UNIVERSITY OF WATERLOO
FACULTY OF ENGINEERING
DEPARMENT OF ELECTRICAL & COMPUTER ENGINEERING

Itinerary Optimization

Abbas Ali Haji	ID 20407973	aahaji@uwaterloo.ca
Anish Patel	ID 20421931	a88patel@uwaterloo.ca
Nikhil Thomas Joy	ID 20419188	njoy@uwaterloo.ca
Russell Borja	ID 20419425	rc3borja@uwaterloo.ca
Venkatesh Appea	ID 20427824	vbdappea@uwaterloo.ca

ECE 457A

Instructor: Alaa Khamis

Submitted: August 3, 2015

Table of Contents

List of Tables

List of Figures

1 Summary

Global tourism is a $2 trillion industry, an enormous figure that could be even larger if travellers knew how to plan for their trips more appropriately. The problem at hand is to provide an optimal travel itinerary for tourists when visiting a new city. There are numerous variables at stake when formulating the problem to provide the optimal solution. However, the primary concerns when developing an itinerary are the quality of the places that are visited, the proximity of the attractions to one another, and finally how much of the day can be completely occupied having little idle time.

The goal of this report is to analyze several metaheuristic algorithms in order to tackle this problem, and try to find optimal or near optimal solutions. The optimality of a developed solution is defined as minimizing total commute time, maximizing the average ratings of attractions contained in a solution, and maximizing the duration spent at each of these attractions, effectively minimizing idle time. The chosen algorithms all have the same objective, and are finally compared amongst each other by considering CPU time and mean fitness of the obtained solutions.

The metaheuristic algorithms all operate from the same data source, having the same pool of attractions with the same commute time matrix determining the travel time from any one attraction to another. For the algorithms that require an initial solution to determine optimality, an initial solution is generated at random. This report analyzes the following metaheuristic algorithms for problem solution:

- Tabu Search (TS)
- Simulated Annealing (SA)
- Genetic Algorithm (GA)
- Particle Swarm Optimization (PSO)
- Ant Colony Optimization (ACO)

The major conclusion made in this report are that Particle Swarm Optimization holds the best performance with respect to the global fitness value in relation to the other algorithms. Genetic algorithm was the worst algorithm whereas the Ant Colony Optimization provided near optimal solutions.

2 Introduction

Planning an itinerary for the day is perhaps the most challenging and time-consuming task when it comes to travelling. It is always ideal to maximize the user's time by visiting the best attractions available; however, choosing from a large pool of highly-rated sites significantly makes the decision-making process more difficult. The motivation behind this study is to determine the most efficient way to find the best attractions to visit in a new city. Travelling is meant to be enjoyable, so having an application eliminate the decision-making process allows people to spend more time sightseeing rather than looking for things to do.

The problem at hand is to create the most optimum itinerary given a list of available attractions, their respective ratings, suggested duration of visit, and the commute time between each destination. An optimal itinerary minimizes the commute time between each attraction whilst maximizing the quality and time spent at all sites. A solution to this problem is not limited to generating itineraries and planning trips; any multi-objective problem that involves resource allocation would have very similar elements. Therefore, doing an investigation on itinerary optimization can be beneficial to other areas of study as well.

The purpose of the report is to compare five different metaheuristic algorithms and their ability to generate an optimal itinerary solution from the same data set. The algorithms used for this study are Tabu Search (TS), Simulated Annealing (SA), Genetic Algorithm (GA), Particle Swarm Optimization (PSO) and Ant Colony Optimization (ACO). This itinerary optimization problem is a combinatorial, multi-variable optimization problem.

To solve the problem, sample data must first be obtained of the attractions. For the scope of this study, 75 attractions are gathered along with their Yelp ratings from 0 to 5 stars. The commute time is then calculated using the distances between attractions and the time spent at each attraction is a rough estimate based on category. An objective function is then formulated incorporating the three variables and implemented in MATLAB through the five different algorithms. Once the results are gathered, they are compared and analyzed based on performance and optimality. The best metaheuristic approach to this problem is revealed in the conclusion, in addition to outlining other applications of this study's findings.

3 Literature Review

The itinerary optimization problem has very similar elements to the popular knapsack problem, which tries to determine the best combination of items that satisfy a specified hard constraint. For this reason, there have been a number of published papers that describe solutions to this problem using a variety of known algorithms. The methodologies used in these articles are compared with the ones implemented in this report, highlighting key similarities and differences that arise.

The standard Tabu Search (TS) implementation [1] of the well-known knapsack problem is the inspiration behind the modified TS implementation used to solve the itinerary problem. An adaptive approach is implemented, such that the tenure that a move is put in the tabu list changes dynamically depending on the state of the solution. If the solution is in an improving state, the tenure is decremented, and in case of the converse, the tenure is incremented. This approach was proven to be effective, when Dell'Amico and Trubian first introduced it in [2] to solve the Job Shop Scheduling problem. Penalizing frequent moves as discussed by Hooshmand, *et al.* [3] effectively allows diversification and exploration of other solutions rather than settling at local maxima. C. H. Aladag and G. Hocaoglu [4] describe the aspiration criterion as a global aspiration by objective and this is employed in the TS implementation so as to converge to the optimal move. Using methodologies from these different strategies, and applying it to the traditional knapsack problem, ideal solutions are produced.

There have been several papers written about Simulated Annealing (SA) and the knapsack problem. Martinjak *et al.* demonstrated that an increase in the total number of iterations leads to better results only if cooling is slower [6]. Simply increasing the number of iterations or changing the cooling schedule does not necessarily lead to improvements. This is similar to what we see in the implementation of SA for this problem since results are more likely to get rejected once the number of iterations is high. When the cooling schedule is more gradual, it allows the algorithm to be more forgiving of worse solutions, which leads to greater exploration of the sample space in search for the global optimum. Furthermore, F.T. Lin's study incorporates the genetic algorithm to his simulated annealing approach to obtain even better results [5]. His version of the algorithm achieves a 3% difference between final and optimum solution, which is significantly better than using plain SA. Genetic algorithm tends to generate better solutions, as

the variation for each iteration is less random; combining this with the efficiency of simulated annealing results in a very robust algorithm.

With regards to a Genetic Algorithm (GA) implementation, R. Singh [7] concluded that GA provides a method to solve the knapsack problem in linear time. I. Martinjak, *et al.* [8] states that genetic algorithms perform better using a uniform crossover operator as opposed to a single point crossover. Larger population sizes directly affect the performance of the genetic algorithm. S. Khuri, *et al.* [9] states that genetic algorithm is not a suitable algorithm for large knapsack problems because it is in essence a blind genetic search. This is relevant to our problem because the population size for our problem is relatively large, therefore by [9], we should expect similar results. Moreover, the GA algorithm is not social in our problem, hence it cannot overcome this problem, and hence by [9] it should result in suboptimal results.

In regards to Particle Swarm Optimization (PSO), there were many papers relevant to solving the knapsack problem using the binary particle swarm optimization algorithm. Kong and Tian [14] conclude that PSO is capable of producing high quality solutions for problems of various characteristics. In [13], Liang, *et al*. proposes an adaptive version of the PSO algorithm in order to avoid the solutions from converging too early and to be able to solve problems with large data sets. They suggest killing particles that have reached the swarms best and initialize new particles with high fitness values and kill weak particles in the swarm. Their results prove that the algorithm outperformed traditional PSO when solving the knapsack problem. Another paper by Hembecker, *et al*. [12] suggests how powerful and simple PSO can be to solve combinatorial problems, specifically the knapsack problem. They concluded that PSO on average converged to 2.269% of the known optimum and that PSO seems to be very efficient in navigating the hypersurface of the search space as well as is very robust. An interesting observation that they made is that PSO is able to find good solutions (and, sometimes, the best solution) independently of the initialization and the trajectory of the particles. Hembecker, *et al*. discovered after running many iterations that the PSO algorithm performance decreases linearly as more complexity is added. However this decrease in performance occurs when the problem is highly complex and requires a large number of iterations to converge. Although the papers mentioned above discuss the efficiency and simplicity of the PSO algorithm, all fail to compare the performance and accuracy of the PSO algorithm to other metaheuristic algorithms especially in the class of

combinatorial problems. F. He [11] is the only one who experimented with binary particle swarm optimization (BPSO) and compared its performance to another metaheuristic algorithm. F. He concludes that BPSO has good performance and the convergence rate is much higher when compared to GA.

Lastly, in regards to the Ant Colony Optimization (ACO), the implementation using an adaptive global updating strategy has been proven to outperform the standard ACO algorithm [15]. The use of a global pheromone update after the ants have finished the iteration helps the algorithm converge much quicker than the standard ACO, as concluded by Sun, *et al.* [15]. R.G. Tiwari, *et al.* [16] discuss how the ACO algorithm was implemented using multiple ant colonies that communicated with each other using a global pheromone and local colony pheromones. The study found that the method of using the iteration-best solution to update the global pheromone outperforms the original ACS one colony algorithm with the same number of ants. With a hybrid version of these two implementations as done in this report, we can expect that the convergence of this algorithm will be relatively quick as ants will keep track of the best solutions while each colony attempts to try finding new routes via random roulette wheel selection. Although, as mentioned in G. Liu, J. Xiong [17], the strategy of updating the global pheromone does not reflect all the differences between good and bad paths, as only the best solution path is used in the global update. This update method may prevent ants from further searching for better solution and get stuck in a local optimal path [17], but the initialization of the pheromone matrix after each run to give adaptive feedback will reset the pheromone concentrations on all paths, and give the global best solution a minor increase in pheromone. This along with the roulette wheel selection should help minimize the issue of getting stuck in a local optimum.

4 Problem Formulation and Modeling

4.1 Problem formulation:

The goal is to compare five different meta-heuristic algorithms for a constraint satisfaction problem. The objective is to create an optimal travel itinerary schedule. A travel itinerary solution is evaluated based on its ability to meet both hard and soft constraints. These constraints rely on maximizing the total duration of the itinerary within a time period, minimizing total commute time in the itinerary and maximizing overall rating of the venues visited.

The input of the objective function is Z_D, Z_c and Z_r and the output is X where X is a maximization function.

$$X = \begin{pmatrix} Z_D \\ Z_c \\ Z_r \end{pmatrix}$$

Z_D represents the duration of the itinerary (minutes). Z_c represents the total commute time in the itinerary. Z_r represents the average rating of all the venues in an itinerary. The ratings are provided from Yelp.

Each solution must abide a set of hard and soft constraints. The hard constraints are:

- Duration of itinerary must not exceed total duration in a given day. This is set to 480 minutes in a day. This is expressed as the summation of the duration of each venue and total commute time to be visited in an itinerary.
- There should never be repetitions of the same venue in a solution.

$$Z_D + Z_c \leq 480\ minutes$$

$$S(v) = \{v\}_{i \in \{1,..,n\}}\ where\ v = [v_1, \dots, v_n]$$

The soft constraints are:

- To maximize the number of venues visited in a day. This is represented by the total number of venues.
- To minimize the total commute time.
- To create an itinerary that holds venues that have high ratings. This is expressed as the average rating of all the venues.

$$Min(Z_c)$$

$$Max(Z_r)$$

$$Max\left(\sum_{i=0}^{n} v_i\right) \; where \; v \in \{S\} \; and \; \{S\} \; represents \; a \; solution$$

These constraints are then measured using the global fitness function:

$$f = \frac{1}{1 + \frac{Z_c}{480}} * Z_r * \frac{Z_D}{480}$$

The fitness function is used to evaluate each solution. Note that every implemented algorithm uses this fitness function to evaluate the quality of a solution.

4.2 Problem Modeling

Our problem is a single objective optimization function that is represented in a three-dimensional search space. It optimizes the three parameters: Z_D, Z_c and Z_r where all parameters exist in the non-negative domain. Z_D and Z_c are both represented using discrete positive integer values because they represent minutes of the time domain. Z_r holds a range of 0-5 which represents the ratings of venues. Although our search space initially appears to be large, it is restricted with our hard constraints. This includes the constraint that our solutions must be below 480 minutes in total commute and duration time. Therefore, search space is further reduced.

4.3 Reduced Size Problem

Some algorithms reduced the number of venues that are used 75 to 9 venues. This is done in order reduce the complexity and the number of resources being used for hand iterations. Also,

algorithms such as ACO, reduced the total duration from 480 minutes to 240 minutes in order to reduce the total number of required computations. In this scenario, the fitness function is changed due to the change in duration, but it is not changed when the number of venues being used is reduced. Lastly, the initial population is reduced for population based meta-heuristic, algorithms such as PSO and genetic algorithms in order to reduce the complexity and computations required for hand iterations.

4.4 Real Sized Problem

The real sized problem accounts for all 75 venues being used with the total maximum duration of 480 minutes. Moreover, the real sized problem will run many more iterations than displayed in the hand iterations. The overall complexity of the global fitness function remains the same regardless of the problem size. The initial population is increased to a larger size for population based meta-heuristic algorithms such as PSO and genetic algorithms.

5 Proposed Solution

5.1 Tabu Search

5.1.1 Design Decisions

5.1.1.1 Initial Conditions

From a pool of 100 feasible solutions, a solution is selected to determine the ideal itinerary for a particular day. A random solution is selected as the starting point, as the initial solution can be any solution, and Tabu search aims to determine the best possible solution from any start point with the use of the different operators.

