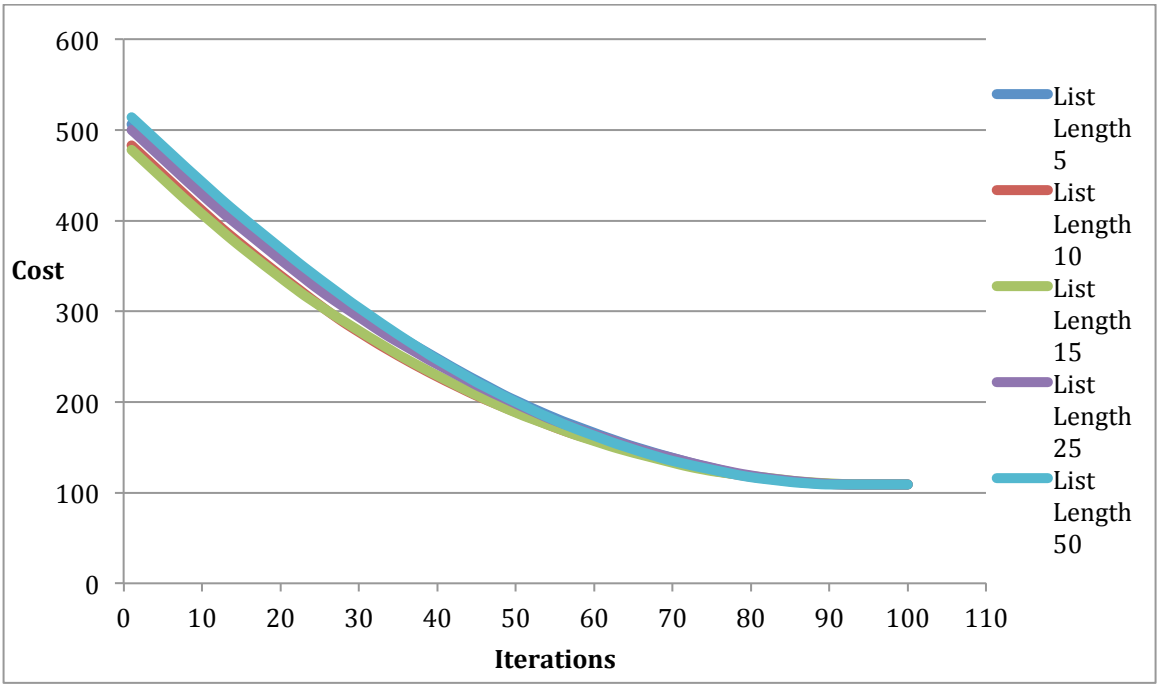
Assignment 2

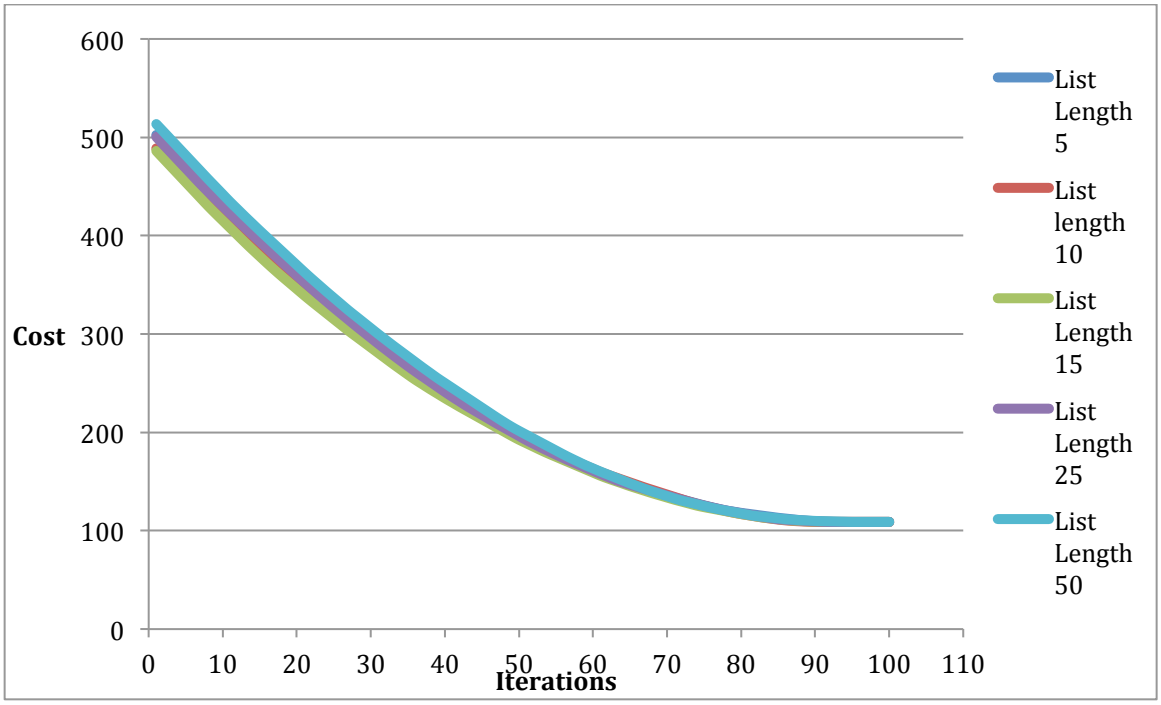
1) a) Using Kruskal and Tabu Search algorithms, the most cost effective path for the graph given in Units100.mat has cost of 109.

