 2 4 7 1 5 6]
$c_{best} = 204158$

Update temperature: $T = T \times \alpha = 108.9 * 0.99 = 107.8$
Move onto iteration 3.

4.6. Genetic Algorithm

4.6.1. Overview

The overview of the implementation for Genetic Algorithm for the Linear Assignment model to the Fleet Scheduling problem is as follows:

Initialize *pop_size, min_num_iters, sd_thresh, pop, iqr, sd*
Create *initial population*
Evaluate *all individuals*
While (*iqr ~= 0)* ***or***
*(*sd > sd_thresh) **or**
(num_iters_completed < num_iters)
Get fitness of current **pop**
Compute selection_prob, cumulative_prob given **fitness** of current **pop**
Get four parents given current **pop** and **cumulative_prob**
Apply Partially mapped crossover
Set child_one = *Crossover(* parent_one , parent_two)
End Crossover
Apply Swap mutation
Set child_three = *Mutation* (parent_three)
Set child_four = *Mutation* (parent_four)
End Mutation
Apply Steady State model for replacement of population
Get four least fit individuals from the population
Set child_one = parent_one, child_two = parent_two
Set child_three = parent_three, child_four = parent_four
End Replacement
Compute iqr (**pop**)
Compute sd (**pop**)
Update num_iters_completed
End While

4.6.2. Hand Iterations

Iteration 0

Randomly initialize a population of size **pop_size** = 4.
Initial Population:
[4 3 6 5 2 7 1
1 4 6 5 3 2 7
7 4 5 6 2 3 1
2 6 4 5 7 1 3]

Compute the cost matrix for the population using the method described in Section 4.3.
Cost of population:
[235576 245876 221674 228792]

Iteration 1

Compute Fitness proportionate Selection vector.
Fitness vector:
[3.9559 3.7901 4.2040 4.0732]

Compute a cumulative probability vector for input to the Roulette Wheel Algorithm to aid in parent selection.
Cumulative Probability vector:
[0.2468 0.4834 0.7457 1.0000]

Parent Selection:
Get two parents from the current population to perform mutation.
Parent_one:
[7 4 5 6 2 3 1]
Parent_two:
[4 3 6 5 2 7 1]

Swap Mutation:
Perform swap mutation to build children corresponding to the two parents.
Child_one:
[7 1 5 6 2 3 4] i.e. swap between indices 2, 7
Child_two:
[4 1 6 5 2 7 3] i.e. swap between indices 2, 7

Population replacement:
Individual 3 from the original population which was the least fit is replaced with the mutated **child_two**.
[4 3 6 5 2 7 1
4 1 6 5 2 7 3
7 4 5 6 2 3 1
2 6 4 5 7 1 3]

Iteration 2

Compute Fitness proportionate Selection vector.
Fitness vector:
[3.9525
3.7993
4.2003
4.0697]

Compute a cumulative probability vector for input to the Roulette Wheel Algorithm to aid in parent selection.

Cumulative Probability vector:

[0.2466
 0.4838
 0.7459
 1.0000]

Get two parents from the current population to perform partially mapped crossover.

Parent selection:

Parent_one:

[2 6 4 5 7 1 3]

Parent_two:

[4 1 6 5 2 7 3]

Select two crossover points (indices 4 and 6 in the parent) at random and copy the segment from **parent_one** to **child**. Generate the rest of the offspring based on the GA permutation algorithm.

Partially mapped crossover:

[4 6 2 5 7 1 3]

Individual 2 from the population in Iteration 1 which was the least fit is replaced with the offspring from the crossover.

Population replacement:

[4 3 6 5 2 7 1
 4 6 2 5 7 1 3
 7 4 5 6 2 3 1
 2 6 4 5 7 1 3]

4.7. Particle Swarm Optimization

4.7.1. Overview

The overview of the implementation for Particle Swarm optimization is as follows:

Initialize particles with solutions randomly and set their velocities to 0
While (n < N)
    For each particle
        Calculate the cost for each particle using the given cost function
        Calculate velocity for each particle, denoting the number of swaps for each particle.
        Update each particle by performing the designated number of swaps, and with each swap, moving a solution close to its neighbor.

Update the global best

end

end

The velocity of a particle for this linear assignment permutation PSO problem is defined as the number of swaps performed on that particle. The worse the performance of a particular particle, the higher its velocity, and vice-versa. The intuition behind this is that if a particle is performing bad, it is further away from the optimal solution, and thus requires more changes. The velocity for particle k after iteration t is defined as follows:

$$v_k = Ceil(V_{MAX} \times (Cost_k / Cost_{Best\ particle\ in\ iteration\ t}))$$

where V_{MAX} is a variable input defined by the user.

Each particle k is updated at the end of each iteration by performing random swaps v_k number of times. Furthermore, after each swap, particle k is updated to have routes similar to that if its better performing neighbor.

4.7.2. Hand iterations

The following parameters are used for the hand iteration:
Number of particles = 2, V_{MAX} = number of routes to schedule = 7.

Iteration 0

Randomly initialize two particles P^1 and P^2 and their velocity V^1 and V^2.