5.1.1.2 Objective Function *f(x)*

The objective function selected for this problem to determine the fitness of a selected neighbour from the neighbourhood should include effects from the overall commute time, total duration spent on sightseeing, and the average rating of the attractions that are visited. For each call that is made to calculate the fitness of the current solution, a travelling salesman optimization is conducted so that the optimal ordering of attractions is maintained so that commute time is kept at a minimum.

$$f(solution) = \begin{cases} \dfrac{avgRating}{1 + \dfrac{totalCommuteTime}{maxDuration}} * \dfrac{\sum attractionDurations}{maxDuration}, & \sum attractionDurations \leq 480 \\ 0, & \sum attractionDurations > 480 \end{cases}$$

5.1.1.3 Neighbour Selection *N(x)*

Different neighbours can be generated for each solution by conducting two different operations, an addition of an attraction to the solution from the attraction pool and a removal of an attraction from the existing solution. The number of neighbours for the addition operator for an attraction pool of size m is m. The number of neighbours for the removal operator for a solution of size n is n.

An addition or removal operator is conducted depending on how empty the current solution is. A random number r_0 is generated such that $r_0 \in (0,1)$ and an addition operator is selected if:

$$r_0 < \frac{empty\ time\ in\ minutes}{total\ time\ in\ minutes}$$

In the case of the converse, that is the addition operator is not selected, the removal operator is selected.

5.1.1.4 Operators

5.1.1.4.1 Addition Operator

The addition move, or the addition of an attraction contained in a particular solution, in which a solution $x' \in X$ is a neighbour of $x \in X$ if it can be obtained by the addition of an attraction s from the attraction pool S into an empty time slot t.

5.1.1.4.2 Removal Operator

The removal move, or the removal of an attraction to a particular solution, in which a solution $x' \in X$ is a neighbour of $x \in X$ if it can be obtained by the removal of an attraction s from the solution set of size n which results in the creation of an empty time slot t.

5.1.1.5 Recency Tabu List

A recency tabu list is implemented in this Tabu Search algorithm so that the algorithm is prevented from doing operations on attractions that have previously been operated on. When an operation occurs, the attractions to which the operation was conducted are then added to the tabu list. The size of the list is limited to 76, as there are 76 attractions in the overall Attraction pool.

5.1.1.6 Aspiration Criterion

As an aspiration criterion, global aspiration by objective is used. If an explored move results in a solution x' with a fitness value much less than the fitness of candidate moves populated in the neighbourhood at random, the move is made regardless of the tabu status.

5.1.1.7 Adaptivity

The algorithm is made adaptive by changing the length of the tabu tenure. The tabu tenure $t \in [1,5]$, and as the fitness of the solution improves t is allowed to decrement by 1 and as the fitness of the solution degrades t is made to increment by 1. If the algorithm comes across the best solution in a region, t is set to 1, so that the search can be focussed in a region of potential improvement.

5.1.1.8 Diversification

When each neighbour is calculated for the current solution, a diversification cost is applied to the fitness such that the fitness is penalized for having used a neighbour that exists in the tabu list.

$$diversificatonCost = neighbourFitness - 0.5 * \frac{tabuList(i)}{tabuTenure}$$

5.1.1.9 Termination Criteria

The TS is conducted for as many iterations as the user requests. As the algorithm proceeds its course, it keeps track of the fitness of the existing solution at any given point. The for loop is terminated when the fitness values begins to converge, and there are no changes to the current solution over the past 50 iterations.

5.1.2 Hand Iterations

5.1.2.1 Iteration 1

Initially, we start with a randomly generated solution that meets the constraint of the maximum duration of the day inclusive of commute time of 480 minutes.

$$solution_{initial} = [0,32,16,62,47]$$

$$totalDuration_{initial} = 390$$

$$fitness_{initial} = 1.6835$$

Now, we generate a random number $\in$ *[1,100]* and we also calculate the percentage of emptiness of a particular solution.

$$r_{num} = 69$$

$$emptyTime = round(\left(1 - \frac{duration_{solution}}{480}\right) * 100) = 3$$

Since:

$$r_{num} > emptyTime$$

A removal operation is undertaken. The neighbours for the current solution given the removal operator involve removing any one of the attractions contained in the solution, and thus the neighbourhood is built.

Table 1: Table of Neighbourhood list sorted by decreasing Diversification Cost of all Neighbours of an Remove Operator

Attraction	Diversification Cost	Fitness
62	1.472727273	1.472727273
32	1.286549708	1.286549708
47	1.218411552	1.218411552
16	0.888157895	0.888157895

Since this is the first iteration, the tabu list is empty. Thus, there will be no difference between the diversification cost and the fitness of the particular move. Since the highest fitness is the removal of attraction 62, it is undertaken and attraction 62 is now in the tabu list.

$$solution_{current} = [0,32,16,47]$$

$$fitness_{current} = 1.4727$$

5.1.2.2 Iteration 2

Another random number is generated and this is compared to emptyTime.

$$r_{num} = 2$$

$$emptyTime = round(\left(1 - \frac{duration_{solution}}{480}\right) * 100) = 10$$

Since:

$$r_{num} < emptyTime$$

An addition operation is undertaken. The neighbours for the current solution given the addition operator include all the attractions in the attraction pool excluding the ones already included in the solution. The fitness of neighbours are assigned a value of 0 when the total duration of the solution exceeds 480 minutes.

Table 2: Table of Neighbourhood list sorted by decreasing Diversification Cost of all neighbours of an Addition Operator

Attraction	Diversification Cost	Fitness
22	2.264972777	2.264972777
21	2.23255814	2.23255814
50	2.220640569	2.220640569
74	2.220640569	2.220640569
…	…	…
3	0	0
4	0	0

Out of the 72 neighbouring solutions, the top one with the greatest fitness is chosen. Attraction 22 is thus added to the current solution. As attraction 22 is added to the solution, it is also added to the tabu list.

$$solution_{current} = [0,32,16,47,22]$$

$$fitness_{current} = 2.2650$$

5.1.2.3 Termination

After exactly two iterations, the algorithm terminates with a solution [0,32,16,47,22] with fitness 2.2650.

5.2 Simulated Annealing

5.2.1 Implementation Description

Simulated Annealing mimics the characteristics of annealing in metallurgy to find an ideal solution to the global optimum of a discrete sample set. It does this by always accepting solutions that are better and probabilistically accepting solutions that are worse. The probability that a worse solution is chosen is determined by a constant (initial temperature), which gradually decreases over time until the global optimum is obtained.

5.2.1.1 Selecting Initial Values

In SA, it is imperative to choose an initial temperature that is neither too high nor too low. Temperature that is too high results in the acceptance of all worse solutions, which means that it takes longer for the algorithm to converge to an optimal solution. Likewise, an initial temperature that is too low also does not lead to the global optimum, as it will not accept any worse solutions. In this case, it may reach a local minimum or maximum and not proceed to explore further.

To find a suitable initial temperature, a heuristic calibration can be used. Since the maximum change in the objective function is not known, it is best to first gauge the average difference between a few evaluated random solutions. To achieve this, 10 random solution sets are generated and the average absolute difference between their objective functions are obtained. The probabilistic function for accepting worse solutions is normally as follows for finding the global maximum:

$$P(worse\ solution\ accepted) = \exp\left(\frac{f(X_2) - f(X_1)}{T}\right)$$

The standard initial acceptance rate for the algorithm to be immediately effective is approximately 60%. Rearranging the above equation, the initial temperature can be determined from the average deltas of the objective function.

$$T_0 = \frac{\frac{\sum_{i=2}^{10} | f(X_i) - f(X_{i-1}) |}{10}}{0.60}$$

The initial temperature changes for each run of the algorithm, but will be within 0.85 to 2.25. For

the purposes of this report, the initial temperature of $T_0 = 2$ will be used.

5.2.1.2 Generating Neighbouring Solutions

The first solution set is random and will be selected from a pool of available attractions, with the condition that the sum of all durations and the commute times of the venues is less than the allotted time. To generate a new solution, one of two possible operations will be applied to the existing solution set. The first is the add operation which concatenates a new attraction by random to the existing set. The second operation is the remove function, which randomly deletes an existing attraction in the list. The operation that is to be chosen will depend on the ratio of current total duration of attractions and the allotted time for all attractions. The add operation will be favoured more if there is plenty of time to be filled within the allotted time, otherwise the remove operation would be preferred. A random variable r would generate a number from [0,1]. If this number is smaller than the duration ratio the remove operator is performed to create the neighbour solution, otherwise an attraction is added to the solution set.

$$P(remove\ attraction) = \frac{(total\ duration + total\ commute\ time)}{(alloted\ time\ for\ attractions)}$$

$$P(add\ attraction) = 1 - P(remove\ attraction)$$

5.2.1.3 Cooling Schedule

The temperature is used to determine the probability of accepting a worse neighbouring solution. It is adaptively lowered once the number of accepted solutions reaches a specified limit and depending on how far the algorithm has come. In the beginning, the initial temperature is high enough to accept the majority of worse solutions, which is essential for exploring the sample space and possibly moving away from local optima. The cooling rate for when the most exploration happens is set to 0.75 and the maximum number of accepts is capped at 3. The reason for this limit is to ensure that that the solutions stabilize for each given temperature. Once a predefined colder temperature is achieved, in this case T=0.75, the algorithm intensifies its search by raising the cooling ratio to 0.9. This allows it to search for better solutions around the near optimal solutions it has found before. A geometric cooling schedule is selected in order for the temperature to reach its final value faster.

$$T_{new} = \alpha * T_{old}$$

where α is the dynamic cooling constant.

5.2.1.4 Accepting a Solution

The algorithm accepts the neighbouring solution under two conditions. If the neighbouring solution has a better objective function than the current solution set, then it is automatically accepted. If it is worse, then it is evaluated by a probabilistic function, which in turn decides whether or not to accept the solution. The lower the temperature of the probability function, the less likely it is for it to accept a worse solution. The following formula is used to determine the acceptance of a worse solution set:

$$\Delta F = f(X_2) - f(X_1)$$

$$P(worse\ solution\ accepted) = \exp\left(\frac{\Delta F}{T}\right)$$

Once the solution is accepted, the travelling salesman algorithm is implemented to obtain the most optimal ordering of the attractions in such a way that commute time is minimized. This allows the algorithm to not only search for the global optimum in the global sample space but within the local sample space as well. The best solution and maximum fitness so far are both updated if the objective function of the accepted solution yields a better result than the current best.

5.2.1.5 Termination Criteria

There are three possible conditions that can terminate the algorithm. The first stop condition is if the total number of iterations exceeds the maximum number of iterations. The second stop condition occurs when the temperature goes below the minimum temperature. The last stop condition occurs when the number of consecutive rejected solutions exceeds the maximum permissible number.

5.2.2 Hand Iterations

5.2.2.1 Initialization

Before generating the first solution set, some initial values must be set. These include the initial temperature, threshold temperature, cooling coefficient alpha, and max number of accepts before temperature is cooled down. In the MATLAB implementation, the alpha coefficient is dynamic

to reflect the improvements made on the best solution set and the initial temperature is first calibrated. However, for the purposes of the hand iterations, these values will remain static.

Table 3: Initial Values for Algorithm

Attribute	Value
Initial Temperature	2
Cooling Coefficient (α)	0.75
Minimum Threshold Temperature	0.05
Maximum Number of Accepts	1

Once these values have been set, an initial solution is generated from the reduced sample set. For the sake of simplicity, the allotted time will be 240 minutes for the total duration and commute time. The total commute time and duration will be normalized by this value (divided by 240 to obtain a value from 0 to 1). Starting from the hotel (attraction 0), attractions are randomly concatenated to the solution set until no more can be added to generate an initial solution.

Table 4: Initial Solution Set X_0 Properties

Property	Value
X_0	[0 2 6 8]
Duration(2,6,8) : Normalized Duration	180 : 0.75
Commute Time(0,2,6,8) : Normalized Commute Time	33 : 0.1375
Average Rating	2.33

$$f(X_0) = \frac{Average\ Rating * Normalized\ Duration}{(1 + Normalized\ Commute)}$$

$$f(X_0) = 1.536$$

5.2.2.2 Iteration 1

A neighbouring solution is obtained by either adding or removing an attraction to the initial solution. The probability of removing an attraction is equal to NormalizedD + NormalizedC. If a

random variable from [0,1] is less than NormalizedD + NormalizedC, then the remove operator is used. If the random variable is greater, then the add operator is performed. For this iteration, the random variable is 0.33.

$$r = rand[0,1] = 0.33$$

$$Normalized\ Duration + Normalized\ Commute = 0.888$$

Since r < (NormalizedD+NormalizedC), remove attraction from X0 by random

Table 5: Solution Set X_1 Properties

Property	Value
X_1	[0 6 8]
Duration(6,8) : Normalized Duration	120 : 0.5
Commute Time(0,6,8) : Normalized Commute Time	31 : 0.129
Average Rating	1

The objective function of X1 is evaluated and compared to f(X0). If it is greater than the previous solution, accept X1 as the current solution. If it is lower, then it must be evaluated with the probabilistic function to determine whether or not to accept it as a viable solution.

$$f(X_1) = \frac{Average\ Rating * Normalized\ Duration}{(1 + Normalized\ Commute)}$$

$$f(X_1) = 0.443$$

Since f(X1) < f(X0), the probabilistic function must be used.

$$\Delta F = f(X_1) - f(X_0)$$

$$P(worse\ solution\ accepted) = \exp\left(\frac{\Delta F}{T}\right) = \exp\left(-\frac{1.093}{2}\right) = 0.579$$

Generating a random variable from [0,1] returns 0.282 which is less than P(X1 accepted) which means that even though X1 is a worse solution than X0, it is accepted. The number of accepts is

incremented and the current solution is updated to X1. Since numAccepts = maxAccepts, numAccepts is reset to 0.

