Nick DeGroote
Optimal Control: Project No. 2
22 June 2021

Given the following normalized [x, y] locations of 50 cities (see Table in Appendix A and Figure below), develop the following using MATLAB:

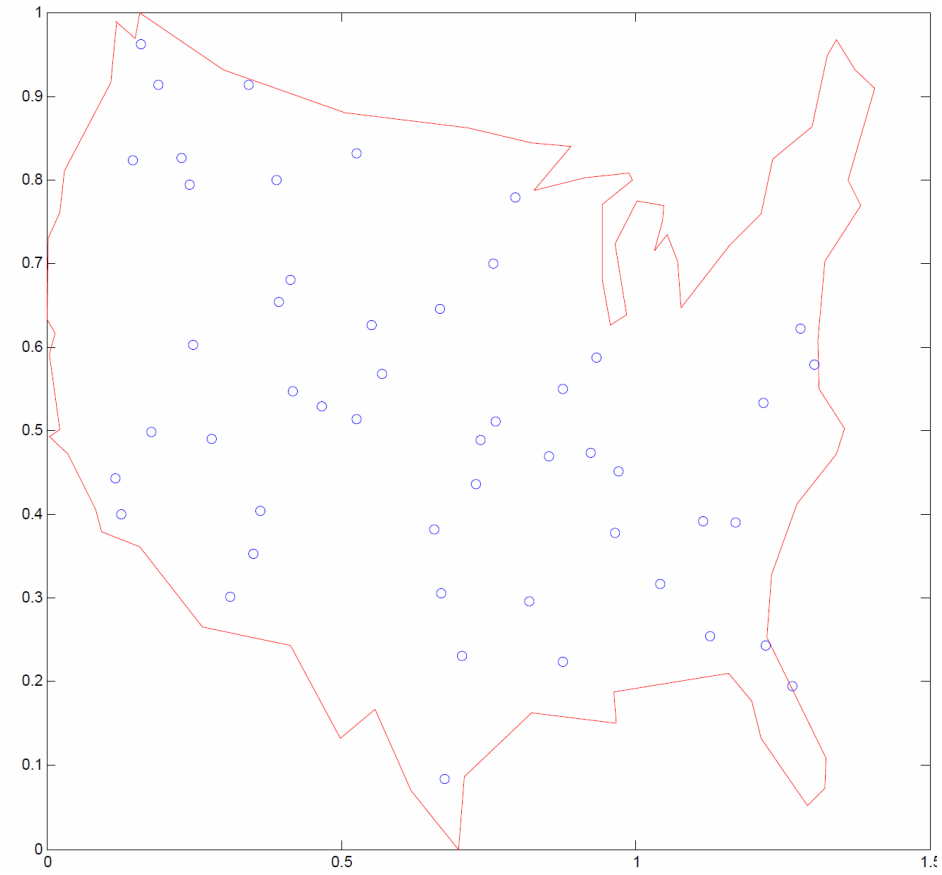

Figure 1: A map showing the locations of all 50 US cities in the dataset.

Note: The pseudo-code for the algorithms is given in a more typical algorithmic format. All benchmarks were performed on an Intel i5-8400 processor @2.8 GHz. All run times are in seconds. Performance metrics are shown directly below thier respective figures.

1.1.Upper bound tour using the nearest neighbor approach (5 Points)

Algorithm 1: Dynamic Programming Algorithm

Result: An ordered list of cities to visit
procedure $nearestNeighbor(cityList)$
$unvisitedCities \leftarrow cityList$
$route \leftarrow$ first city in $cityList$
$routeDistance \leftarrow 0$
$currentCity \leftarrow$ first city in $cityList$
while $unvisitedCities$ *is not empty* **do**
  $unvisitedCities \leftarrow unvisitedCities - currentCity$
  $closestCity \leftarrow$ closest unvisited city to the current city
  $route \leftarrow route \cup closestCity$
  $routeDistance \leftarrow routeDistance+$ distance to $closestCity$
  $currentCity \leftarrow closestCity$
end
return $route, routeDistance$

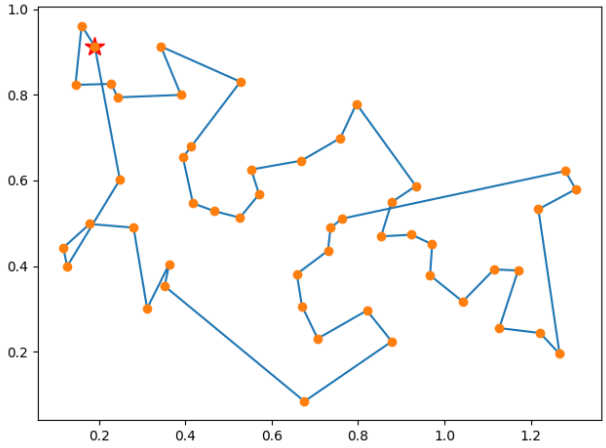

Figure 2: The convex hull of the cities in the dataset.

