ECE457A Assignment 2

Group 27

Lucas Wojciechowksi, Ariel Weingarten, Alexander Maguire,
Austin Dobrik, Dane Carr

May 22, 2014

1 Question 1

1.1 Part A

1.1.1 Solution Representation

Solutions will be represented as a set of m vectors of varying length l_i s.t. $\sum_{i=0}^{m} dim(v_i) = c$. Each vector represents the route for one vehicle, and each element represents a city along the vehicle's route with cities being visited in order.

$$v_0 = \begin{bmatrix} v_{0,0} & v_{0,1} & \cdots & v_{0,l_0} \end{bmatrix}$$
$$v_1 = \begin{bmatrix} v_{1,0} & v_{1,1} & \cdots & v_{1,l_1} \end{bmatrix}$$
$$\vdots$$
$$v_m = \begin{bmatrix} v_{m,0} & v_{m,1} & \cdots & v_{m,l_m} \end{bmatrix}$$

1.1.2 Neighbourhood Operators

There will be two neighbourhood operators, one to change the order of the cities and one to change the length of a route. The first operation is to swap any two elements. The elements can either be in the same vehicle vector, or in two different vehicle vectors. The second operation is to remove one element from a vehicle vector and add it to another.

1.1.3 Objective Function

Let $Depot_{c_i}$ be the distance between the depot and city i
Let D_{c_i,c_j} be the distance between city i and city j
Let S_{c_i} be the time required to service city i

$$\sum_{i=0}^{m}(Depot_{v_{i,0}} + Depot_{v_{i,l_i}} + \sum_{j=0}^{l_i-1} D_{v_{i,j},v_{i,j+1}} + \sum_{j=0}^{l_i} S_{v_{i,j}})$$

1.2 Part B

To account for a constraint on the length of each route, we could penalize each solution that contains a route longer than T. Every route that exceeds T would add an additional cost P to the objective function. This way the SA algorithm would be able to probabilistically choose this poor solution if it needed to diversify. The new objective function would be

$$\sum_{i=0}^{m}(Depot_{v_{i,0}} + Depot_{v_{i,l_i}} + \sum_{j=0}^{l_i-1} D_{v_{i,j},v_{i,j+1}} + \sum_{j=0}^{l_i} S_{v_{i,j}} + P|_{if cost(v_i)>T})$$