5.2.2.3 Iteration 2

The current solution is now X1 = [0 6 8] with f(X1) = 0.445 and (NormalizeD + NormalizedC) = 0.629. A neighbouring solution must be created from X1 and the temperature must be updated since the number of accepts has reached the max number. The objective function of X2 is then evaluated and compared with f(X1).

Table 6: Cooling Step

Property	Value
T_{old}	2
Cooling Coefficient (α)	0.75
T_{new}	1.5

$$r = rand[0,1] = 0.656$$

$$Normalized\ Duration + Normalized\ Commute = 0.629$$

Since r = > (NormalizeD + NormalizedC), therefore random attraction is added to X_1 to generate neighbour X_2

Table 7: Solution Set X2 Properties

Property	Value
X_2	[0 6 8 7]
Duration(6,8,7) : Normalized Duration	180 : 0.75
Commute Time(0,6,8,7) : Normalized Commute Time	34 : 0.142
Average Rating	1.667

$$f(X_2) = \frac{Average\ Rating * Normalized\ Duration}{(1 + Normalized\ Commute)}$$

$$f(X_2) = 1.095$$

Since $f(X_2) > f(X_1)$, the solution is accepted and the number of accepts is incremented by 1. After two iterations, the best solution so far is [0 2 6 8] with an objective function value of 1.536.

5.3 Genetic Algorithm

5.3.1 Design Decisions

5.3.1.1 Data structure:

Each individual is represented in a permutation encoding where each gene is a venue, expressed as an integer. Each gene holds properties: duration, commute time and venue rating in which the algorithm is trying to optimize. The commute time is the time from one gene to the next gene. Therefore, the last gene in an individual holds a commute time of 0 because there is no object after it. The first gene, x_1, is set a constant in all individuals and thus, represents the starting point. Since user is not expected to stay at the starting point, the duration and venue rating are not included in its properties. Note that the length of each individual is subjected to a variable length.

$$Individual = [x_1 x_2 x_3 \ldots]$$

where x_n is an integer ID of a venue representing a gene in an individual.

$$x_n = \begin{matrix} duration \\ commute\ time \\ venue\ rating \end{matrix}$$

5.3.1.2 Initial Population

The initial population will be a set of 5 different individuals. Placing an arbitrary venue in a random position in the solution generates a new solution. Therefore, this will generate a variety of individuals of different permutations and lengths. This process is used to develop the initial 5-hardcoded solutions. All the individuals in the initial population all follow the hard constraints; therefore no individual will have a fitness of 0. Following the first iteration, the population pool is filtered to only have individuals that do not have a fitness of 0. This is performed in order to help the algorithm converge to an optimal solution. Note that there will exist a set of 75 objects in the population pool in which the algorithm is allowed to use to construct solutions.

5.3.1.3 Parents Selection

The four most fit individuals are selected using the elitism methodology. For this hand iteration, only the 2 best individuals are selected because of the reduced population size.

5.3.1.4 Crossover

A single-point crossover operator is used as the reproduction operator. The probability of crossover is set at 0.9. This is an appropriate operator for the encoding schema that is being used to represent each individual. Moreover, it is able to operate on variable individual lengths. The single-point crossover operator chooses an arbitrary point in both parents. Data beyond the selected points are then swapped between the parents, thus generating the children. The example below illustrates the single-point crossover.

"yy.xx" where the . represents a randomly chosen point for a crossover

Parent 1: 123.45

Parent 2: 1.979

Child 1: 123979

Child 2: 145

Note that the first gene in all individuals cannot be crossed because it is the starting point that is common among all individuals. Also, the parents selected to crossover with each other are arbitrarily selected among the parents.

5.3.1.5 Mutation

The mutation operator is an adaptive operator and is analogous to bit-flipping mutation. At each gene, a mutation will occur if a randomly generated number is less than the mutation rate. In the case of a mutation, an arbitrary venue is selected from the population and it then replaces the existing gene. The arbitrary venue is chosen by randomly selecting a venue in the population based on its venue ID.

Since the mutator operator follows Rechenberg's 1/5 success rule, it is initially set to 0.2. This operator is also adaptive. It evaluates the success rate of the mutator operator using the formula:

$$Mutator\ success\ rate = \frac{\#\ of\ success}{Total\ number\ of\ mutations}$$

A successful mutation is when the mutation results in an increase in fitness function from the former child. Through experimentation of the genetic algorithm, the success rate is evaluated every 100 generations. If the mutator success rate is greater than 0.2 then we increase the mutation rate else we decrease the mutation rate by 0.03.

5.3.1.6 Survival Selection

An elitism selection algorithm is used as a survival selection method. This method selects the 2 highest fit individuals for survival based on their overall fitness value. The fitness is evaluated using the global fitness function discussed in the problem statement. It selects the 2 highest fit individuals among the parents and children for the next iteration.

$$Global\ fitness = \frac{1}{1 + \left(TotalCommuteTime\frac{1}{480}\right)}(AverageRating) * (\frac{Duration}{480})$$

Note that each individual can hold up to 480 minutes in its itinerary. Note that a violation in any hard constraints results in a fitness of 0. This includes repetition in venues in an individual and exceeding the maximum time duration in a solution.

$$Hard\ constraint: TotalCommuteTime\ + TotalDuration > 480\ then\ fitness = 0$$

$$Hard\ constraint: repeated\ venues\ in\ individuals\ then\ fitness = 0$$

5.3.1.7 Termination

The genetic algorithm is terminated under certain conditions. The first condition is if the algorithm does not improve the solution within 2000 iterations. This number is found experimentally. The other termination condition is whether it reaches its maximum of 10000 iterations. This number is also found through experimentation.

5.3.2 Hand Iterations

In both hand iterations, we assume that a random number generated is below the crossover probability, therefore crossover occurs in both iterations.

5.3.2.1 Population initialization

The population is initialized by arbitrarily selecting two random individuals. Each individual generated abides the hard constraints.

Population:

$$Individual_1 = [0\ 2\ 4\ 7\ 6]$$

$$Individual_2 = [0\ 1\ 5\ 3]$$

5.3.2.2 Iteration 1

A pair of parents are selected from the set population. Note that the hand iterations will only consist a subset of the total points.

Parents:

$$Parent_1 = [0\ 2\ 4\ 7\ 6]$$

$$Parent_2 = [0\ 1\ 5\ 3]$$

The crossover is now applied on the parents. The crossover points in each individual are arbitrarily chosen and a single-point crossover is conducted.

Note: The '.' represents the crossover point in each individual.

$$Parent_1 = [0\ 2\ \ 4\ .\ 7\ 6]$$

$$Parent_2 = [0\ 1.5\ \ 3]$$

The resulting crossover generates the following children:

$$Child_1 = [0\ 2\ \ 4\ 5\ 3]$$

$$Child_2 = [0\ 1\ 5\ 7\ 6]$$

Following the crossover, the mutation algorithm is applied to each child. The mutation occurs on each gene with the probability based on the mutation rate. The mutation rate is dependent on the population size and abides the formula:

$$Mutation\ rate = \frac{1}{Population\ size}$$

Since the population size is 2, the mutation rate is 0.5. Mutation occurs if a randomly generated number from 0-1 is less than the mutation rate. A venue is arbitrary chosen

Note: The '*' represents that a mutation occurs in the gene.

$$Child_1 = [0\ 2^*\ 4\ 5\ 3] = [0\ 9\ 4\ 5\ 3]$$

$$Child_2 = [0\ 1\ 5^*\ 7\ 6] = [0\ 1\ 3\ 7\ 6]$$

The results from the mutation are as follows:

Table 8: Fitness values before and after mutation at iteration 1

Individual	Fitness
$Child_1 = [0\ 2\ \ 4\ 5\ 3]$	0
$Child_2 = [0\ 1\ 5\ 7\ 6]$	1.05
$Child_{1\ Mutation} = [0\ 9\ 4\ 5\ 3]$	0
$Child_{2\ Mutation} = [0\ 1\ 3\ 7\ 6]$	1.58

The mutation results in a more fit child, therefore, mutation success rate is 1/2 =0.5. This rate will be constantly updated in every mutation and will be recorded for 10 generations. Following the mutation, the total population and their total fitness is calculated below. The survivors are selected using an elitism algorithm. The 2 most fit individuals are selected for the next iteration. The first child and second parent hold a fitness of 0 because its total duration exceeds the hard constraint of maximum 8 hours. Parent 2 individual is removed from the population pool because it holds a 0 fitness.

Table 9: A comparison of all the fitness values after crossover and mutation at iteration 1. These values are used for survivor selection

Individual	Fitness
$Parent_1 = [0\ 2\ 4\ 7\ 6]$	**1.05**
$Parent_2 = [0\ 1\ 5\ 3]$	0.27
$Child_1 = [0\ 9\ 4\ 5\ 3]$	0
$Child_2 = [0\ 1\ 3\ 7\ 6]$	**1.58**

Therefore, the two most fit individuals are still parent 1 and child 2. Child 2 is then added into the population pool.

Population:

$$Individual_1 = [0\ 2\ 4\ 7\ 6]$$

$$Individual_2 = [0\ 1\ 3\ 7\ 6]$$

$$Individual_3 = [0\ 1\ 5\ 3]$$

5.3.2.3 Iteration 2

Two random individuals are selected from the population pool for the next generation. The population is:

$$Indvidual_1 = [0\ 2\ 4\ 7\ 6]$$

$$Individual_2 = [0\ 1\ 3\ 7\ 6]$$

A pair of parents is selected from the set population.

Parents:

$$Parent_1 = [0\ 2\ 4\ 7\ 6]$$

$$Parent_2 = [0\ 1\ 3\ 7\ 6]$$

The crossover is now applied on the parents. The crossover points in each individual are arbitrarily chosen and a single-point crossover is conducted.

Note: The '.' represents the crossover point in each individual.

$$Parent_1 = [0\ 2\ 4\ 7.\ 6]$$

$$Parent_2 = [0\ 1.\ 3\ 7\ 6]$$

The resulting crossover generates the following children:

$$Child_1 = [0\ 2\ 4\ 7\ 3\ 7\ 6]$$

$$Child_2 = [0\ 1\ 6]$$

Following the crossover, the mutation algorithm is applied to each child. The mutation occurs on each gene with the probability based on the mutation rate. The mutation rate is dependent on the population size and abides the formula:

$$Mutation\ rate = \frac{1}{Population\ size}$$

Since the population size is 2, the mutation rate is 0.5. Mutation occurs if a randomly generated number from 0-1 is less than the mutation rate. A venue is arbitrary chosen

Note: The '*' represents that a mutation occurs in the gene.

$$Child_1 = [0\ 2\ 4^*\ 7\ 3\ 7^*\ 6] = [0\ 2\ 5\ 7\ 3\ 8\ 6]$$

$$Child_2 = [0\ 1^*\ 6^*] = [0\ 4\ 3]$$

The results from the mutation are as follows:

Table 10: Fitness values before and after mutation at iteration 2

Individual	Fitness
$Child_1 = [0\ 2\ 4\ 7\ 3\ 7\ 6]$	0
$Child_2 = [0\ 1\ 6]$	0.49
$Child_{1\ Mutation} = [0\ 2\ 5\ 7\ 3\ 8\ 6]$	0
$Child_{2\ Mutation} = [0\ 4\ 3]$	1.32

This mutation results in a higher fitness of child one. Therefore, mutation success rate is 2/4. This rate will be constantly updated in every mutation and will be recorded for 10 generations. Following the mutation, the total population and their total fitness is calculated below. The survivors are selected using an elitism algorithm. The two most fit individuals are selected for the next iteration.

Table 11: A comparison of all the fitness values after crossover and mutation at iteration 2. These values are used for survivor selection

Individual	Fitness
$Parent_1 = [0\ 2\ 4\ 7\ 6]$	**1.05**
$Parent_2 = [0\ 1\ 5\ 3]$	0
$Child_1 = [0\ 2\ 5\ 7\ 3\ 8\ 6]$	0
$Child_2 = [0\ 4\ 3]$	**1.32**

Therefore, a parent and a child are the most fit and thus, are the survivors of this generation. Note that since parent 1 is not added to the population pool because it already exists. On the other hand, child 2 doesn't exist in the population, thus it is added to the population pool.

Population:

$$Individual_1 = [0\ 2\ 4\ 7\ 6]$$

$$Individual_2 = [0\ 1\ 3\ 7\ 6]$$

$$Individual_3 = [0\ 1\ 5\ 3]$$

$$Individual_4 = [0\ 4\ 3]$$

5.4 Particle Swarm Optimization

5.4.1 Implementation Description

To solve the problem at hand, a variation of particle swarm optimization, known as binary particle swarm optimization (BPSO) [18], was applied. A synchronous version of BPSO is implemented.

5.4.1.1 Data Structure

Each particle in the system keeps track of the following properties:

1) Its own state. This is a vector of all possible attractions that can be visited. A value of 1 mean that the attraction will be visited and a value of 0 means the attraction will not be included in the final itinerary/solution.
2) Its personal best value. This a double value updated after every iteration using the fitness function.
3) Its personal best solution. This is a vector of all possible attractions and keeps track of the particle's best places to visit/solution so far and is updated every iteration. Consists of binary values similar to the state.
4) Its velocity. A vector keeping track of the velocity of every attraction for the particle and is updated every iteration.