b) We ran the algorithm 5 times for 100 iterations for each tabu list length. It was observed that regardless of the Tabu length all answers converged to the optimal solution of 109 after around 90 iterations. Below is a graph of the data obtained:

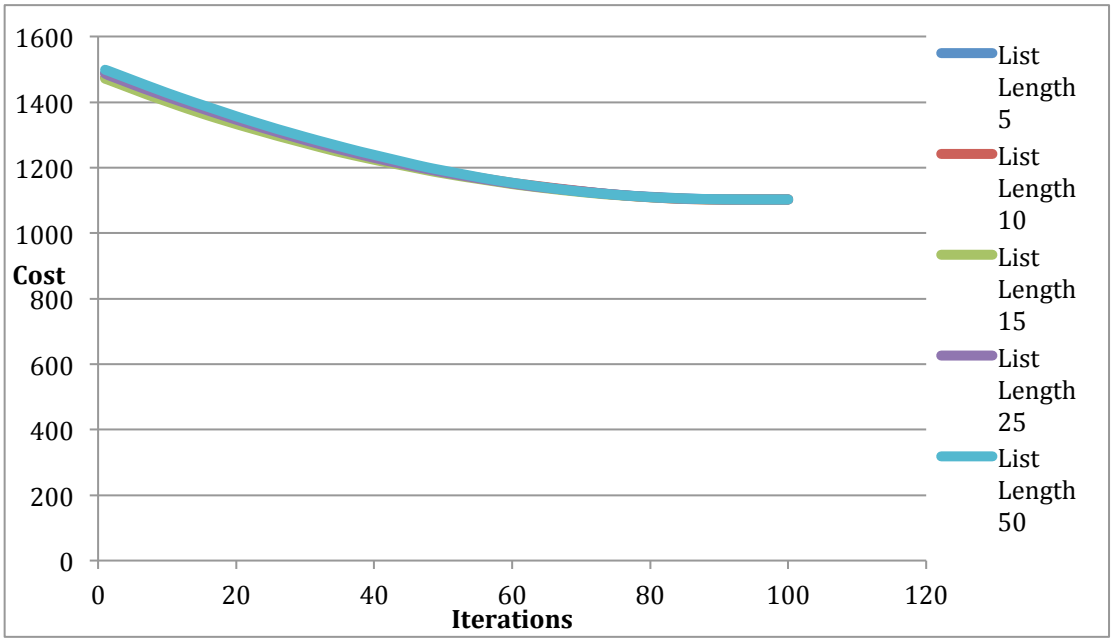


c) Modified GetBestNeighbourSTPartC.m included.

After adding the aspiration criteria, the optimal solution for the different list lengths remains the same as before, except the fact that the tabu search algorithm converges to the optimal solution of 109 in less iterations, meaning the optimal solution is found more quickly. Below is a graph of the data obtained.



d) Modified GetBestNeighbourSTPartD.m included.



After adding the cost of the splitters, when using the Tabu search algorithm the optimal solution converges to 1100 for all the tabu list lengths. Whereas, Kruskal is unable to produce a solution.

e) Tabu Search algorithm produces the same answer as Kruskal for the original problem. For the original problem, Kruskal greedy algorithm produces optimal solution in O(n) time. Whereas, Tabu search produces the optimal solution in many iterations.

When the cost of splitters is added to the problem, Kruskal is no longer feasible and unable to produce a solution. After adding the cost of the splitters and making the problem complex, Tabu search algorithm is still able to produce an optimal solution. Therefore, greedy algorithm is more feasible for the original problem, but when the problem becomes complex tabu search is feasible.

2) i)

In Figure 1, the graph shows the cooling curve of an exponential multiplicative cooling function and a linear multiplicative cooling function. For this graph the initial temperature is initialized to 1, the number of iterations is initialized to 100 and alpha is set to 0.9. The example code given to us was using exponential multiplicative cooling whereas the modified code was using linear multiplicative cooling.