$P_0^1 = [\ 3 \quad 2 \quad 4 \quad 5 \quad 6 \quad 7 \quad 1\ ],\ C_0^1 = 236576,\ v_0^1 = 0$
$P_0^2 = [\ 1 \quad 7 \quad 4 \quad 2 \quad 6 \quad 5 \quad 3\ ],\ C_0^2 = 308060,\ v_0^2 = 0$

Iteration 1

Cost of each particle at this iteration is given by:
$C_1^1 = 236576,\ C_1^2 = 308060$

The velocity for each particle can be calculated as follows:
$v_1^1 = ceil\ (7 \times 236576 / 308060) = 6$
$v_1^2 = ceil\ (7 \times 308060 / 308060) = 7$

Update P_1 and P_2 by performing 6 and 7 random swaps on each, respectively. After each swap, make a change to P^1 (poor performing neighbor), to make it similar to P^2. Let the following denote the result of these swaps:

$$P_1^1 = [\ 3 \quad 5 \quad 4 \quad 1 \quad 6 \quad 7 \quad 2\ ]$$
$$P_1^2 = [\ 1 \quad 4 \quad 3 \quad 5 \quad 2 \quad 6 \quad 7\ ]$$

Note that as a consequence of moving P^2 to be closer to its neighbor, route 6 is beside 7 and 3 is beside 5 in both of them.

Move onto iteration 2.

Iteration 2

Cost of each particle at this iteration is given by:
$C_2^1 = 211030,\ C_2^2 = 304910$

The velocity for each particle can be calculated as follows:
$v_2^1 = ceil\ (7 \times 211030 / 304910) = 5$
$v_2^2 = ceil\ (7 \times 304910 / 304910) = 7$

Using the same update strategy as iteration 1, generate new solutions:

$$P_2^1 = [\ 1 \quad 4 \quad 2 \quad 3 \quad 7 \quad 5 \quad 6]$$
$$P_2^2 = [\ 1 \quad 6 \quad 7 \quad 5 \quad 3 \quad 2 \quad 4]$$

Move onto iteration 3.

It is important to note that the first few iterations of this algorithms perform rigorous swaps, but ultimately, these particle solutions converge to the same value due to the nature of the swaps, which are semi-random, but also have a social component.

4.8. Ant Colony Optimization

4.8.1. Overview

Ant Colony System (ACS) algorithm is used for this problem. The overview of the implementation for the algorithm is as follows:

Initialize the number of ants, evaporation rate, the pheromone deposit value, the pheromone matrix, alpha, beta and r_o.
Initialize the initial solution for each ant by selecting a random permutation of routes.
While (n < N)
    For each ant
        Initialize an empty ant solution vector.
        Generate a random value between 0 and 1. If this value is less than r_o, select the best edge and proceed to the Evaporate step. Else, continue to the next step.
        Generate cumulative probabilities for selecting the next route values. The higher the pheromone value, the higher is the chance of selecting a particular route. This selection of the route is determined by generating a random number between 0 and 1 and using the roulette wheel strategy.
        Evaporate the pheromone by multiplying the pheromone matrix by (1 - evaporation rate)
        Deposit pheromone on the route taken by the ant. The desirability is set to be Q/cost where Q is a constant.
    End loop
    Adaptively decrease the evaporation rate and increase the r_o value to intensify more with increase in iteration number.
    Update the global best cost, the iteration number and the iqr value.
End loop

4.8.2. Hand Iteration

Iteration 0

The following variables are initialized at the beginning of the algorithm:

Evaporation Rate: 0.9 (Have high evaporation rate initially to allow more exploration)
r_o = 0.01 (Have small r_o rate initially to allow more exploration)
Number of ants: 2

Initially, all edges have the same pheromone value of $\tau = 0.5$. If one of the solutions for the ant is [3 5 4 1 6 7 2], pheromone is deposited in the pheromone matrix at points (S, 3), (3,5), (5,4) and so on. Note that 'S' stands for starting point.

Pheromone Matrix:

Source \ Destination	S	1	2	3	4	5	6	7
S	0	0.5	0.5	0.5	0.5	0.5	0.5	0.5
1	0	0	0.5	0.5	0.5	0.5	0.5	0.5
2	0	0.5	0	0.5	0.5	0.5	0.5	0.5
3	0	0.5	0.5	0	0.5	0.5	0.5	0.5
4	0	0.5	0.5	0.5	0	0.5	0.5	0.5
5	0	0.5	0.5	0.5	0.5	0	0.5	0.5
6	0	0.5	0.5	0.5	0.5	0.5	0	0.5
7	0	0.5	0.5	0.5	0.5	0.5	0.5	0

Iteration 1

Evaporation Rate: 0.9 (Have high evaporation rate initially to allow more exploration)
$r_o = 0.01$ (Have small r_o rate initially to allow more exploration)

$N_1^1 = \{ 1, 2, 3, 4, 5, 6, 7 \}$ where N is the set of routes that can be selected

$\sum_{n \varepsilon N_1^1} \tau_{mn}/d_{mn}$, where τ_{mn} is the pheromone deposit at point (m,n) and d_{mn} is the cost of assigning a route to flight number m and route n.

Generate Ant Solutions:

Generate r. Assume it is 0.6. Because $r > r_0$, explore.