The search space consists of a vector of particles which is updated every iteration. The particle object can be modelled as follows:

$$Particle = \begin{bmatrix} \begin{bmatrix} A_1 \\ A_2 \\ A_3 \\ \vdots \\ A_n \end{bmatrix} & \begin{bmatrix} V_1 \\ V_2 \\ V_3 \\ \vdots \\ V_n \end{bmatrix} & \begin{bmatrix} PBest_1 \\ PBest_2 \\ PBest_3 \\ \vdots \\ PBest_n \end{bmatrix} & PBestValue \end{bmatrix}$$

Where V_X is the velocity of attraction x, A_X is attraction x and $PBest_X$ is the best value for attraction x. PBestValue stores the personal best fitness function value for the particle.

5.4.1.2 Initialization

BPSO relies heavily on random numbers and does not have any constants that need to be initialized. A set of particles is initialized with different candidate solutions to the problem, which is the starting point for the algorithm. The velocity for each particle is set to zero and the personal best is set to the current state of the particle. Furthermore, the algorithm requires input data in order for it to be able to compute the fitness function. The static data that is required is

the commute time matrix between all attractions, the duration vector and the ratings vector for the attractions. The data being used is described in the Problem Formulation and Modelling section.

5.4.1.3 Swarm Update

All the particles in the swarm are updated at every iteration. The update procedure can be modelled in the following steps:

1) Update the velocity of all attractions of the particle as described below and compute the particles new position.
2) Evaluate the particles fitness using the fitness function and update the personal best if need be.
3) Analyze the personal best of all particles in the swarm and update the global best of the swarm if need be.

5.4.1.4 Velocity Calculation and Particle Movement

The velocity for each attraction in the particle is revaluated every iteration. The velocity calculation depends on the best global position, the current velocity, current particle position, the personal best position of the particle and uniformly generated random numbers. The following equation describes the velocity update procedure:

$$V_{t+1}^{id} = V_t^{id} + \phi_1\left(P_t^{id} - x_t^{id}\right) + \phi_2\left(P_t^{gd} - x_t^{id}\right)$$

Where:

- i = particle number.
- d = dimension or the attraction.
- V_x^{id} = velocity at iteration x for particle i and dimension d.
- P_x^{id} = personal best at iteration x for particle i and dimension d.
- P_x^{gd} = global best at iteration x for dimension d.
- X_x^{id} = current position at iteration x for particle i and dimension d.
- ϕ_1, ϕ_2 = uniformly generated random numbers between 0 and 2.

Once the velocity vector is updated, the sigmoid value of each of the velocities is updated as follows:

$$Sig\left(V_{t+1}^{id}\right) = \frac{1}{1 + e^{-V_{t+1}^{id}}}$$

A new vector with the sigmoid value of the velocities is created.

Next the particle position is updated as follows:

$$x_{t+1}^{id} = \begin{cases} 1 & ,if\ r < Sig\left(V_{t+1}^{id}\right) \\ 0 & ,otherwise \end{cases}$$

Where:

- i = particle number.
- d = dimension or the attraction.
- r = uniformly generated random number between 0 and 1.
- X_x^{id} = current position at iteration x for particle i and dimension d.
- $Sig\left(V_{t+1}^{id}\right)$ = sigmoid value of the velocity at V_{t+1}^{id}

5.4.1.5 Termination Criteria

The two conditions that will cause the algorithm to terminate are:

- The maximum number of iterations is reached. This is currently set to 1000.
- If the global best of the swarm does not change over 250 iterations, the algorithm will terminate.

5.4.2 Hand Iteration

Two iterations of the BPSO algorithm, as described above, are performed with 4 particles in the swarm. The static data required to compute the fitness function is explained below. For clarity, the data for each iteration is divided into tables and the structure of the tables is clarified below. The four particles are initialized and two iterations are performed on these particles.

5.4.2.1 Static Data

In order to compute the fitness function at every iteration, certain inputs are required. The commute time in minutes, the ratings and the duration values for each attraction id are required. The ID of zero is reserved for the starting venue which is defaulted to the hotel, this location does not have a duration or rating. The following tables show the commute time matrix and the ratings and duration matrix.

Table 12: The commute times between all attractions.

Commute Time (mins)										
ID		To								
		1	2	3	4	5	6	7	8	9
From	0	10	20	7	11	8	19	7	1	1
	1	0	12	14	1	7	12	10	22	22
	2	12	0	25	21	28	1	22	12	12
	3	14	25	0	15	15	25	4	6	6
	4	1	21	15	0	25	22	11	10	10
	5	7	28	15	25	0	28	14	28	29
	6	12	1	25	22	28	0	22	12	12
	7	10	22	4	11	14	22	0	3	2
	8	22	12	6	10	28	12	3	0	1
	9	22	12	6	10	29	12	2	1	0

Table 13: The duration and ratings for each attraction

ID	Rating	Duration (mins)
0	Starting hotel	
1	2	120
2	3	60
3	3	180
4	0	180
5	5	120
6	1	60
7	4	60
8	0	60
9	2	120

5.4.2.2 Data Description

Five tables are used to display the appropriate data after every iteration of the algorithm. The headings of the tables can be summarized as follows:

- X_Y = particle with id Y
- ϕ_1, ϕ_2 = uniformly generated random numbers between 0 and 2.
- V_{XY} = velocity of the Yth attraction after iteration X.
- $Sig(V_{XY})$ = sigmoid value of the velocity of the Yth attraction after iteration X.
- r_{XY} = uniformly generated random number used for determining the Yth attractions next position after iteration X.

- P_X = current binary value for the position of attraction X for a particle.
- $Total\ commute\ time$ = Total commute time for the particle.
- $Total\ places\ visited$ = Total number of places visited in the particle.
- $Average\ rating$ = Average rating of all places visited in the particle.
- $Total\ Duration$ = Total duration spent at all places that are visited in the particle.
- $F_Y(X_i)$ = Fitness value evaluated for that particle at iteration Y.
- $PBest_{XY}$ = Personal best value for attraction Y at iteration X.
- $PBestVal$ = Personal best fitness value.

5.4.2.3 Initialization

Initial velocities of the particle:

Table 14: Particle current velocity at initialization.

	ϕ_1	ϕ_2	V_{01}	V_{02}	V_{03}	V_{04}	V_{05}	V_{06}	V_{07}	V_{08}	V_{09}
X_1	0	0	0	0	0	0	0	0	0	0	0
X_2	0	0	0	0	0	0	0	0	0	0	0
X_3	0	0	0	0	0	0	0	0	0	0	0
X_4	0	0	0	0	0	0	0	0	0	0	0

Initial particle state and fitness function:

Table 15: Particle current state at initialization.

	P_1	P_2	P_3	P_4	P_5	P_6	P_7	P_8	P_9	Total Commute Time	Total Places Visited	Average Rating	Total Duration	$F_0(X_i)$
X_1	1	1	0	0	0	0	0	0	0	34	2	2.000	180	0.700
X_2	0	0	1	0	0	0	0	1	0	19	2	5.000	240	2.405
X_3	0	1	0	0	1	0	0	0	0	76	2	2.500	180	0.809
X_4	1	0	0	0	0	0	1	0	0	30	2	3.000	180	1.059

Initial personal best of the particle:

Table 16: Particle current personal best at initialization.

	$PBest_{01}$	$PBest_{02}$	$PBest_{03}$	$PBest_{04}$	$PBest_{05}$	$PBest_{06}$	$PBest_{07}$	$PBest_{08}$	$PBest_{09}$	$PBestVal$
X_1	1	1	0	0	0	0	0	0	0	0.700
X_2	0	0	1	0	0	0	0	1	0	2.405
X_3	0	1	0	0	1	0	0	0	0	0.809
X_4	1	0	0	0	0	0	1	0	0	1.059

Determine global best value and global best position of the swarm after initialization:

$$GBest = \begin{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \end{bmatrix} & 2.405 \end{bmatrix}$$

5.4.2.4 Iteration 1

Updated velocities of the particle after iteration 1:

Table 17: Particle current velocity after iteration 1.

	ϕ_1	ϕ_2	V_{11}	V_{12}	V_{13}	V_{14}	V_{15}	V_{16}	V_{17}	V_{18}	V_{19}
X_1	0.958	0.830	-	-	0.830	0.000	0.000	0.000	0.000	0.830	0.000
X_2	1.347	1.517	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
X_3	1.320	1.649	0.000	-	1.649	0.000	-	0.000	0.000	1.649	0.000
X_4	0.757	0.678	-	0.000	0.678	0.000	0.000	0.000	-	0.678	0.000

Updated sigmoid velocity values of the particle after iteration 1:

Table 18: Particle current sigmoid velocity after iteration 1.

	$Sig(V_{11})$	$Sig(V_{12})$	$Sig(V_{13})$	$Sig(V_{14})$	$Sig(V_{15})$	$Sig(V_{16})$	$Sig(V_{17})$	$Sig(V_{18})$	$Sig(V_{19})$
X_1	0.304	0.304	0.696	0.500	0.500	0.500	0.500	0.696	0.500
X_2	0.500	0.500	0.500	0.500	0.500	0.500	0.500	0.500	0.500
X_3	0.500	0.161	0.839	0.500	0.161	0.500	0.500	0.839	0.500
X_4	0.337	0.500	0.663	0.500	0.500	0.500	0.337	0.663	0.500

Uniformly generated random numbers for deciding particle updated after iteration 1:

Table 19: Particle current random decision values after iteration 2.

	r_{11}	r_{12}	r_{13}	r_{14}	r_{15}	r_{16}	r_{17}	r_{18}	r_{19}
X_1	0.477	0.724	0.875	0.654	0.088	0.089	0.853	0.925	0.528
X_2	0.673	0.530	0.438	0.785	0.218	0.763	0.838	0.749	0.590
X_3	0.534	0.086	0.301	0.763	0.653	0.754	0.809	0.974	0.763
X_4	0.218	0.697	0.875	0.854	0.116	0.941	0.678	0.742	0.965

Updated particle state and fitness function after iteration 1:

Table 20: Particle current state after iteration 1.

	P_1	P_2	P_3	P_4	P_5	P_6	P_7	P_8	P_9	Total Commute Time	Total Places Visited	Average Rating	Total Duration	$F_1(X_i)$
X_1	0	0	0	0	1	1	0	0	0	64	2	3.500	180	1.158
X_2	0	0	1	0	1	0	0	0	0	37	2	5.000	300	2.901
X_3	0	1	1	0	0	0	0	0	0	70	2	2.500	240	1.091
X_4	1	0	0	0	1	0	0	0	0	24	2	4.500	240	2.143

Updated personal best of the particle after iteration 1:

Table 21: Particle current personal best after iteration 1.

	$PBest_{11}$	$PBest_{12}$	$PBest_{13}$	$PBest_{14}$	$PBest_{15}$	$PBest_{16}$	$PBest_{17}$	$PBest_{18}$	$PBest_{19}$	$PBestVal$
X_1	0	0	0	0	1	1	0	0	0	1.158
X_2	0	0	1	0	1	0	0	0	0	2.901
X_3	0	1	1	0	0	0	0	0	0	1.091
X_4	1	0	0	0	1	0	0	0	0	2.143

Determine global best value and global best position of the swarm after iteration 1:

$$GBest = \begin{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} & 2.901 \end{bmatrix}$$

5.4.2.5 Iteration 2

Updated velocities of the particle after iteration 2:

Table 22: Particle current velocity after iteration 2.

	ϕ_1	ϕ_2	V_{21}	V_{22}	V_{23}	V_{24}	V_{25}	V_{26}	V_{27}	V_{28}	V_{29}
X_1	0.242	0.774	0.875	0.654	0.862	0.089	0.079	0.151	0.528	0.774	0.000
X_2	0.018	1.604	0.438	0.785	0.218	0.763	-	0.749	0.590	1.604	0.000
X_3	0.772	0.045	0.301	0.718	0.653	0.754	0.809	0.974	0.763	0.045	0.000
X_4	0.679	0.980	-	0.854	1.096	0.941	-	0.742	0.965	0.980	0.000

Updated sigmoid velocity values of the particle after iteration 2:

Table 23: Particle current sigmoid velocity after iteration 2.

	$Sig(V_{21})$	$Sig(V_{22})$	$Sig(V_{23})$	$Sig(V_{24})$	$Sig(V_{25})$	$Sig(V_{26})$	$Sig(V_{27})$	$Sig(V_{28})$	$Sig(V_{29})$
X_1	0.706	0.658	0.703	0.522	0.520	0.538	0.629	0.684	0.500
X_2	0.608	0.687	0.554	0.682	0.317	0.679	0.643	0.833	0.500
X_3	0.575	0.672	0.658	0.680	0.692	0.726	0.682	0.511	0.500
X_4	0.474	0.701	0.750	0.719	0.425	0.677	0.724	0.727	0.500

Uniformly generated random numbers for deciding particle updated after iteration 2:

Table 24: Particle current random decision values after iteration 2.

	r_{21}	r_{22}	r_{23}	r_{24}	r_{25}	r_{26}	r_{27}	r_{28}	r_{29}
X_1	0.803	0.329	0.183	0.865	0.961	0.693	0.667	0.275	0.874
X_2	0.917	0.951	0.371	0.207	0.982	0.794	0.983	0.165	0.845
X_3	0.600	0.390	0.385	0.768	0.054	0.880	0.962	0.874	0.924
X_4	0.605	0.918	0.795	0.490	0.787	0.733	0.894	0.854	0.648

Updated particle state and fitness function after iteration 2:

Table 25: Particle current state after iteration 2.