2 Question 2

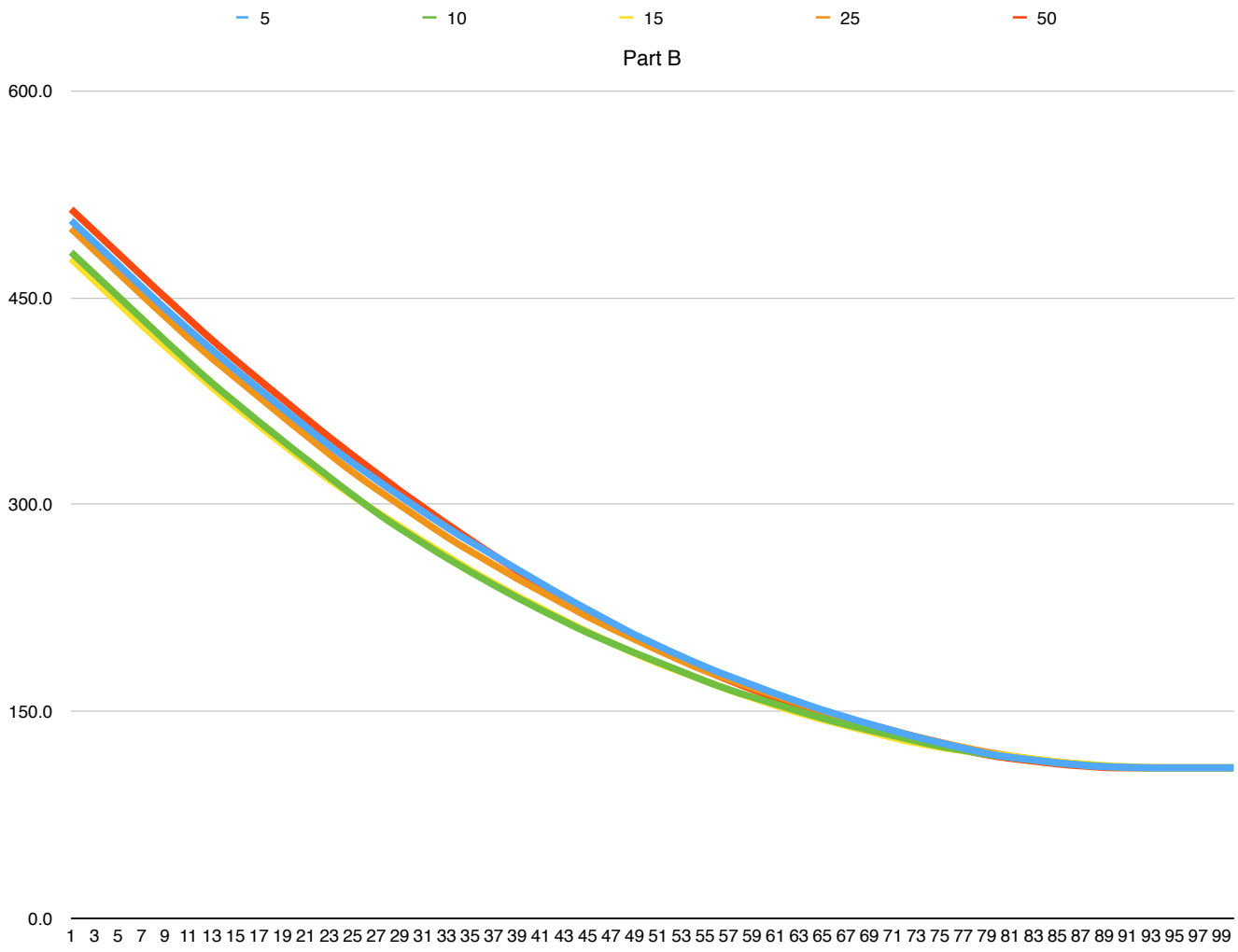
2.1 Part A

Using Kruskal's Algorithm, we calculate the most cost-effective paths for the graph given in `Units100.mat` and find the cost of this solution to be 109.

2.2 Part B

Since the number of iterations was not specified, we have included a graph of the solution provided by the algorithm for iterations 1 through 100, averaged over 5 runs.

We observe that regardless of tabu length, all answers converge to the ideal answer of 109 around 90 iterations.


5
10
15
25
50
Part B
600.0
450.0
300.0
150.0
0.0
1 3 5 7 9 11 13 15 17 19 21 23 25 27 29 31 33 35 37 39 41 43 45 47 49 51 53 55 57 59 61 63 65 67 69 71 73 75 77 79 81 83 85 87 89 91 93 95 97 99