$m = 1$
$\tau_{11} = 0.5, \tau_{12} = 0.5, \tau_{13} = 0.5, \tau_{14} = 0.5, \tau_{15} = 0.5, \tau_{16} = 0.5, \tau_{17} = 0.5$
$d_{11} = 179000, d_{12} = 17000, d_{13} = 21000, d_{14} = 4000, d_{15} = 1000, d_{16} = 3080, d_{17} = 27510$

$\sum_{n \varepsilon N_1^1} \tau_{Sn}/d_{1n} = 0.00086$

$p_{11}^1 = 2.8 * 10^{-6}, p_{12}^1 = 2.9 * 10^{-5}, p_{13}^1 = 2.3 * 10^{-5}, p_{14}^1 = 1.3 * 10^{-4}, p_{15}^1 = 0.5 * 10^{-4},$
$p_{16}^1 = 1.6 * 10^{-4}, p_{17}^1 = 1.8 * 10^{-5}$

Using the roulette wheel strategy with the above probabilities, the first route is selected to be 3.
Using the above strategy, the next 6 flights are assigned to their corresponding routes yielding a solution of following:
$ant_1^1 = [\ 3 \quad 2 \quad 4 \quad 5 \quad 6 \quad 7 \quad 1\ ],\ C_1^1 = 236576$
Similarly, a solution for the second ant is obtained as follows:
$ant_1^2 = [\ 1 \quad 7 \quad 4 \quad 2 \quad 6 \quad 5 \quad 3\ ],\ C_1^2 = 237374$

Pheromone Evaporation:

Since the evaporation rate is 0.9, the entire matrix is multiplied by (1 - 0.9).

Source \ Destination	S	1	2	3	4	5	6	7
S	0	0.05	0.05	0.05	0.05	0.05	0.05	0.05
1	0	0	0.05	0.05	0.05	0.05	0.05	0.05
2	0	0.05	0	0.05	0.05	0.05	0.05	0.05
3	0	0.05	0.05	0	0.05	0.05	0.05	0.05
4	0	0.05	0.05	0.05	0	0.05	0.05	0.05
5	0	0.05	0.05	0.05	0.05	0	0.05	0.05
6	0	0.05	0.05	0.05	0.05	0.05	0	0.05
7	0	0.05	0.05	0.05	0.05	0.05	0.05	0

Pheromone Deposit:

Using Online Delayed Update strategy, a value of $Q/C_1^1 = 0.004$ and $Q/C_1^2 = 0.004$ is deposited on the path taken by the ant. Here Q is initialized to 1000.

Source \ Destination	S	1	2	3	4	5	6	7
S	0	0.054	0.05	0.054	0.05	0.05	0.05	0.05
1	0	0	0.05	0.05	0.05	0.05	0.05	0.054
2	0	0.05	0	0.05	0.05	0.05	0.054	0.05
3	0	0.05	0.054	0	0.05	0.054	0.05	0.05
4	0	0.05	0.054	0.05	0	0.054	0.05	0.05
5	0	0.05	0.05	0.05	0.05	0	0.05	0.05
6	0	0.05	0.05	0.05	0.05	0.054	0	0.054
7	0	0.054	0.05	0.05	0.054	0.05	0.05	0

Iteration 2

Using the strategy described in Iteration 1, two ant solutions are generated again:

$ant_2^1 = [\ 1 \quad 2 \quad 4 \quad 5 \quad 6 \quad 7 \quad 3\ ],\ C_2^1 = 246076$
$ant_2^2 = [\ 1 \quad 6 \quad 4 \quad 2 \quad 7 \quad 5 \quad 3\ ],\ C_2^2 = 243492$

Pheromone Evaporation:

Since the evaporation rate is 0.9, the entire matrix is multiplied by (1 - 0.9).

		Destination						
	S	**1**	**2**	**3**	**4**	**5**	**6**	**7**
S	0	0.0054	0.005	0.0054	0.005	0.005	0.005	0.005
1	0	0	0.005	0.005	0.005	0.005	0.005	0.0054
2	0	0.005	0	0.005	0.005	0.005	0.0054	0.005
3	0	0.005	0.0054	0	0.005	0.0054	0.005	0.005
4	0	0.005	0.0054	0.005	0	0.0054	0.005	0.005
5	0	0.005	0.005	0.005	0.005	0	0.005	0.005
6	0	0.005	0.005	0.005	0.005	0.0054	0	0.0054
7	0	0.0054	0.005	0.005	0.0054	0.005	0.05	0

Source

Pheromone Deposit:

Using Online Delayed Update strategy, a value of $Q/C_1^1 = 0.004$ and $Q/C_1^2 = 0.004$ is deposited on the path taken by the ant. Here Q is initialized to 1000.

		Destination						
	S	**1**	**2**	**3**	**4**	**5**	**6**	**7**
S	0	0.0134	0.005	0.0054	0.005	0.005	0.005	0.005
1	0	0	0.009	0.005	0.005	0.005	0.009	0.0054
2	0	0.005	0	0.005	0.009	0.005	0.0054	0.009
3	0	0.005	0.0054	0	0.005	0.0054	0.005	0.005
4	0	0.005	0.0094	0.005	0	0.0094	0.005	0.005
5	0	0.005	0.005	0.009	0.005	0	0.009	0.005
6	0	0.005	0.005	0.005	0.009	0.0054	0	0.0094
7	0	0.0054	0.005	0.009	0.0054	0.009	0.05	0

Source

It is important to note that periodically as the values in the pheromone matrix get very small, the matrix is normalized. Also, the evaporation and r_o values are adaptively changed to increase intensification as the number of iterations increases.

5. Performance Evaluation

The performance of the five algorithms can be compared against a toy fleet scheduling problem derived solely for the purpose of performance evaluation. Let this problem consist of n = 100 aircrafts (100 routes) of m = 25 model types that need to be scheduled. A corresponding cost matrix can be computed, reflecting the demand for each of the 100 routes, and cost associated to each of the 25 aircraft models (similar to section 4.2).