	P_1	P_2	P_3	P_4	P_5	P_6	P_7	P_8	P_9	Total Commute Time	Total Places Visited	Average Rating	Total Duration	$F_2(X_i)$
X_1	0	1	1	0	0	0	0	1	0	63	3	3.333	300	1.842
X_2	0	0	1	1	0	0	0	1	0	38	3	4.333	420	3.514
X_3	0	1	1	0	1	0	0	0	0	88	3	3.333	360	2.113
X_4	0	0	0	1	0	0	0	0	0	37	1	3.000	180	1.044

Updated personal best of the particle after iteration 2:

Table 26: Particle personal best after iteration 2.

	$PBest_{21}$	$PBest_{22}$	$PBest_{23}$	$PBest_{24}$	$PBest_{25}$	$PBest_{26}$	$PBest_{27}$	$PBest_{28}$	$PBest_{29}$	$PBestVal$
X_1	0	1	1	0	0	0	0	1	0	2.228
X_2	0	0	1	1	0	0	0	1	0	3.831
X_3	0	1	1	0	1	0	0	0	0	2.629
X_4	1	0	0	0	1	0	0	0	0	2.357

Determine global best value and global best position of the swarm after iteration 2:

$$GBest = \begin{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \end{bmatrix} & 3.514 \end{bmatrix}$$

5.5 Ant Colony Optimization

In Ant Colony optimization, viable solutions are developed in an incremental approach modelled by hypothetical ants where each ant determines the next attraction in the itinerary depending on the concentration of pheromone deposited by previous ants. The pheromone on all paths is set to an initial value in the initialization step, and then modified via evaporation and updating. The purpose of the pheromone is to provide feedback of paths travelled by previous ants in terms of the quality of the solution. In terms of the algorithm, the pheromone is used at each node to evaluate the transition probability. Then a random value is chosen in the range of [0,1] which determines which of the nodes to visit next by evaluating each node one-by-one to see whether the transition probability of that node is greater than or equal to the random value. The first node that satisfies the condition is chosen as the next node in the solution and this process is repeated until the termination criteria is met.

5.5.1 Design Decisions

5.5.1.1 Data Structures

The solution of each ant is represented as a vector of IDs that correspond to the node that they have traversed in each solution. The distance and rating of each venue in the population is stored in vectors where the index value represents the ID of the venue. The graph of possible venues is represented by a matrix that stores the commute times between each of the venues. The venues are identified by their index value in the matrix that corresponds to the ID of the venue. The pheromone concentration on each of the paths is also stored in a matrix with the same structure, where the value of the index [i,j] represents the pheromone concentration on the path between the venues i and j. The initial matrix of pheromone concentrations is set to 1 and the initial ant paths are represented as empty vectors.

5.5.1.2 Movement of Ants

The movement of ants is determined by transition probability function, which has been customized to include the constraints of our problem. The transition probability of each viable node that has not been visited yet will be calculated as follows:

$$P_{ij} = \begin{cases} \dfrac{\tau_{iu}^{\alpha}\eta_{iu}^{\beta}}{\sum_{u \in N_i^k} \tau_{iu}^{\alpha}\eta_{iu}^{\beta}} & if\ j \in N_i^k \\ 0 \quad if\ j \notin N_i^k \end{cases}$$

Where:

- τ_{iu} is the pheromone concentration on the path from *i* to *u*
- α is the influence of the pheromone
- β is the influence of the desirability function
- N_i^k is the set of viable nodes connected to *i* and have not been visited with respect to ant *k*
- η_{iu} is desirability of the node as defined as:

$$\eta_{ij} = \begin{cases} \dfrac{R}{1 + C_{ij}} & if\ D_R + D_j + C_{ij} \leq D_{max} \\ 0 \quad if\ D_R + D_j + C_{ij} > D_{max} \end{cases}$$

$$D_{max} = 480$$

Where:

- D_R is the running duration in minutes of the current list of venues in the solution
- D_j is the duration in minutes of the venue *j*
- D_{max} is the maximum duration of the day which is 8 hours * 60 minutes = 480 minutes
- C_{ij} is the commute time in minutes between venues *i* and *j*
- R is the rating of the venue

This transition probability is computed for each node, and then a random number is chosen from [0,1] to introduce randomness in the choice of the next venue. This process continues until the limit for the duration of 480 minutes has been reached, at this point all η_{ij} become 0 and the

transition probabilities become 0's. Once a full path is found, it is then evaluated with the global fitness function.

5.5.1.3 Pheromone Evaporation and Pheromone Update

The evaporation of the pheromone concentrations in the pheromone concentration matrix will occur at the end of each of the iterations once all ants have formed a solution. The evaporation of the pheromone will cause the pheromone between all venues to decrease to allow for more exploration of new paths. The evaporation rate used in this is implementation is 0.5.

After the step of the evaporation of the pheromone, the paths between the venues for each of the solutions generated by each of the ants are evaluated using the global fitness function and the Ant-Cycle pheromone update method is used to update the pheromone along the path with the maximum fitness. The function used for this update method is:

$$\tau_{ij}(t+1) = (1-\rho)\tau_{ij}(t) + \Delta\tau_{ij}(t)$$

$$\Delta\tau_{ij}(t) = QL^{k}(t)$$

Where $L^{k}(t)$ is the global fitness function and Q is chosen to be $Q = 1$. This will eventually result in edges with better solutions having higher pheromone concentrations, increasing the probability of that path being traversed by future ants.

5.5.2 Hand iterations

For the purposes of the hand iterations, the problem will be simplified to scale down to 9 possible venues and a max duration of the itinerary to be 4 hours, which is 240 minutes. The iterations will closely follow the Ant System algorithm with a modified evaluation function and the pheromone will be updated on all paths according to ant cycle pheromone update method. In the MATLAB implementation, the pheromone is updated on the best path but for the purposes of the hand iterations, they are done on all paths because the input data is very small and may converge too quickly if done this way for demonstration purposes.

5.5.2.1 Initialization

Two iterations of the Ant System optimization algorithm will be done with two ants each. The first step will be to initialize the pheromone matrix with the value of 1. The reason why there are

0s in the diagonal is because those indexes correspond to a non-existent path (ie. path from venue 1 to venue 1). The solutions of Ant 1 and Ant 2 are initialized to a vector with ID 0 and the evaporation rate is set to 0.5. The commute time in minutes is known and is displayed as a matrix, the ratings and the duration values are displayed in a separate matrix. The ID of zero is reserved for the starting venue which is defaulted to the hotel, this location does not have a duration or rating.

Table 27: Initialization of Pheromone (left) and Commute Time (right) Matrices

Initial Pheromone (mins)										
ID		To								
		1	2	3	4	5	6	7	8	9
From	0	1	1	1	1	1	1	1	1	1
	1	0	1	1	1	1	1	1	1	1
	2	1	0	1	1	1	1	1	1	1
	3	1	1	0	1	1	1	1	1	1
	4	1	1	1	0	1	1	1	1	1
	5	1	1	1	1	0	1	1	1	1
	6	1	1	1	1	1	0	1	1	1
	7	1	1	1	1	1	1	0	1	1
	8	1	1	1	1	1	1	1	0	1
	9	1	1	1	1	1	1	1	1	0

Commute Time (mins)										
ID		To								
		1	2	3	4	5	6	7	8	9
From	0	10	20	7	11	8	19	7	1	1
	1	0	12	14	1	7	12	10	22	22
	2	12	0	25	21	28	1	22	12	12
	3	14	25	0	15	15	25	4	6	6
	4	1	21	15	0	25	22	11	10	10
	5	7	28	15	25	0	28	14	28	29
	6	12	1	25	22	28	0	22	12	12
	7	10	22	4	11	14	22	0	3	2
	8	22	12	6	10	28	12	3	0	1
	9	22	12	6	10	29	12	2	1	0

Table 28: Table of ratings and durations of venues

ID	Rating	Duration (mins)
0	Starting hotel	
1	2	120
2	3	60
3	3	180
4	0	180
5	5	120
6	1	60
7	4	60
8	0	60
9	2	120

Evaporation rate is set to 0.5, α=1, β=1, Q=1, Ant 1 = [0], Ant 2 = [0], Dmax = 240

5.5.2.2 Iteration 1

5.5.2.2.1 Movement of ants

The ants will begin at venue with ID of 0 and find a solution by evaluating the transition probabilities at each step. Computing the probabilities at the first step and then using that data and applying the roulette wheel selection method, we get the ranges of which a random number would correspond to the venue.

Table 29: Transition probabilities and probability ranges for venue 0 for the first iteration

Probability Ant 1										
ID		To (j)								
		1	2	3	4	5	6	7	8	9
$P_{ij}{}^{1}$	0	0.065	0.051	0.134	0.000	0.198	0.018	0.178	0.000	0.356
Probability Range		0.000	0.065	0.116		0.249	0.447	0.465		0.644

If we get a random value of r to be 0.33, then it is clear that venue 5 is the next venue in the solution.

Ant 1 = [0, 5], RunningDuration1 = 120 + 8 = 128 minutes.

We repeat this process until we reach our hard constraint for the reduced problem which is a max duration of 240 mins.

Table 30: Transition probabilities and probability ranges for venue 5 for the first ant in the first iteration

Probability Ant 1									
ID		To (j)							
		1	2	3	4	6	7	8	9
$P_{ij}{}^{1}$	5	0.000	0.256	0.000	0.000	0.085	0.659	0.000	0.000
Probability Range			0.000			0.256	0.341		

Choose random number r to be 0.67. Then venue 7 is the next venue in the solution.

Ant 1 = [0, 5, 7], RunningDuration1 = 128 + 60 + 14= 202 minutes.

Now in the next iteration, since we will be approaching the maximum duration of the itinerary, all venues will have a desirability of 0 because the addition of the duration and commute time of the corresponding venue and commute time will result in the hard constraint being violated. Hence we have met our termination criteria.

Table 31: Total duration if given venues are added to the existing solution

Venue ID	Duration (mins)	Commute Time	Total Duration (mins)
1	120	10	332
2	60	22	284
3	180	4	386
6	60	22	284
9	120	2	324

Ant 1 = [0, 5, 7], RunningDuration1 = 202 minutes.

Now, the same process is repeated by the second ant, we can use the same probability data as we did for the first ant since the transition probabilities are constant from the initial venue 0. If a random number is chosen to be 0.21, venue 3 is chosen.

Ant 2 = [0, 3], RunningDuration2 = 180 + 7 = 187 minutes.

There are no further nodes that can be visited without conflicting with the hard constraint, hence we have a final solution for ant 2.

Ant 2 = [0, 3], RunningDuration2 = 187 minutes.

That terminates the solution of the second ant as there are no viable venues with a duration that can fit in the solution that will not violate the hard constraint.

Therefore the two solutions found in the first iteration were [0, 5, 7] and [0, 3], respectively.

5.5.2.2.2 Pheromone evaporation

The evaporation of pheromone is computed as:

$$\tau_{ij}(t+1) = (1-\rho)\tau_{ij}(t)$$

$$\tau_{ij}(t+1) = (1-0.5)\tau_{ij}(t) = (0.5)\tau_{ij}(t)$$

This results in the following pheromone matrix after evaporation:

Table 32: Pheromone matrix after evaporation

Evaporated Pheromone										
ID		To								
		1	2	3	4	5	6	7	8	9
From	0	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5
	1	0	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5
	2	0.5	0	0.5	0.5	0.5	0.5	0.5	0.5	0.5
	3	0.5	0.5	0	0.5	0.5	0.5	0.5	0.5	0.5
	4	0.5	0.5	0.5	0	0.5	0.5	0.5	0.5	0.5
	5	0.5	0.5	0.5	0.5	0	0.5	0.5	0.5	0.5
	6	0.5	0.5	0.5	0.5	0.5	0	0.5	0.5	0.5
	7	0.5	0.5	0.5	0.5	0.5	0.5	0	0.5	0.5
	8	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0	0.5
	9	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0

5.5.2.2.3 Pheromone Update

The pheromone update step requires a calculation to determine the fitness value of the solutions generated by each ant.

$$f(solution) = \frac{avgRating}{1 + \frac{\sum avgCommuteTime}{maxDuration}} \left(\frac{\sum durationsInSolution}{maxDuration}\right)$$

$$f([0,5,7]) = \frac{\frac{5+4}{2}}{1 + \frac{8+14}{240}} \left(\frac{120+60}{240}\right) = 3.0916$$

$$f([0,3]) = \frac{\frac{3}{1}}{1 + \frac{7}{240}} \left(\frac{180+7}{240}\right) = 2.2712$$

These results are applied to the pheromone matrix using the formula, where the greyed out term has already been accounted for in the evaporation step:

$$\tau_{ij}(t+1) = (1-\rho)\tau_{ij}(t) + \Delta\tau_{ij}(t)$$

$$\Delta\tau_{ij}(t) = QL^{k}(t) = (1)(f(solution))$$

This means that the pheromone concentrations on each of the paths of each solution are incremented by the value of the corresponding solution's global fitness value. This results in the following updated pheromone concentration matrix:

Table 33: Pheromone matrix after update

Updated Pheromone 1										
ID		To								
		1	2	3	4	5	6	7	8	9
From	0	0.500	0.500	2.771	0.500	3.591	0.500	0.500	0.500	0.500
	1	0.000	0.500	0.500	0.500	0.500	0.500	0.500	0.500	0.500
	2	0.500	0.000	0.500	0.500	0.500	0.500	0.500	0.500	0.500
	3	0.500	0.500	0.000	0.500	0.500	0.500	0.500	0.500	0.500
	4	0.500	0.500	0.500	0.000	0.500	0.500	0.500	0.500	0.500
	5	0.500	0.500	0.500	0.500	0.000	0.500	3.591	0.500	0.500
	6	0.500	0.500	0.500	0.500	0.500	0.000	0.500	0.500	0.500
	7	0.500	0.500	0.500	0.500	3.591	0.500	0.000	0.500	0.500
	8	0.500	0.500	0.500	0.500	0.500	0.500	0.500	0.000	0.500
	9	0.500	0.500	0.500	0.500	0.500	0.500	0.500	0.500	0.000

5.5.2.3 Iteration 2

The pheromone matrix at the start of the second iteration is that computed after the pheromone update in the first iteration. The solutions of both the ants are reinitialized back to [0] and the running totals for the durations are also reset to 0. Computing the transition probability for the second iteration for the first ant results in the following transition probabilities:

Table 34: Transition probabilities and probability ranges for venue 0 for the second iteration

Probability Ant 1										
ID		To (j)								
		1	2	3	4	5	6	7	8	9
$P_{ij}{}^{1}$	0	0.022	0.017	0.249	0.000	0.527	0.006	0.060	0.000	0.120
Probability Range		0.000	0.022	0.039		0.287	0.815	0.821		0.880

Choose random number r to be 0.74. Then venue 5 is the next venue in the solution.