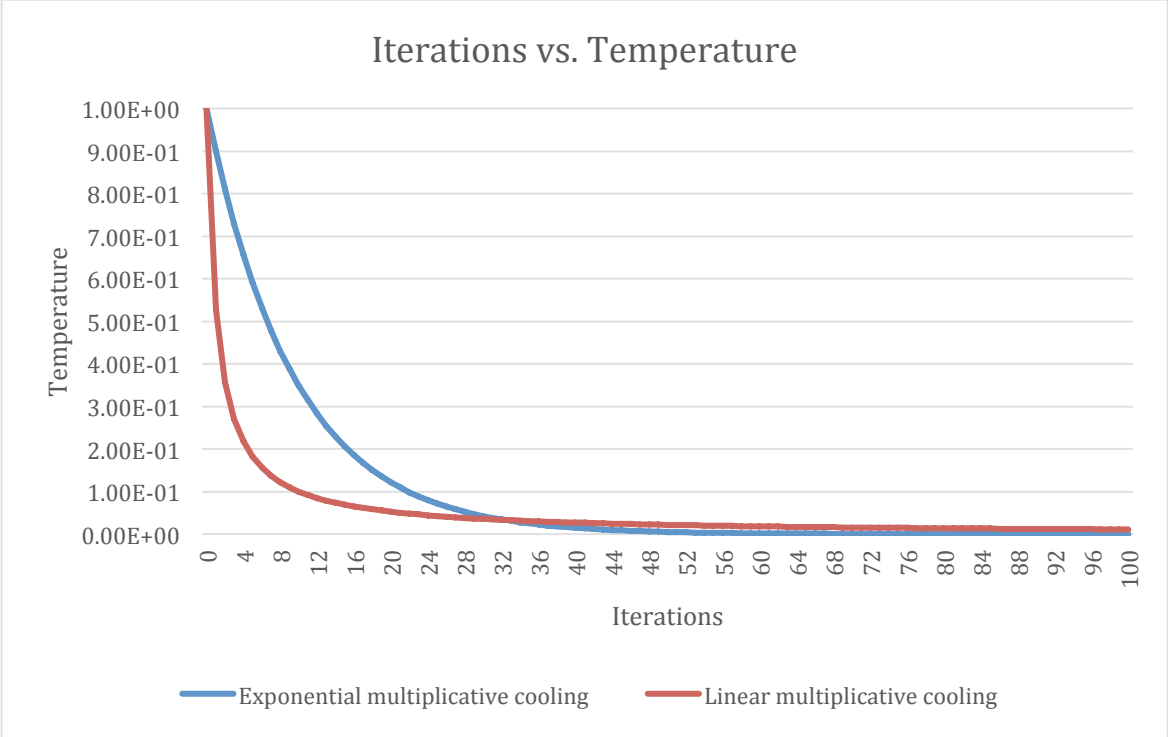


Figure 1: Multiplicative cooling curves.

For linear multiplicative cooling, alpha should be chosen so that as T(t) approaches 0, the algorithm should get closer to the global minimum. As shown in the graph, if we keep the alpha the same as the example code, the temperature rapidly drops for linear multiplicative cooling compared to exponential multiplicative cooling. For both cases we can gain a greater accuracy by trading off runtime performance by increasing the number of iterations.

ii)
For multimodal optimization problem, we can use any cooling function as long as the cooling function converges to zero as the number of iterations increases to infinite. It actually helps to use a non-monotonic function because the change in temperature in the cosine raises the temperature and the algorithm is less likely for it to be stuck in local optima.

3)

Part A

- **Solution Representation**

  Simulated Annealing (SA) can be used to solve this problem by having the following vector solution representation:

  $$Sol = [d, R_1, d, R_2, \dots, d, R_m]$$

  Where R_i denotes the list of customers/cities serviced by vehicle ***i*** and these lists are separated by d denoting the depot.

- **Neighborhood**

  The neighborhood can be defined by:
  - **Swapping** two randomly selected cities in any two randomly selected routes, the cities can be in either the same route or different routes.
  - **Removing** a randomly selected city from one randomly selected route and **inserting** it into a randomly selected position in a different randomly selected route.

- **Objective Function**

  Depot is the vector that contains the distances of each city from the depot.
  D is the matrix that contains the distances between city i and city j. Assume ***symmetric problem***
  Let ***S*** be the matrix that contains the time required to service each city.

  Then the cost of a route R_i is given by (n is the number of cities in a route):

  $$Cost(R_i) = \left(Depot(R_{i,1}) + Depot(R_{i,n}) + \sum_{j=1}^{n-1} D(R_{i,j}, R_{i,j+1}) \right) + \sum_{j=1}^{n-1} S(R_{i,j})$$

  and the total cost of solution (the objective function) is given by (m is the number of vehicles):

  $$Cost(sol) = \sum_{i=1}^{m} Cost(R_i)$$

Part B

The constraint that the total duration of any route should not exceed a preset bound ***T*** introduced a new dynamic to the problem formulation. The **Neighborhood** is affected in such a case. We now have to look at the sum of the service time for each route and **swap** or **remove and add** cities to the different routes in such a manner that the $\sum_{j=1}^{n-1} S(R_{i,j})$ > T. If we are using simulated annealing to solve the problem, addition of the constraint that the total duration of any route should not exceed a preset bound ***T*** can be accounted for by penalizing each solution that contains a route longer that T. A cost factor can be multiplied so that the cost grows exponentially.

Let ***P*** be this new cost defined by

$$\begin{array}{ll} P = T & if \sum_{j=1}^{n-1} S(R_{i,j}) > T \\ 0 & otherwise \end{array}$$

The new objective function is:

$$Cost(R_i) = \left[\left(Depot(R_{i,1}) + Depot(R_{i,n}) + \sum_{j=1}^{n-1} D(R_{i,j}, R_{i,j+1})\right) + \sum_{j=1}^{n-1} S(R_{i,j})\right] \times P$$