The following criteria are used to compare the performance of each of the implemented algorithms:

1. Optimality of solution.
2. Iterations required to converge.
3. Total time for execution
4. Time per iteration

5.1. Optimality of Solution

The toy fleet scheduling problem does not have an explicit ground truth available for performance analysis. However, the approach used here is to consider the *cost* or *fitness* of the best performing algorithm in terms of minimization of the cost function as a virtual ground truth. This virtual ground truth will be used throughout as a baseline for comparison against all other metaheuristic algorithms. Table 4 summarizes the data generated by running these algorithms 10 times and computing the mean of the optimality.

It can be observed from Figure 2, TS, SA, GA, and PSO all converge to ~4.3 million cost units. ACO converges to suboptimal value of ~4.5 million cost units.

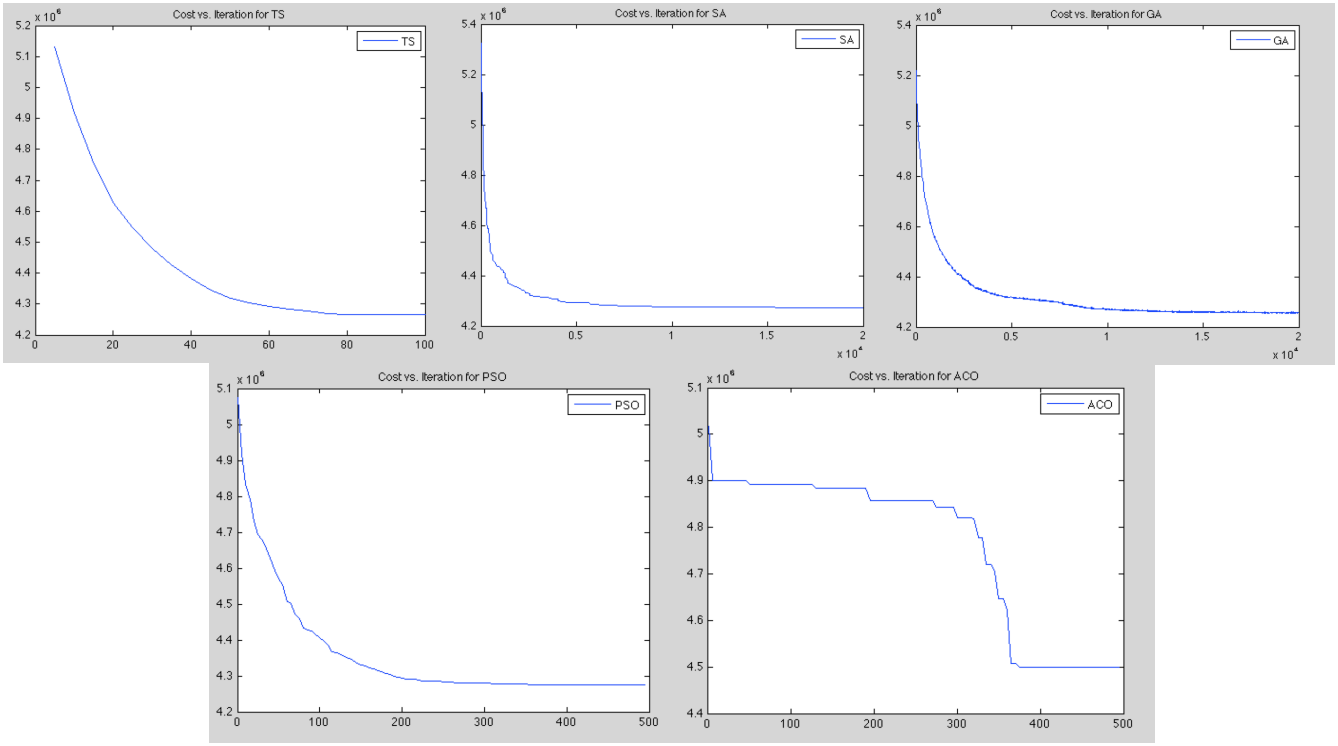

Figure 2: Cost vs. Iterations for (a) TS (top-left) (b) SA (top-center) (c) GA (top-right) (d) PSO (bottom-left) (e) ACO (bottom-right)

The foremost advantage of using 'Optimality' as an evaluation metric is that metaheuristic algorithms that result in statistically significant results consistently may give a strong indication of the most efficient combination of aircraft assignment to routes. This is directly inline with the problem statement stated in Section 3.1. Another advantage of this metric is that it becomes simple to identify whether an algorithm is implemented correctly, parameters are tuned appropriately and whether the chosen design for a particular metaheuristic algorithm scales to handle large inputs for the Fleet Scheduling problem. In other words, it may be the case that certain algorithms are able to obtain optimal solutions for smaller datasets. However as the number of aircraft models and flight routes are increased, the explosion in possible solutions due to

the combinatorial nature of the problem causes certain algorithms to converge to suboptimal solutions or local minima in the multi-modal cost function that is being optimized. The Optimality metric directly gives insight into these situations.