Ant1 [0, 5], RunningDuration1 = 120 + 8 = 128 minutes.

Now we compute the transition probabilities from venue 5, many have zero probability because the addition of the duration of and commute times to the venue violates the hard constraint:

Table 35: Transition probabilities and probability ranges for venue 5 for the first ant in the second iteration

Probability Ant 1									
ID		To (j)							
		1	2	3	4	6	7	8	9
$P_{ij}{}^{1}$	5	0.000	0.046	0.000	0.000	0.015	0.939	0.000	0.000
Probability Range			0.000			0.046	0.061		

Choose random number r to be 0.0294. Then venue 2 is the next venue in the solution.

Ant1 = [0, 5, 2], RunningTotal1= 128 + 60 +28 = 216 minutes.

This is the final venue and the termination of the solution since no additional venues are viable that can be added that hold the hard constraint.

Ant1 = [0, 5, 2], RunningTotal1= 216 minutes.

Now the second ant will begin constructing its solution for the second iteration. Again the transition probabilities from venue 0 are already calculated from the first ant in the second iteration. Choose a random number r to be 0.845, this means that the next node in the solution will be venue 7.

Ant2 = [0, 7], RunningTotal2 = 60 + 7 = 67 minutes.

Now computing the transition probabilities from venue 7 to the viable venues produces the following probabilities:

Table 36: Transition probabilities and probability ranges for venue 7 for the second ant in the second iteration

Probability Ranges									
ID		To (j)							
		1	2	3	4	5	6	8	9
$P_{ij}{}^{2}$	7	0.134	0.096	0.000	0.000	0.246	0.032	0.000	0.492
Probability Range		0.000	0.134			0.230	0.476	0.508	0.508

Choose random number r to be 0.303. Then the next venue will be venue 5.

Ant2 = [0, 7, 5], RunningTotal2 = 67 + 120 + 14 = 201 minutes.

The iteration is now terminated because no further venues can be visited. The solutions are:

Ant1 = [0, 5, 2], RunningTotal1 = 216 minutes.

Ant2 = [0, 7, 5], RunningTotal2 = 201 minutes.

5.5.2.3.1 Pheromone Evaporation

Applying the pheromone evaporation on the pheromone concentration matrix results in the following matrix:

Table 37: Pheromone matrix after evaporation

Evaporated Pheromone										
ID		To								
		1	2	3	4	5	6	7	8	9
From	0	0.250	0.250	1.386	0.250	1.796	0.250	0.250	0.250	0.250
	1	0.000	0.250	0.250	0.250	0.250	0.250	0.250	0.250	0.250
	2	0.250	0.000	0.250	0.250	0.250	0.250	0.250	0.250	0.250
	3	0.250	0.250	0.000	0.250	0.250	0.250	0.250	0.250	0.250
	4	0.250	0.250	0.250	0.000	0.250	0.250	0.250	0.250	0.250
	5	0.250	0.250	0.250	0.250	0.000	0.250	1.796	0.250	0.250
	6	0.250	0.250	0.250	0.250	0.250	0.000	0.250	0.250	0.250
	7	0.250	0.250	0.250	0.250	1.796	0.250	0.000	0.250	0.250
	8	0.250	0.250	0.250	0.250	0.250	0.250	0.250	0.000	0.250
	9	0.250	0.250	0.250	0.250	0.250	0.250	0.250	0.250	0.000

5.5.2.3.2 Pheromone Update

The pheromone update step requires a calculation to determine the fitness value of the solutions generated by each ant.

$$f(solution) = \frac{avgRating}{1 + avgCommuteTime}\left(\frac{\sum durationsInSolution}{maxDuration}\right)$$

$$f([0,5,2]) = \frac{\frac{5+3}{2}}{1+\frac{8+28}{240}}\left(\frac{120+60}{240}\right) = 2.6086$$

$$f([0,7,5]) = \frac{\frac{3+4}{2}}{1+\frac{7+14}{240}}\left(\frac{60+120}{240}\right) = 2.4137$$

These results are applied to the pheromone matrix using the formula, where the greyed out term has already been accounted for in the evaporation step:

$$\tau_{ij}(t+1) = (1-\rho)\tau_{ij}(t) + \Delta\tau_{ij}(t)$$

$$\Delta\tau_{ij}(t) = QL^k(t) = (1)(f(solution))$$

The result of the pheromone update is:

Table 38: Pheromone matrix after update

Updated Pheromone 2										
ID		To								
		1	2	3	4	5	6	7	8	9
From	0	0.250	0.250	1.386	0.250	4.404	0.250	2.664	0.250	0.250
	1	0.000	0.250	0.250	0.250	0.250	0.250	0.250	0.250	0.250
	2	0.250	0.000	0.250	0.250	2.859	0.250	0.250	0.250	0.250
	3	0.250	0.250	0.000	0.250	0.250	0.250	0.250	0.250	0.250
	4	0.250	0.250	0.250	0.000	0.250	0.250	0.250	0.250	0.250
	5	0.250	2.859	0.250	0.250	0.000	0.250	4.209	0.250	0.250
	6	0.250	0.250	0.250	0.250	0.250	0.000	0.250	0.250	0.250
	7	0.250	0.250	0.250	0.250	4.209	0.250	0.000	0.250	0.250
	8	0.250	0.250	0.250	0.250	0.250	0.250	0.250	0.000	0.250
	9	0.250	0.250	0.250	0.250	0.250	0.250	0.250	0.250	0.000

This is the end of the second iteration of the Ant System algorithm with ant cycle pheromone update method.

6 Results

6.1 Overview

The results of the five metaheuristic algorithms are outlined as follows.

6.1.1 Tabu Search

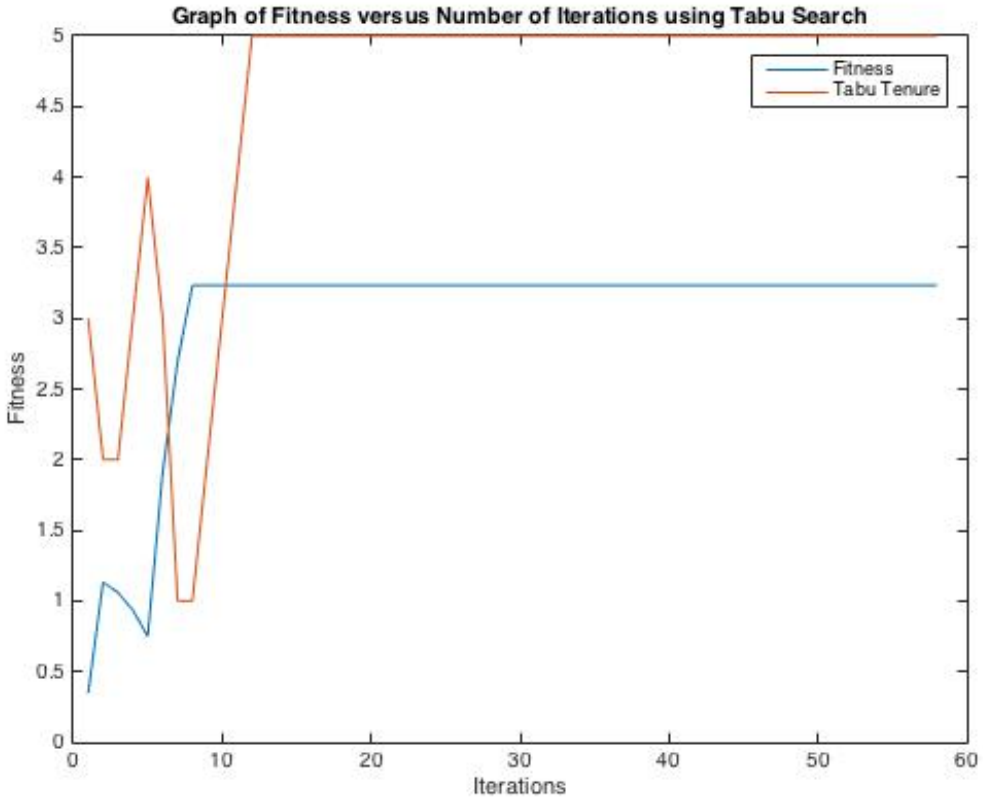

Figure 1: Graph indicating the change in Fitness and Tabu Tenure as iterations proceed

From Figure 1, we can see that the solution is continually improving as time progresses. Deviations occur as the algorithm proceeds and this is due to the fact that Tabu Search accepts solutions that decrease fitness in hopes of finding a solution that will improve as a whole. The length of the tabu tenure changes dynamically as the solution is in an improving or degrading stage, as discussed in [2]. As the algorithm approached the best solution, the length of the tabu tenure is set to 1, and as the algorithm tends to stabilize at a maximum, the tabu tenure is incremented. This is done so that it can potentially get out of a local maximum. Since the fitness of the solution did not change over the past 50 iterations, the algorithm terminates and returns the

best-found result, stopping premature to the initially set 1000 iterations, saving computation power. The operators used, addition and remove proved to be effective, as neighbours involved frequent changes in the attraction utilized in a solution.

6.1.2 Simulated Annealing

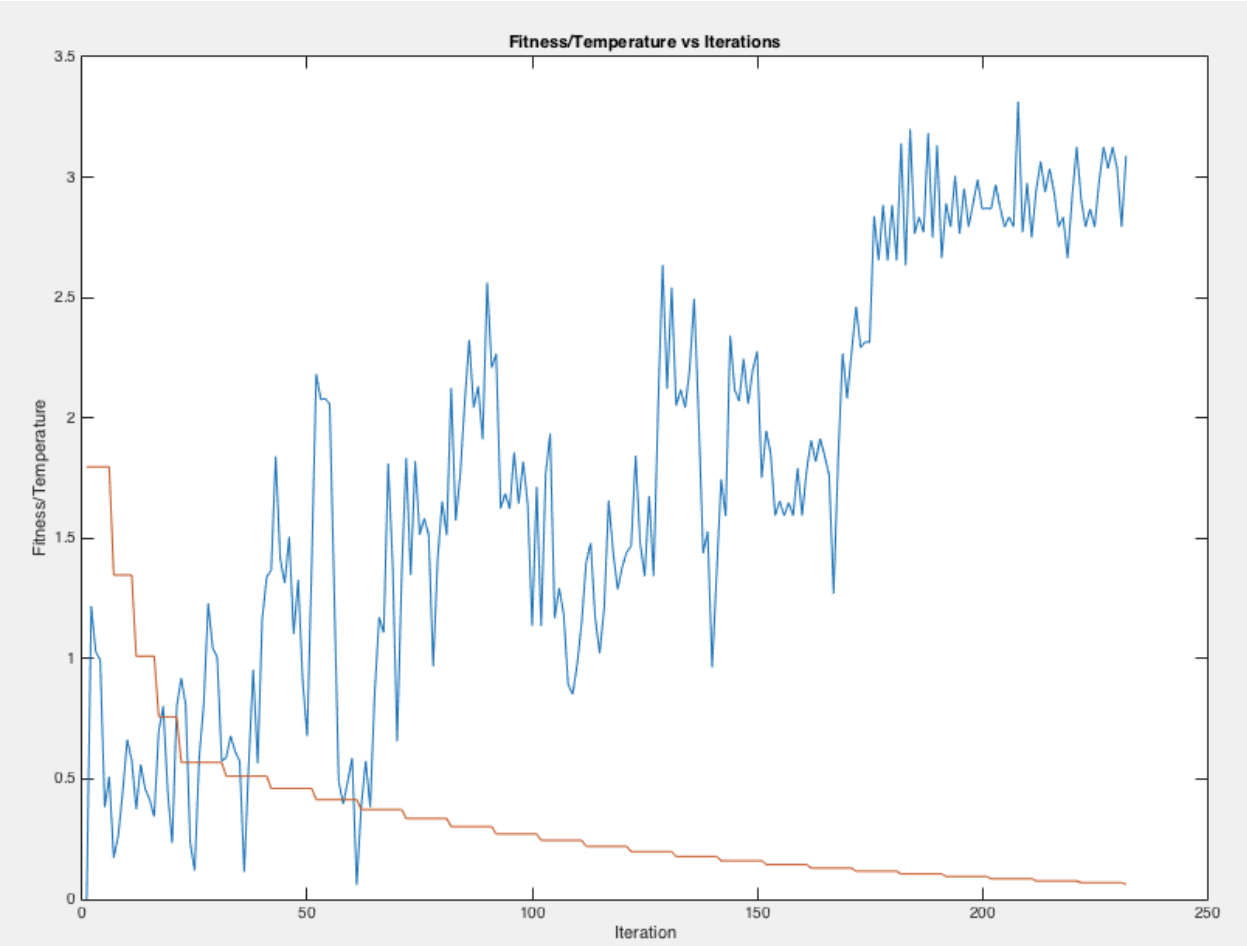

Figure 2: Fitness and Temperature vs Iterations for maxIterations = 10000 and maxConsRej = 1000

Figure 2 shows one run of the Simulated Annealing algorithm, showing the temperature decay over time as well as the steady increase of the best overall fitness. It can be seen that exploration occurs more when the temperature is high, meaning that worse solutions are more readily being accepted. This is good for escaping local optima and uncovering more of the global sample space. The temperature decreases dramatically initially to speed up convergence; once it dips below a specified temperature the cooling coefficient is raised to stabilize the system. This

allows the algorithm to increase intensification and eventually accept only solutions that are improving. From Figure 2, this is evident near the tail end of the algorithm.