Nearest Neighbor Upper Bound: 6.201

Route: 0, 28, 3, 26, 20, 30, 25, 15, 8, 23, 1, 21, 33, 17, 40, 7, 12, 18, 46, 16, 19, 14, 38, 44, 4, 2, 35, 11, 13, 49, 45, 31, 32, 43, 34, 41, 5, 42, 48, 39, 10, 24, 37, 36, 47, 6, 9, 27, 29, 22

1.2.Lower bound tour using the convex hull (5 Points)

The convex hull is found by drawing the minimum-size convex polygon which inscribes all data in the set. The computation was performed using the scipy Python package. The exact algorithm for computing the convex hull is outside the scope of this project.

Convex Hull Lower Bound: 3.405

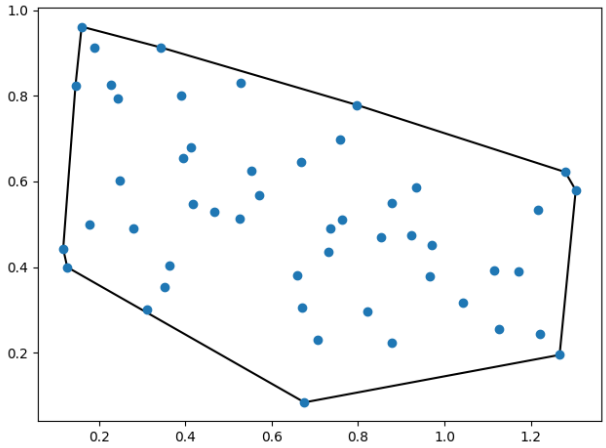

Figure 3: The convex hull of the cities in the dataset.

1.3. Use the GA toolbox to create the computationally fastest AND shortest tour (15 Points)

Algorithm 2: Genetic Algorithm (DeGroote)

Result: An ordered list of cities to visit
procedure $geneticAlgorithm(cityList)$
$population \leftarrow$ random array of ordered $cityLists$ of size $populationSize$
while $iteration \leq numIterations$ **do**
  for *each* $individual$ *in* $population$ **do**
    $individual.calculateFitnessValue$
  end
  $newPopulation \leftarrow N$ fittest individuals in this population
  while $newPopulation$ *length* $\leq populationSize$ **do**
    $parentOne \leftarrow tournamentSelection(population)$
    $parentTwo \leftarrow tournamentSelection(population)$
    if $randomNumber[0,1] \leq crossoverProbability$ **then**
      $childOne, childTwo \leftarrow crossover(parentOne, parentTwo)$
      $newPopulation \leftarrow newPopulation \cup \{childOne, childTwo\}$
    else
      $newPopulation \leftarrow newPopulation \cup \{parentOne, parentTwo\}$
    end
  end
  for *each* $individual$ *in* $newPopulation$ **do**
    if $randomNumber[0,1] \leq mutationProbability$ **then**
      $individual \leftarrow mutateChromosomes(individual)$
    else
      $newPopulation \leftarrow newPopulation \cup \{parentOne, parentTwo\}$
    end
  end
  $population \leftarrow newPopulation$
end
return $fittestIndividual$

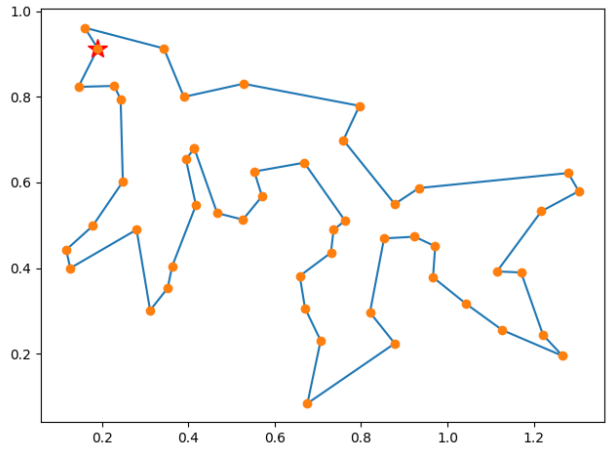

Figure 4: The shortest path found my my GA implementation.