Table 4: Optimal value reported by each of the metaheuristic algorithms for an average of 10 runs

Metaheuristic Algorithm	Optimal Value	Rank
Tabu Search	4207288	4
Simulated Annealing	4200578	3
Genetic Algorithm	**4197373**	1
Particle Swarm Optimization	4197557	2
Ant Colony Optimization	4516824	5

The primary disadvantage of using 'Optimality' as an evaluation metric is that it may be possible for a number of metaheuristic algorithms to give similar solutions with a *assignment cost* value that is within a marginal tolerance level of one another. In such a situation it may be difficult to gauge, rank and hence evaluate the comparative performance of those algorithms. Another drawback of this evaluation metric may be noticed in the fact that different metaheuristic algorithms have their own definition of when optimality is achieved and the criteria used to determine this varies across them. For example, the criteria in swarm based intelligence algorithms may have convergence between particles or ants to be an important signal in optimality or the swarm's best is known to be converging to the optimum solution. On the other hand, population based or memory based search methods employ different rules to converge to the optimal solution. It is important to understand the definition of 'Optimality' each algorithm uses and how well it aligns with that of the general problem statement.

It can be visually analyzed that Tabu Search eventually leads to a near optimal solution in terms of the final cost of assigning *n* routes to *m* aircraft types. Figure 2 (a) illustrates there is in fact over 16.2% increase in the optimality of the solution when juxtaposed against the cost of the initial solution generated by the algorithm. The advantage of using Tabu Search for the fleet assignment problem is that it is able to efficiently generate random permutations of a possible solution, pick the next most optimal swap and then also explore non-improving solutions via the use of a tabu list (short-term memory). All these ideas combined make Tabu Search very suitable for giving near optimal solutions to combinatorially hard problems. One of the shortcomings that can be seen here is the lack of intermediate or long-term memory structures that aid in intensification and diversification of the search space.

The next algorithm that in this evaluation metric is Simulated Annealing which leads to a slightly more sub-optimal solution of 4.4 million units in cost which is close to an 18% improvement from the initial random solutions [Figure 2(b)]. Simulated Annealing introduces the idea of probabilistically accepting

non-improving solutions based on parameters such as the current temperature and the annealing schedule. The introduction of more parameters into an algorithm often make it more likely to obtain a more optimal solution to harder problems. However, these parameters require correct adaptive behaviour and tuning to give the best performing results in this evaluation metric of 'Optimality'.

In the above experiment, summarized in Table 4, Genetic algorithm has the best performance in terms of the achieving the most optimal solution when compared against the other metaheuristic algorithms. Genetic algorithm has the advantage that it allows the population to evolve by using reproduction operators such as mutation and crossover to improve the fitness of the overall population. Over successive generation cycles it also removes the possibility of achieving suboptimal solutions. This is because it is based on the assumption that replacing the least fit individuals of the population with fresh offspring would lead to a more optimal population; realized in our implementation.

It was seen that Particle Swarm Optimization consistently gives us solutions that are second best to the most optimal solution for this toy dataset. The technique of Particle Swarm Optimization maintains a notion of global best which is the best performing solution in the swarm of particles. After every iteration all particles are randomly swapped in an intelligent manner (see Section 4.7.1) by our implementation to mimic the behaviour of the global best solution. The notion of a swarm inherently allows for exploration of different solutions to occur. These concepts make PSO perhaps the one of the most ideal for giving optimal solutions to the Fleet Assignment problem.

The least performing metaheuristic algorithm in terms of optimality of solution was Ant Colony Optimization, whose global minimum ant solution gave an improvement from the initial random solution by a factor of 15 percent. ACO is another swarm based algorithm that requires the tuning and correct adaptive behaviour of a number of parameters such as the evaporation rate, r_0, number of ants and Q etc. A disadvantage of this algorithm is that a simple implementation of ACO may lead to a local optimum solution for multi-modal functions like the cost function of our Fleet Scheduling problem. On the other hand, one of the advantages of ACO is that allows for more exploration of the possible solutions via the use of ants. These ants communicate the fleet assignment combination with the least cost via pheromone deposits. However, as can be seen this does not always give the global minimum solution to the Fleet Scheduling problem.

5.2. Iterations required to converge

The several algorithms developed for the fleet scheduling problem can be compared against different metrics, one of them being the iterations required to converge. If the amount of work done in one iteration is similar for each algorithm, an algorithm with a smaller number of iteration required to reach a suboptimal solution may be preferred over an algorithm that requires a large number of iterations even if it results in a more optimal solution. For this project, the work done per iteration varies significantly for each algorithm. Hence, other metrics such as the time required for convergence should also be taken into consideration when comparing the performance of the algorithms. Each of the algorithms was run 10 times and the

mean of the average number of iterations required to converge was recorded. The results of this test is summarized in Table 5.

Table 5: Average number of iterations required to converge for different Metaheuristic Algorithms over 10 runs

Metaheuristic Algorithm	Number of Iterations required to converge
Tabu Search	79
Simulated Annealing	9019
Genetic Algorithm	12501
Particle Swarm Optimization	309
Ant Colony Optimization	390

The most important advantage of using iterations required to converge as an evaluation metric for metaheuristic algorithm is that it states how fast an algorithm could possibly run given a highly parallelized system. In each of the algorithms, the work done in a single iteration can be parallelized. Hence, theoretically, if there exists a large number of resources to run tasks in parallel, the algorithm can be parallelized. This would end up reducing the time taken per iteration thereby reducing the total execution time. For example, in Tabu Search, the costs of approximately 100^2 candidate solutions are serially looked at in every iteration before the best candidate solution is selected. The work done in one iteration can be parallelized here by passing sets of candidate solutions to systems that can be run in parallel. This would end up reducing the time taken per iteration which would reduce the total execution time.