6.1.3 Genetic Algorithm

The genetic algorithm implementation of our problem is illustrated in Figure 3. Figure 3 displays the best solution created by genetic algorithm.

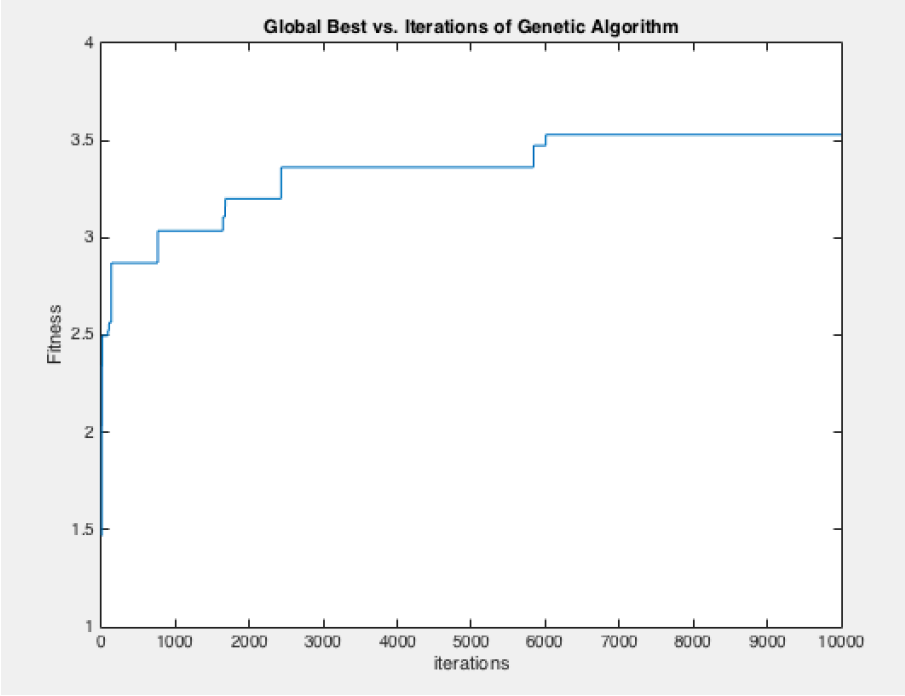

Figure 3: Global Best vs. Iterations of Genetic Algorithms over 10000 iterations.

From Figure 3, it is evident that the genetic algorithm reaches the maximum iterations. During the first 3000 iterations, there exists a significant increase in global fitness function. Therefore, the best solutions are created early on and the solutions after this point stagnate for a long duration of time before further improving. Therefore, it is evident that genetic algorithm eventually reaches considerable optimal results, however it requires a large portion of time. This is partially because elitism is used as a selection operator, thus it potentially results in a decreased diversity. With a decreased diversity, there is a high possibility of converging to a local maximum. After the 3000-iteration mark, it is unable to crossover better individuals than the existing most fit individual, thus the results begin to stagnate. Moreover, only the best 4 individuals are selected in the selection process, therefore the crossovers among the parents are most likely repeating the same individuals resulting in very small variations. This will result in

suboptimal solutions, hence the genetic algorithm is unable to improve and it begins to stagnate. Moreover, S.Khuri, et al. [3] stated that genetic algorithm is not a good algorithm for a large population size. Since our problem holds a large population, the results confirm this fact because the algorithm is unable to converge quickly.

6.1.4 Particle Swarm Optimization

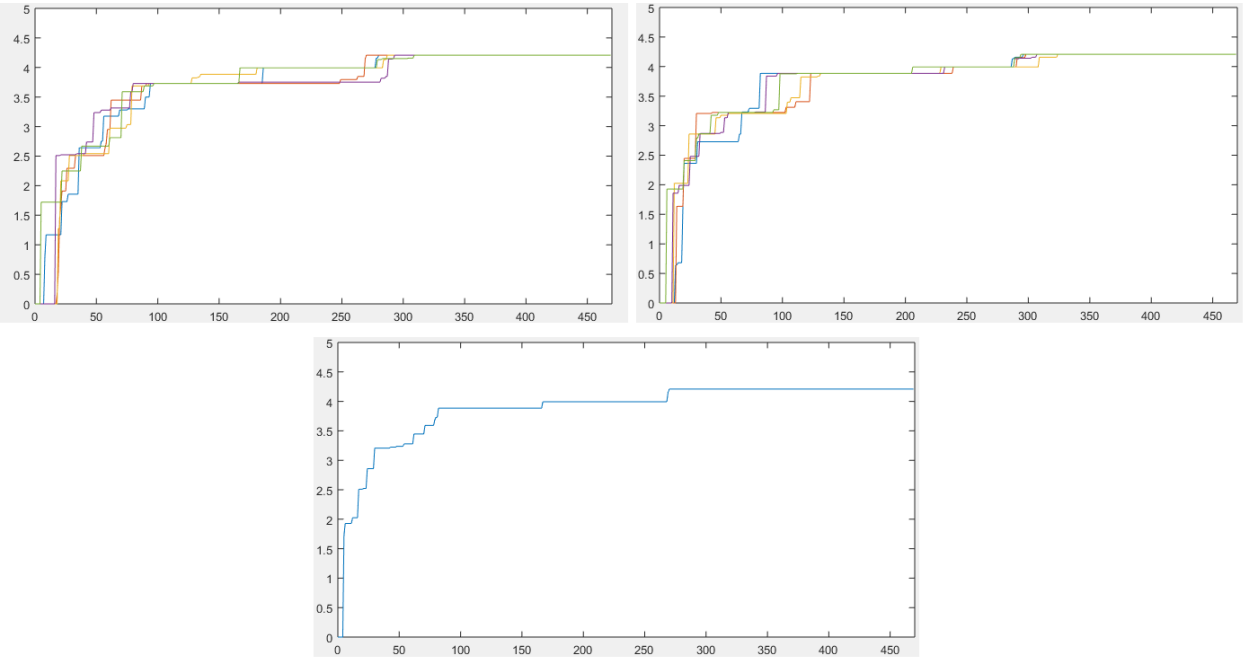

Figure 4: Personal best of 5 particles in swarm 1 (a - top left), personal best of 5 particles in swarm 2 (b – top right), and the global best of both swarms(c – top right) as 1000 iterations are run.

Figure 4.a and Figure 4.b show how the personal best of two swarms with 5 particles each evolves as more and more iterations are run. Similarly, Figure 4.c illustrates how the global best of both swarms evolves as the algorithm progresses. Observing the figures, it is evident that both the global best and the personal best rise very steeply initially to a suboptimal solution and slowly improve thereafter. An interesting observation in Figure 4 is how information is exchanged between the swarms. The first particle in swarm 1 updates to the optimal value at approximately the 270^{th} iteration. Once the information is shared with the second swarm we observe the first particle in swarm 2 reach the optimal value at iteration 290.Furthermore, PSO is quick to converge to an optimal value and the obtained fitness value is within an acceptable range.

6.1.5 Ant Colony Optimization

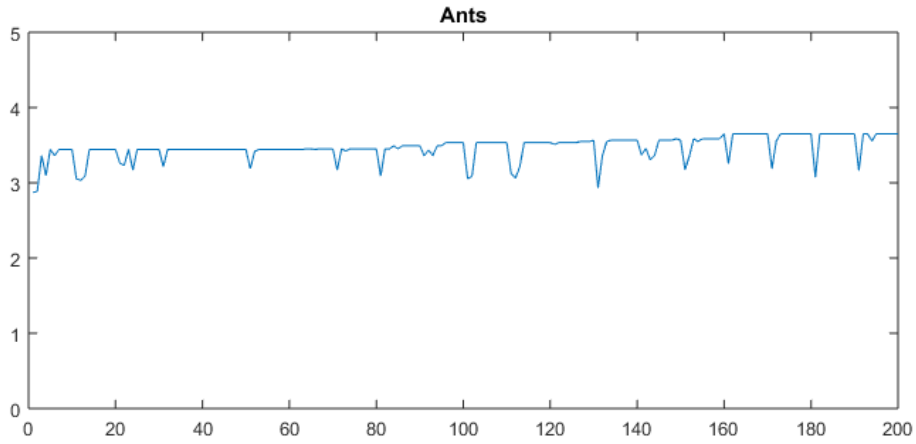

Figure 5: Maximum fitness value among ants in each iteration

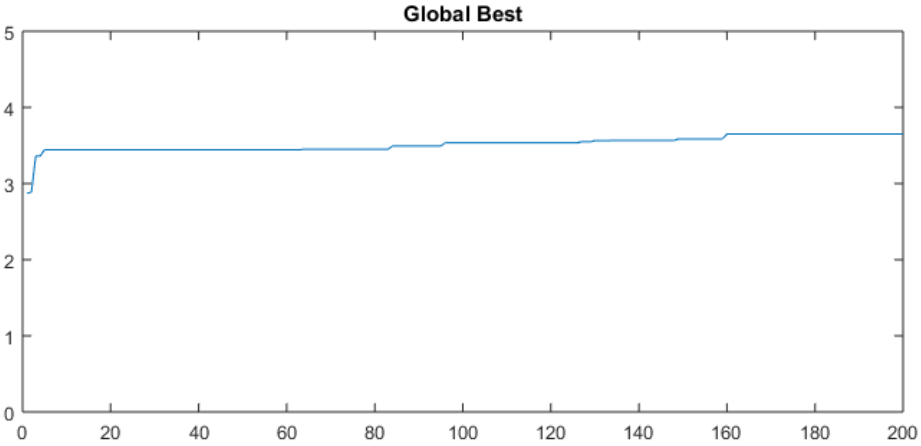

Figure 6: Global best solution over each iteration

Figure 5 above displays the maximum fitness values between each of the ants per iteration. This graph has noise because of the roulette wheel selection, which introduces randomness in the paths the ants take. Figure 6 displays the fitness value the global best solution over the iterations. The ACO implementation reaches considerable optimal results within the first 20 iterations of the algorithm after which it slowly progress to better solutions as the search intensifies on the previously found optimal solution. The degree of randomness allows the ants to explore solutions around the current optimal solution, but there is still a considerable level of stagnation evident where the solutions are non-improving. This can be due to the pheromone update of the global

best preventing ants from exploring new solutions as stated by G. Liu, J. Xiong [17]. In terms of speed, the ACO optimization algorithm converges quickly to a considerable optimal solution, but requires many more iterations to improve, if at all.

6.2 Algorithm Fitness

The optimal algorithm is determined through a comparison of several parameters. These include their average fitness function, standard deviation and their computation performance. This information will indicate the overall effectiveness of each algorithm with respect to our problem.

6.2.1 Accuracy

The average fitness function of each algorithm indicates whether the algorithm is able to identify the optimal or near optimal solution. This is measured in figure 7. In order to reduce any bias and experimental errors, all algorithms are run the same number of iterations.

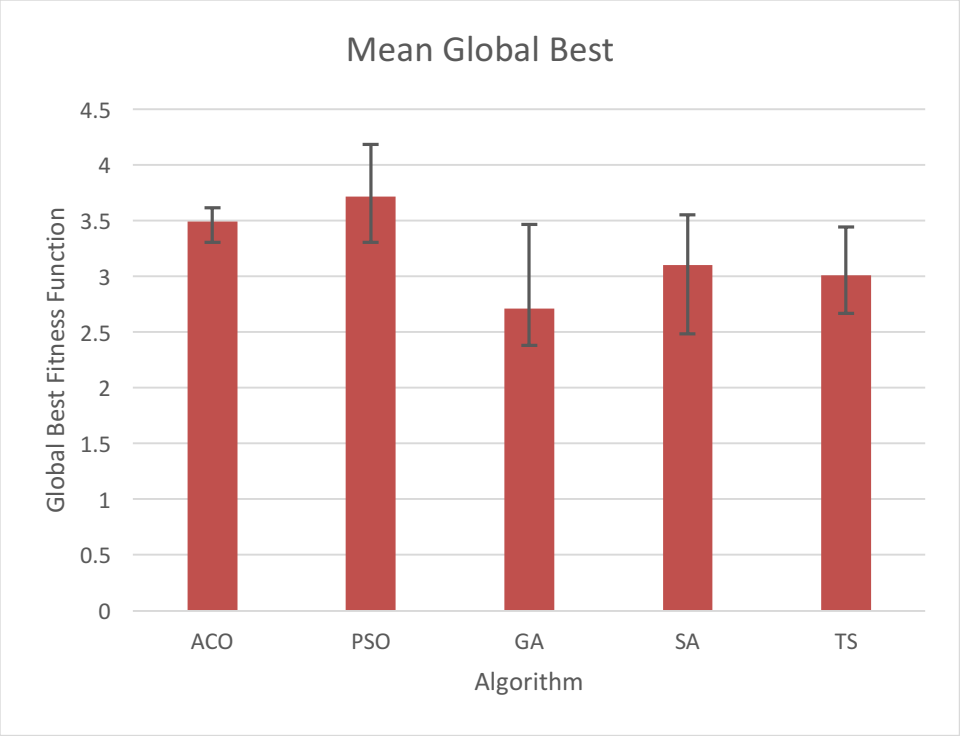

Figure 7: Average Global Best Fitness of each Algorithm.