2.3 Part C

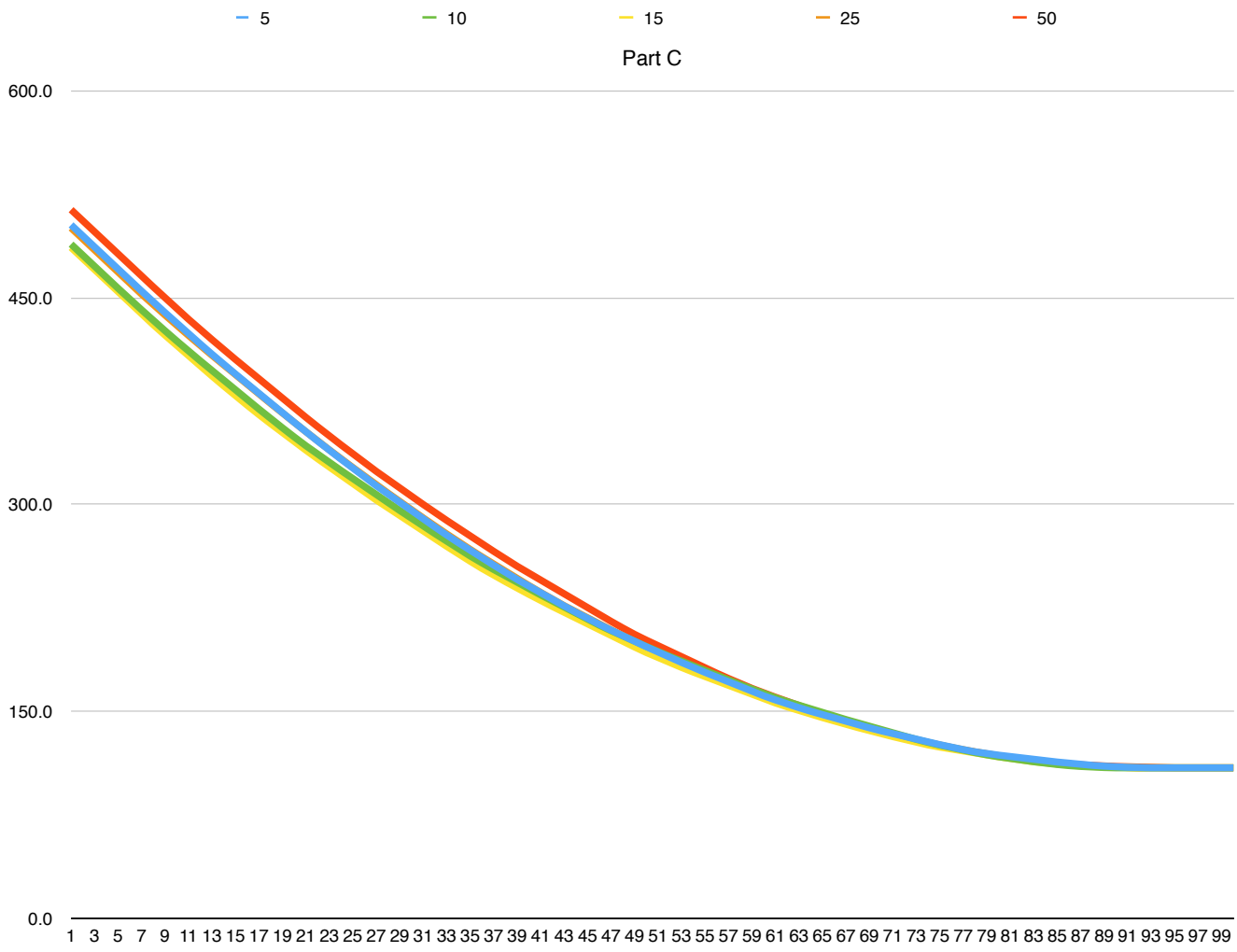
We modified `GetBestNeighbourST.m`'s use of the tabu list to implement an aspiration criterion. In particular, we changed

```
% 2. Check if the edge to be removed (e1) is in the tabu list
if TabuEdges(NT1(e1), NT2(e1)) == 0
    if NewSTCost < BestNeighbourSTCost
        BestNeighbourST = NewST;
        BestNeighbourSTCost = NewSTCost;

        TabuEdgeN1 = N1(BestEdge);
        TabuEdgeN2 = N2(BestEdge);
        AddTabuMove = true;
    end
end
```

to read

```
% 2. Check if the edge to be removed (e1) is in the tabu list
if (TabuEdges(NT1(e1), NT2(e1)) == 0 && NewSTCost < BestNeighbourSTCost) || ...
            (TabuEdges(NT1(e1), NT2(e1)) ~= 0 && NewSTCost < STCost)

        BestNeighbourST = NewST;
        BestNeighbourSTCost = NewSTCost;

        TabuEdgeN1 = N1(BestEdge);
        TabuEdgeN2 = N2(BestEdge);
        AddTabuMove = true;
    end
end
```

We graphed this code in the same way as part B. Note that the graph is shallower as all runs converge to the ideal answer more quickly than in part A.


5
10
15
25
50
Part C
600.0
450.0
300.0
150.0
0.0
1 3 5 7 9 11 13 15 17 19 21 23 25 27 29 31 33 35 37 39 41 43 45 47 49 51 53 55 57 59 61 63 65 67 69 71 73 75 77 79 81 83 85 87 89 91 93 95 97 99