A clear disadvantage of using iterations required to converge as an evaluation metric is that the work done per iteration may significantly differ per algorithm. Hence, it is impossible to judge the performance of an algorithm solely based on the number of iterations it takes to converge.

It can be noted from Table 5 that Tabu Search requires the fewest number of iterations to converge. This is primarily because each iteration in Tabu Search does significantly more swaps that other algorithms. In one iteration of Tabu Search, all possible swaps are explored and the best swap out of all the possible swaps is selected. For this case, since there are 100 routes, approximately 100^2 swaps are performed per iteration in the worst case. On the other hand, Simulated Annealing and Genetic Algorithm require a large number of iterations before convergence primarily because only one swap is performed per iteration.

PSO and ACO roughly require the same number of iterations to converge. Both these algorithms have a swarm of particles that generate multiple new solutions per iteration. This may be attributed to the cooperative behavior of swarm intelligence, where solutions from good particles are leveraged to generate new solutions.

5.3. Total time for execution

An important factor when comparing the performance of these algorithms is the time it took for them to converge. If an algorithm always converges to an optimal solution, but takes a long time to do so, then an algorithm that quickly converges to a somewhat sub-optimal solution may be preferred. For each of the implemented algorithms, the time it took for them to converge to a solution was recorded. Figure 3 and Table 6 highlights the results over 25 runs of the algorithm.

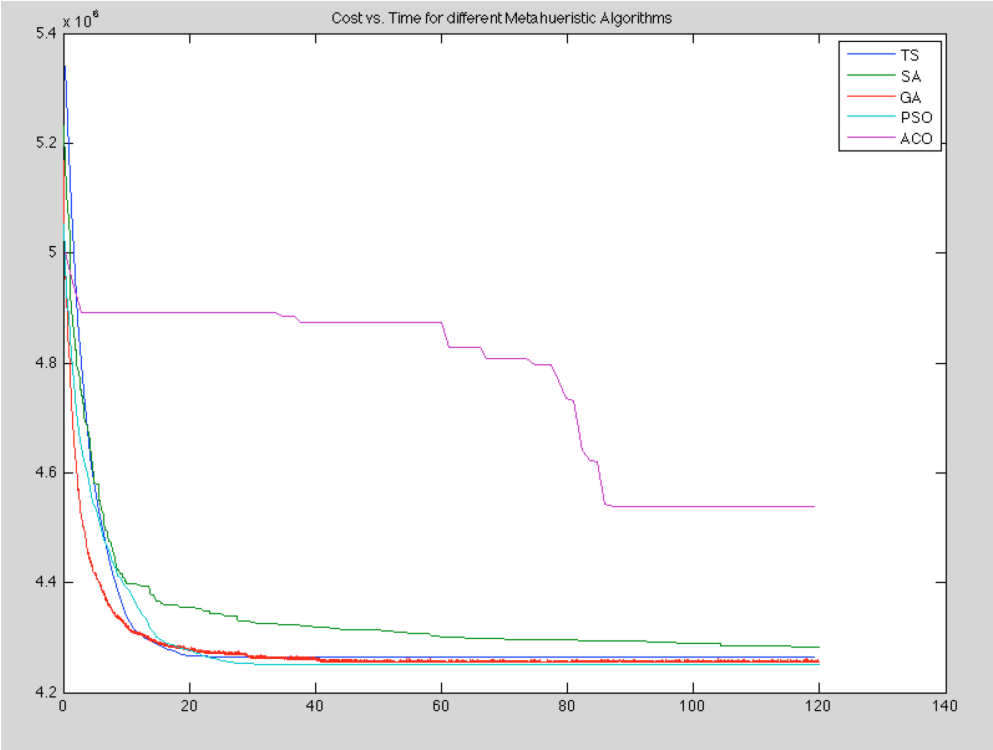

Figure 3: Cost vs. Average time (in seconds) for different metaheuristic algorithms.

Table 6: Average time for convergence of different Metaheuristic Algorithms over 25 runs.

Metaheuristic Algorithm	**Average Time to converge (s)**
Tabu Search	22.2
Simulated Annealing	117.3
Genetic Algorithm	44.2
Particle Swarm Optimization	29.7
Ant Colony Optimization	88.2

The advantage of using total time for execution as a metric for comparison is that it provides a good way of classifying what algorithms can be used based on the requirement of the system. For example, for a real-time system, an algorithm that takes a small amount of time to converge will be prefered even if it yields a suboptimal solution. The disadvantage of using total time for execution as a metric for comparison is that it states nothing about the optimality of the solution. In this case, Ant Colony Optimization takes more time than TS, GA and PSO but yields a suboptimal solution compared to them. Also, Simulated Annealing reaches close to its optimal solution very quickly, but it takes a longer time to converge. Just based on the total time to converge, it would be incorrect to state that Simulated Annealing is a poor algorithm as it does reach close to optimal solution in small amount of time.

From Figure 3, it is noted that Tabu Search converges about 7 seconds before PSO, however it yields a slightly suboptimal solution when compared to PSO. Also, Genetic algorithm takes 22 more seconds when compared to Tabu Search to converge. Simulated Annealing shows improvements for a long duration and hence does not converge for a long time. It should be noted that TS, SA, GA and PSO drop below a cost of 4.4 million units within 5 to 15 seconds, and then diverge in the time they take to converge.