From Figure 7, it is evident that genetic algorithms yield the most suboptimal results with respect to the other algorithms. This may be a result of the elitism operator used in the algorithm. Elitism reduces diversity; therefore it is likely that the algorithm is locked in a local optima. On the other

hand, the swarm intelligent algorithms provide the best global fitness function. PSO provides the highest global fitness because it uses intelligence to communicate with other particles; therefore it is an informed search as opposed to SA and TS. Moreover, PSO is a cooperative algorithm, therefore there are multiple swarms running in parallel and communicating. Hence, the results are able to identify the global optimum. Likewise, ASO algorithm is able to retrieve a high global fitness. The ASO algorithm achieves a high global fitness value because it utilizes a large array of ants in multiple colonies, which result in a variety of optimal solutions. The best solution from the set of generated solutions contributes to the global pheromone matrix, which is then used to improve the overall search to find the global optimal solution.

6.2.2 Stability (standard deviation)

The standard deviation of each algorithm measures the variance in the solutions. This indicates whether the algorithm is trapped in a local optimum. Figure 8 displays the standard deviation.

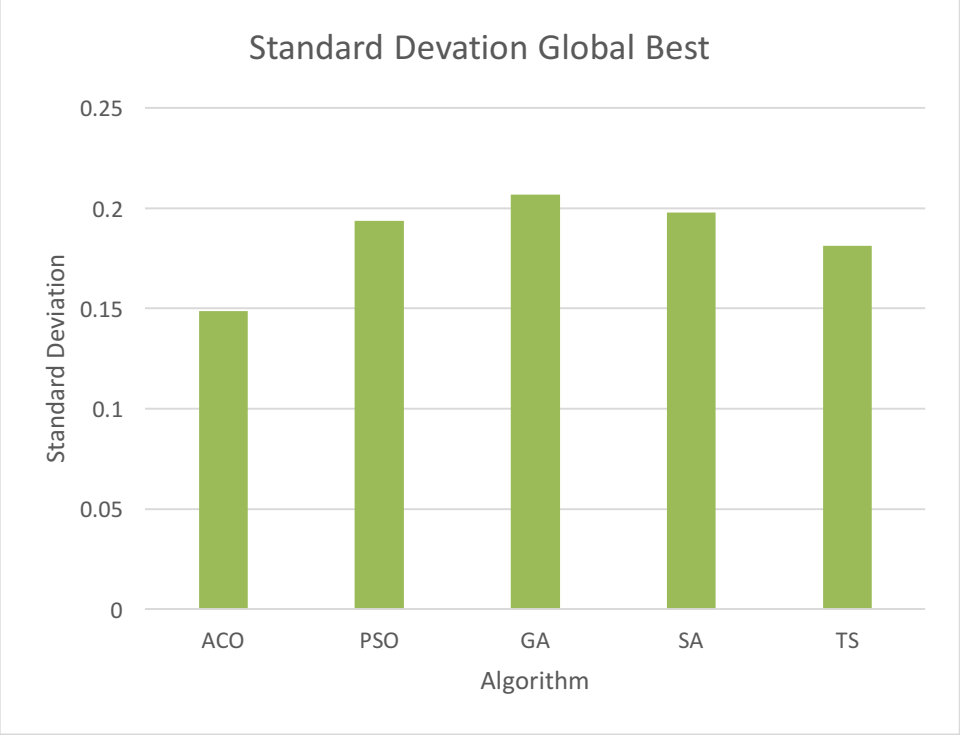

Figure 8: Standard Deviation of Best Solutions found.

Note: All algorithms have been run the same number of iterations to eliminate any bias

From Figure 8, it is evident that ACO holds the least variance in its solutions. This indicates that the solutions converge to an optimum the fastest among all the algorithms. In general, all the algorithms perform relatively the same with respect to their generated solutions. Therefore, all the algorithms hold relatively the same variance in their respective generated solutions.

6.3 Algorithm Performance (mean runtime)

Another important evaluation criteria when analyzing the algorithms is performance. To keep the comparison fair, the algorithms were all run on a Dell XPS 13 with an i7(2.60 GHz) processor and 8 GB RAM. Figure 9 below displays the average run time for the algorithms as well as highlights the maximum and minimum runtime values.

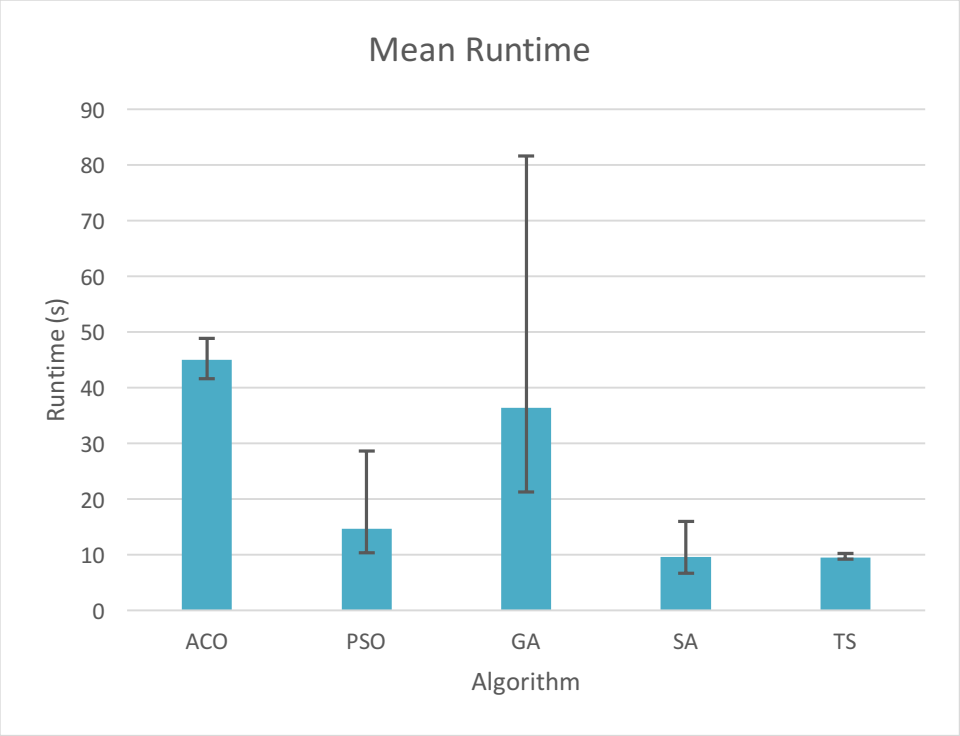

Figure 9: Average Runtimes of Metaheuristic Algorithms

As illustrated in Figure 9, tabu search performs the best in terms of average runtime. Furthermore, the maximum and minimum values of tabu search are close to the average and there is not much variation. Simulated annealing and particle swarm optimization are the next best performing algorithm. Genetic algorithm is not the slowest algorithm in terms of average runtime but can take a very large range of runtime values for different run iterations. Lastly, ant colony optimization takes the longest average runtime for the problem at hand.

7 Conclusions and Recommendations

This study analyzed five metaheuristic algorithms (Tabu Search, Simulated Annealing, Genetic Algorithm, Particle Swarm Optimization, and Ant Colony Optimization) and their ability to create the optimum itinerary from a given data set.

From the results generated, it was found that the Particle Swarm Optimization yielded the best solution whilst having great performance relative to the others. The cooperative nature of the algorithm allowed it to achieve solutions, which were very close to the optimal value. The characteristic of PSO that makes it stand out is that particles and swarms are able to exchange information with each other and hence guide one another to the optimal solution. Furthermore, the randomness introduced in binary particle swarm optimization ensures that various solutions are tried and the best solutions shared amongst everyone. Although PSO was relatively slower than some of the other algorithms, the increased accuracy it provides makes it the suitable option to solve our combinatorial problem.

The remaining algorithms still produced near-optimal solutions at very respectable running times, albeit not on the same level as PSO. The two trajectory-based algorithms in particular, TS and SA, had the best performance amongst all of them. In contrast, the two population-based algorithms, ACO and PSO, had the highest mean global fitness values. GA, being the sole evolutionary-based algorithm, yielded the least optimal results for this problem.

In conclusion, the metaheuristic algorithm that would produce the most optimal results for the itinerary optimization problem with large data sets is the Particle Swarm Optimization. This result can be extended to resolve problems related to resource allocation, amongst others.

8 References

[1] F. Glover, "Tabu Search - Part I," *OSRA Jornal on Computing*, vol. 1, no. 3, pp. 190-206, Summer 1989.

[2] M. Dell'Amico and M. Trubian, "Applying Tabu Search to the Job Shop Scheduling Problem", *Annals of Operation Research*, 1993, vol. 41.

[3] Salman Hooshmand, Mehdi Behshameh, and Omid Hamidi, "A Tabu Search Algorithm With Efficient Diversification Strategy For High School Timetabling Problem," *International Journal of Computer Science & Information Technology*, vol. 5, no. 4, pp. 21-34, August 2013.

[4] C. H. Aladag and G. Hocaoglu, "A Tabu Search Algorithm to Solve a Course Timetabling Problem," *Hacettepe Journal of Mathematics and Statistics*, vol. 36, no. 1, pp. 53-64, 2007.

[5] F.T. Lin, "Applying the genetic approach to simulated annealing in solving some NP-hard problems," *Systems, Man and Cybernetics*, vol. 23, no. 6, pp. 1752-1767, 2002.

[6] M. Golub I. Martinjak, "Comparison of Heuristic Algorithms in Functions Optimization and Knapsack Problem," Department of Electrical Engineering and Computing, University of Zagreb, Zagreb, 2006.

[7] R.Singh. "Solving the 0–1 Knapsack problem using Genetic Algorithm," *in proceedings at IEEE 3rd International Conference on Communication Software and Networks*, 2011, pp.1120-1125.

[8] I.Martinjak and M.Golub. "Comparison of Heuristic Algorithms and Optimization Functions and Knapsack Problem," in proceedings at 17th International Conference on Information and Intelligent Systems, 2006, pp.413-418.

[9] S.Khuri, T. Bäck and J. Heitkötte. "The zero/one multiple knapsack problem and genetic algorithms." 1994 ACM symposium on Applied computing. [online], pp. 188-193.

[10] M. Hristakeva and D.Shrestha. "Solving the 0-1 Knapsack Problem with Genetic Algorithms."Internet:https://www.assembla.com/spaces/ia2008/documents/aQOG04rHKr3Bdjab7jnrAJ/download/aQOG04rHKr3Bdjab7jnrAJ[August 1, 2015].

[11] Fangguo He. "An Improved Particle Swarm Optimization for Knapsack Problem" in Computational Intelligence and Software Engineering, *CiSE*, Wuhanan, Hubei, China 2009, pp 1-4.

[12] F. Hembecker, H.S. Lopes, and W. Goody Jr., "Particle Swarm Optimization for the Multidimensional Knapsack Problem" *Adaptive and Natural Computing Algorithms, Lecture Notes in Computer Science*, vol. 4431, 2007, pp. 358-365.

[13] Y. Liang, L. Liu, D Wang, and R. Wu, "Optimizing Particle Swarm Optimization to Solve Knapsack Problem" *Information Computing and Applications, Communications in Computer and Information Science*, vol. 105, 2010, pp. 437-443.

[14] M. Kong, and P Tian, "Apply the Particle Swarm Optimization to the Multidimensional Knapsack Problem" Artificial Intelligence and Soft Computing – ICAISC 2006, Lecture Notes in Computer Science, vol. 4029, 2006, pp. 1140-1149.

[15] J. Sun, S. Xiong, F. Guo, "A new pheromone updating strategy in ant colony optimization," Machine Learning and Cybernetics, 2004. Proceedings of 2004 International Conference, vol.1, no., pp.620, 625 vol.1, 26-29 Aug. 2004.

[16] R.G. Tiwari, M. Husain, S. Gupta, and A. P. Srivastava. "A new ant colony optimization meta-heuristic algorithm to tackle large optimization problem". *In Proceedings of the Third Annual ACM Bangalore Conference*. ACM, New York, NY, USA. Article 28, 4 pages, 2010.

[17] G. Liu, J. Xiong. "Ant Colony Algorithm Based on Dynamic Adaptive Pheromone Updating and Its Simulation," *Computational Intelligence and Design (ISCID), 2013 Sixth International Symposium on*, vol.1, no., pp.220,223, 2013.
[18] J. Kennedy and R. C. Eberhart. “A Discrete Binary Version of The Particle Swarm Algorithm”. *Proceedings of the World Multiconference of Systemics, Cybernetics and Informatics*, pp. 4101-4109, 1997.