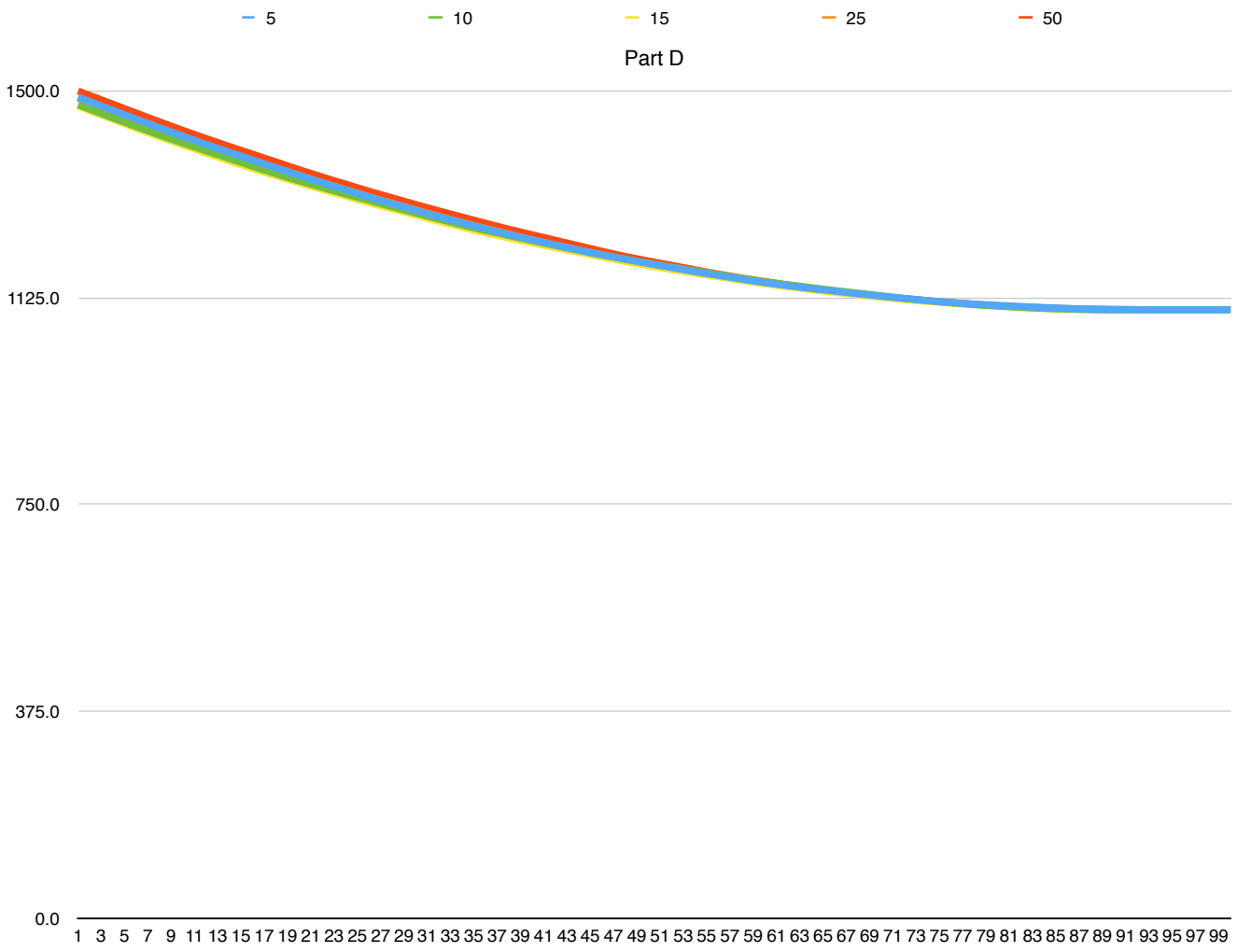
2.4 Part D

In this part, we replaced all cost calculations with a call to the function

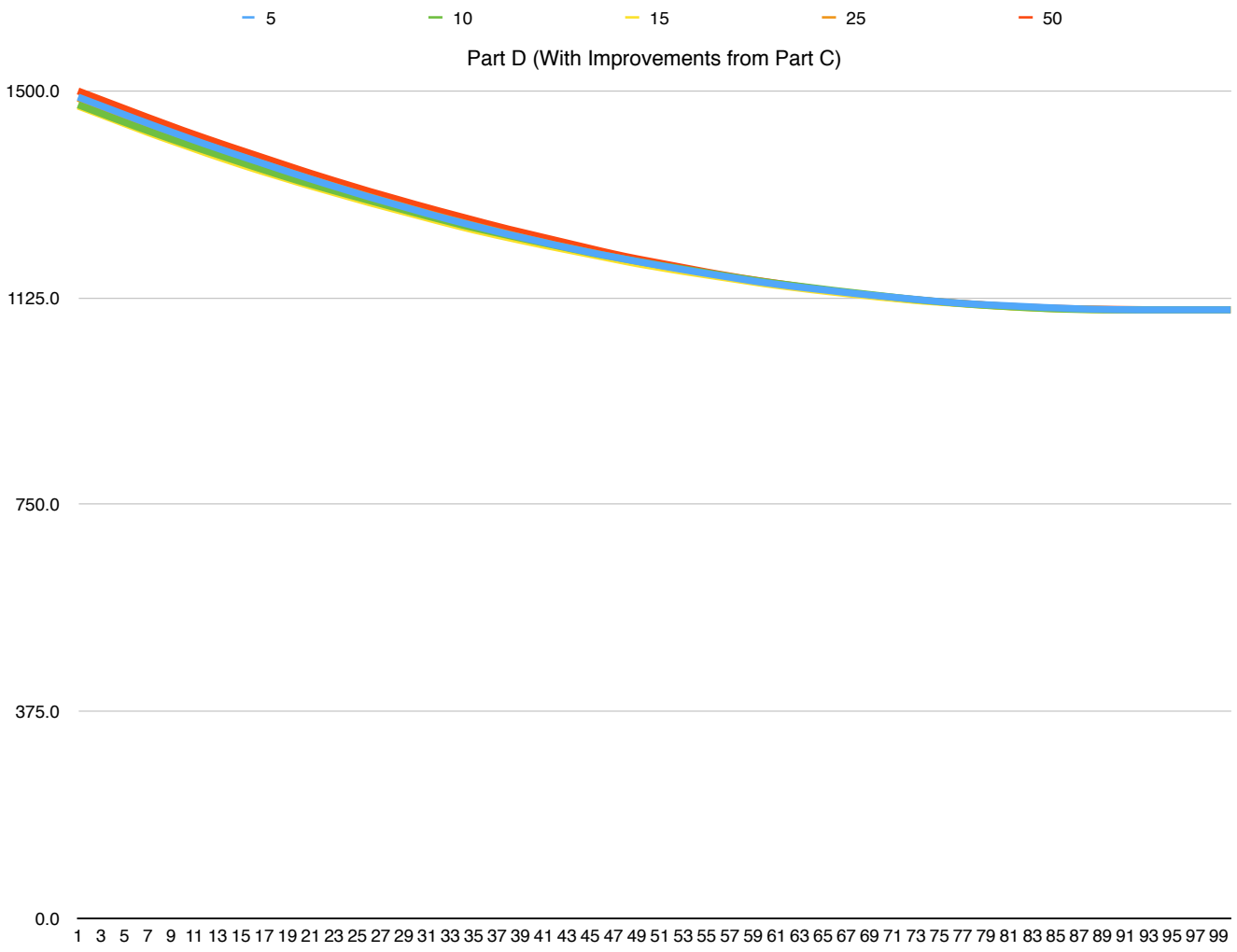
```
function Cost = GetCost(Graph, ST)

EdgeCost = sum(sum(Graph .* ST)) / 2;
NodeCost = sum(10 ./ (1 + exp(-1 * sum(Graph) ./ 10)));

Cost = EdgeCost + NodeCost;
```

This new cost function did not produce an ideal solution with Kruskal's algorithm.

Using Tabu Search, however, it did converge to an answer at a cost of 1103. We graphed the solution versus iterations averaged over 5 runs in the same way as part A.


5
10
15
25
50
Part D
1500.0
1125.0
750.0
375.0
0.0
1 3 5 7 9 11 13 15 17 19 21 23 25 27 29 31 33 35 37 39 41 43 45 47 49 51 53 55 57 59 61 63 65 67 69 71 73 75 77 79 81 83 85 87 89 91 93 95 97 99

We also tried using the improvements from part C in the new problem proposed for part D and found it to have better performance characteristics again. We graphed the solution versus iterations averaged over 5 runs in the same way as part A.


5
10
15
25
50
Part D (With Improvements from Part C)
1500.0
1125.0
750.0
375.0
0.0
1 3 5 7 9 11 13 15 17 19 21 23 25 27 29 31 33 35 37 39 41 43 45 47 49 51 53 55 57 59 61 63 65 67 69 71 73 75 77 79 81 83 85 87 89 91 93 95 97 99

2.5 Part E

In parts A and B, Kurskal's algorithm produced the same answer as Tabu Search in many orders of magnitude less time. However, in part D, the nature of the problem changed such that a greedy algorithm such as Kurskal's was no longer suitable.

In general, for problems like this, one should use a greedy algorithm if at all feasible; the greedy algorithm will produce a guaranteed optimal solution in $O(n)$ time. However, when a greedy algorithm is not feasible, such with the complex cost function in part D, a Tabu Search algorithm is a good tool to try.