ACO appears to show some improvements from the initial solution, but gets stuck in a local minima. This is mostly like because the ants prematurely take a suboptimal path. This may be improved by tuning the r_o parameter to adaptively improve exploration of the ants if they appear to converge to suboptimal solution.

5.4. Time per iteration of Metaheuristic Algorithms

Another critical dimension of evaluation is the CPU time measured on average per iteration for each of these algorithms. The rationale for choosing this metric is that it may give the reader some insight into the work done per iteration of the algorithm. For example, an algorithm that requires a large number of iterations to converge or reach an optimal solution but performs comparatively minimal work per iteration may be a desirable option for certain applications. On the other hand, a disadvantage here is that this metric does not account on how much improvement is being made on average per iteration towards reaching convergence or optimality.

Table 6: Average Time per iteration of Metaheuristic algorithms

Metaheuristic Algorithm	Average Time per Iteration (s)
Tabu Search	0.28
Simulated Annealing	0.013
Genetic Algorithm	**0.0035**
Particle Swarm Optimization	0.096
Ant Colony Optimization	0.22

It can be seen that Tabu Search requires the highest time per iteration of the algorithm. This is reasonable simply because in our implementation of selecting the next best fleet assignment, all possible swaps are explored and the best one chosen. To explore all possible swaps given *n* possible flight routes, is an $O(n^2)$ algorithm where each element of the vector is compared against every other to determine the next best move.

Simulated Annealing introduces a random change on every iteration of the algorithm which leads to a new solution. The energy of the new solution in relation to the current solution determines whether it is accepted or probabilistically taken given the temperature at that point. This is fairly much simpler work done per iteration than most algorithms. One reason why this number might have a bias towards the higher end is the implementation involved the definition of Matlab User defined classes.

The algorithm that does minimal work per iteration is the Genetic algorithm. Our implementation for the Fleet scheduling problem involved the selection of two individuals from the population and their mutation based on the swap algorithm, followed by replacement of least fit older generation parents by the offspring. At the crux of the algorithm are very simple computational tools such as Roulette wheel selection and exchanging genes within an individual's chromosome. This is reflected in the short duration of time it takes per iteration.

The first swarm based intelligence algorithm does comparatively well on this evaluation metric. There are three key steps per iteration of this algorithm. First is the update of each particle's' position vector based on their current velocity. Next, the cost of each particle is computed and the current best solution for the iteration acquired. Depending on the optimality of the global solution it may or may not be updated. The final step is to update the velocity vector for each particle. Given the extensive cooperative behaviour of this algorithm and the computation that is undertaken to achieve alignment, cohesion and separation between the candidate solutions, it is justified to to take on average slightly more time per iteration.

Finally, Ant Colony optimization takes one of the higher average times per iteration. This may attributed to the behaviour of each ant in selecting a candidate fleet assignment for all **n** routes. Since the model that was adopted is similar to that of Travelling Salesman each ant in the colony deposits pheromone for edge selection based on the cost and certain other heuristics of travelling from the source node to destination. To ensure that pheromone values do not exceed certain thresholds the pheromone matrix is normalized at each iteration. To allow for the adaptive behaviour of the algorithm, parameters such as the evaporation rate and the r_0 values are updated to balance the tradeoff between exploration and exploitation. All these steps constitute a reason as to the high value seen in the CPU time per iteration of ACO.

5.5. Adaptive Implementation of Algorithms

One of the improvements that is made to Tabu Search is to adaptively change the Tabu Length with the number of iterations. The idea is to keep the Tabu Length high initially to allow for more exploration. The Tabu Length is adaptively decreased with the increase in the number of iterations to increase

intensification. This method decreased the time required for the algorithm to converge by about 2.3%. There wasn't any improvement in the final cost reported by the algorithm.

Similarly, an adaptive model is used for Simulated Annealing, where the value of alpha is increased with the number of iterations. This ensures that higher number of iterations are performed at low temperatures, thereby increasing intensification. This approach reduced the convergence time by about 1.22% and optimality of the solution by 0.76% on average.

Genetic algorithm is also modified to include some adaptive features. Particularly, the number of iterations that the algorithm is executed for is adaptively determined. The idea used here is to automatically terminate the algorithm by detecting when convergence is achieved. This is detected by observing when the Inter Quartile Range and standard deviation of the population reaches a particular range. This change decreased the time required to reach the optimal results by 5.4%.

Ant Colony Optimization is also modified to adaptively change the evaporation rate and r_o value. As the number of iterations increases, the evaporation rate is decreased and the r_o value is increased to increase intensification. A very slight improvement of 0.19% in optimal value was observed after this change.

5.6. Summary of findings

Table 7 below summarizes the data presented in the above sections on the toy dataset, with the bolded values denote the best performing algorithm.

Table 7. Summary of performance evaluation

Algorithm	Optimal Cost	Num Iterations required to converge	Avg. Time to converge (s)	Avg. Time per Iteration (s)
TS	4207288	**79**	**22.2**	0.28
SA	4200578	9019	117.3	0.013
GA	**4197373**	12501	44.2	**0.0035**
PSO	4197557	309	29.7	0.096
ACO	4516824	390	88.2	0.22

6. Conclusions and Recommendations

6.1. Deriving Problem Solution

There were a number of steps that had to be followed to come up with a solution to the Fleet assignment problem. First the problem was characterized and numerous factors were identified that would directly impact the cost of assigning aircrafts to flight routes. These factors are listed below:

- Capacity of Aircraft
- Fuel cost per mile of Aircraft
- Demand of passenger per flight route destination
- Distance to flight route destination

The next step was to formulate the problem, define an appropriate cost function and a set of constraints that would aid in optimizing the allocation of aircraft models to flight routes. Closely related to this was the model selection which was determined to be a linear assignment problem such that aircraft model types were assigned to origin-destination pair cities.

Given a strong problem formulation and model in place, the next crucial step was to apply this suitably into the implementation of each metaheuristic algorithm. Our approach was to first standardize the method via which the inputs such as *number of aircraft types, number of flight routes, cost matrix* etc to the problem were generated. Following this a very simple implementation of each algorithm was incorporated to test their basic performance on reduced size problems. As the size of the inputs were scaled to mimic real world fleet assignment problems, each algorithm was tuned by making the parameters more adaptive or modifying certain steps of the algorithm to be tailored more for Permutation based problems. Finally, a set of experiments were designed to have different evaluation metrics that could be used for a comparative study. Evaluation scripts were coded that varied parameters such as number of iterations to run, number of seconds to run each algorithm, determining number of iterations to allow for algorithms to converge to its ‘optimal’ solution.

6.2. Challenges Faced

One of the challenges faced was to ensure that all algorithms converged to similar solutions with fleet assignment costs within some tolerance range of each other. Another major difficulty was to determine the right parameter values that controlled the adaptive behaviour of the metaheuristic algorithms in such a manner so as to give rise to the most optimal solution. In terms of experimentation it was not trivial to come up with benchmarks for the numerous evaluation metrics for the comparative study.

6.3. Results from Experimental Study

The results from the experimental study showed interesting trends for all metaheuristic algorithms across the various evaluation metrics. To begin with, it was seen that on average Genetic algorithm gave slightly more optimal solutions when compared to other metaheuristic algorithms. This is under the condition that all algorithms were allowed to converge to their optimal solution on each run of the experiment. Particle Swarm optimization followed behind GA with a margin of 0.004% (negligible) in its cost associated with the optimal solution.

When the performance of the algorithms were measured with respect to the minimum number of iterations required to converge to an optimal solution the trend was quite different. Tabu Search was able to quickly converge to a near optimal solution with a velocity of up to 4x faster than the next best algorithm in this criterion which was swarm based intelligence optimization, PSO.

An important factor in deciding the performance of algorithms when dealing with large data sets is the CPU time and resources that are required to converge and output their optimal fleet assignment solution. It was seen that Tabu Search also excelled in this metric when compared to its competitors of PSO which was about 1.3 times slower and GA which took up to twice the time to converge to the most optimal solution.

Closely related to the above evaluation metric is the average time per iteration taken by each metaheuristic algorithm. The observation seen from the experiments led to the conclusion that Genetic Algorithms took the least amount of time to complete one iteration, followed by Simulated Annealing and then Particle Swarm Optimization.

6.4. Refinement of Proposed Solution

Several refinements can be made to the proposed solution to increase the optimality of the solutions, minimize CPU time and number of iterations to convergence as well as work done per iteration of the algorithm. The current implementation of our algorithms, all do not exhibit cooperative behaviour. By modifying each algorithm such that input parameters to them adapt automatically based on a tradeoff for exploitation and exploration of the large search space, the proposed solution would definitely be refined to a finer result.

To improve upon the consumption of CPU time and resources, one approach could be to leverage MATLAB's multithreading libraries and implement the cooperative behaviour for the metaheuristic algorithms. This would parallelize the work that can be accomplished in an iteration of the algorithm and also increase the amount of information that can be shared among concurrent solutions to reach more optimal solutions. All in all, it is hoped that these refinements allow each metaheuristic algorithm to identify the most optimal solutions efficiently, given inputs that scale our problem to large real world fleet scheduling challenges.

References

[1] Yaohua Li, Na Tan. (2013, 26 Feb). *Study on Fleet Assignment Problem Model and Algorithm* [Online]. Available http://www.hindawi.com/journals/mpe/2013/581586/

[2] Shangyao Yan & Hwei-Fwa Young. (1994, 27 Sept). *A decision support framework for Multi-Fleet routing and multi-stop flight scheduling*. Available http://cassi.nuaa.edu.cn/download/paper/1147746174.pdf

[3] Internet Archive. (2014, 28 July). *Selecting aircraft routes for long-haul operations* [Online]. Available http://archive.org/stream/selectingaircraf00bala/selectingaircraf00bala_djvu.txt

[4] Xue-Feng Zhang, Bin Tong, Miyuki Koshimura, Hiroshi Fujita and Ryuzo Hasegawa. (2010, 23 July). *A Hybrid Particle Swarm Optimization Algorihtm HTPS for the Flow-Shop Scheduling Problem* [Online]. Availabel http://catalog.lib.kyushu-u.ac.jp/handle/2324/18985/paper1.pdf

[5] Columbia Education. (2010, July 10). *Airline Scheduling: An Overview* [Online]. Avaible http://www.columbia.edu/~cs2035/courses/ieor4405.S14/airline.doc

[6] Wikipedia Foundation. (2014, 26 June). *Assignment Problem* [Online]. Avaiable http://en.wikipedia.org/wiki/Assignment_problem