Route Length: 5.709

Run Time: 3.978

Route: 3, 26, 20, 22, 6, 9, 27, 29, 47, 37, 36, 5, 42, 48, 24, 39, 10, 11, 13, 49, 31, 32, 45, 35, 2, 4, 44, 38, 14, 46, 16, 19, 34, 41, 43, 12, 18, 7, 40, 17, 33, 21, 1, 23, 8, 15, 30, 25, 28, 0

Parameter	Value
Crossover Probability	0.6
Mutation Probability	0.03
Elitism Ratio	0.02
Population Size	75
Generations	2500

Table 1: The stochastic parameters used in the custom GA implementation

1.4. Benchmark your algorithm using the following two provided algorithms: (5 Points)

- MATLAB's Central's 'tsp_ga.m'

Algorithm 3: Genetic Algorithm (MATLAB)

Result: An ordered list of cities to visit
procedure $geneticAlgorithm(numIterations, populationSize)$
$population \leftarrow$ random array of ordered cities of size $populationSize$
while $iteration \leq numIterations$ **do**
  for *each* $individual$ *in* $population$ **do**
    $individual.calculateFitnessValue$
  end
  $newPopulation \leftarrow$ empty array
  for *each* $individual$ *in the top 25% of fitness* **do**
    $childOne \leftarrow flipMutation(individual)$
    $childTwo \leftarrow swapMutation(individual)$
    $childThree \leftarrow slideMutation(individual)$
    $newPopulation \leftarrow$
     $newPopulation \cup \{individual, childOne, childTwo, childThree\}$
  end
  $population \leftarrow newPopulation$
end
return $fittestIndividual$

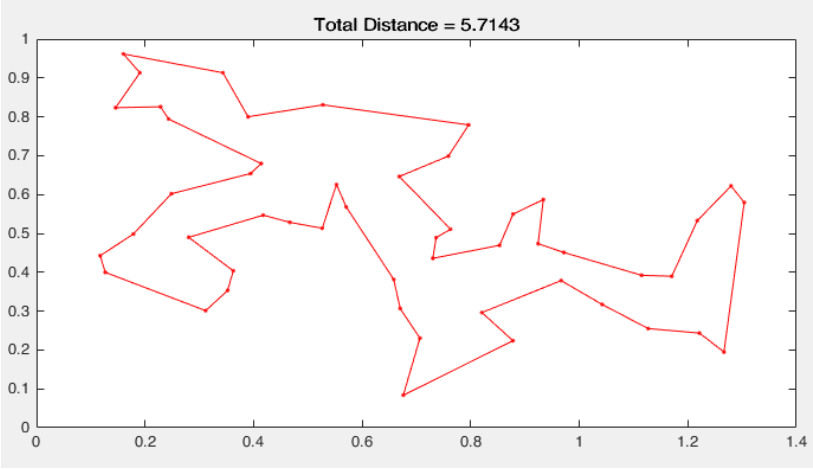

Figure 5: The shortest path with MATLAB's GA implementation.

Route Length: 5.7140

Run Time: 3.4907

- Anoop's '2-opt_TSP.m'

Algorithm 4: Two-opt Algorithm

Result: An ordered list of cities to visit
procedure $twoOpt(route)$
$existingRoute \leftarrow route$
$n \leftarrow$ number of cities $totalDistance \leftarrow 0$
for $iter = 1 : numIterations$ **do**
  for *city* $i = 1 : n - 2$ **do**
    for *city* $j = i + 2 : n$ **do**
      d_1 = distance from [i to $i + 1$] + distance from [j to $j + 1$]
      d_2 = distance from [i to j] + distance from [$i + 1$ to $j + 1$]
    end
    if $d_1 \geq d_2$ **then**
      $existingRoute \leftarrow existingRoute \cup$ swap cities at indices $i + 1$ and j
    end
  end
end
for *city* $j = 1 : numCities - 1$ **do**
  distance = distance + distance to city j
end
return $existingRoute, distance$

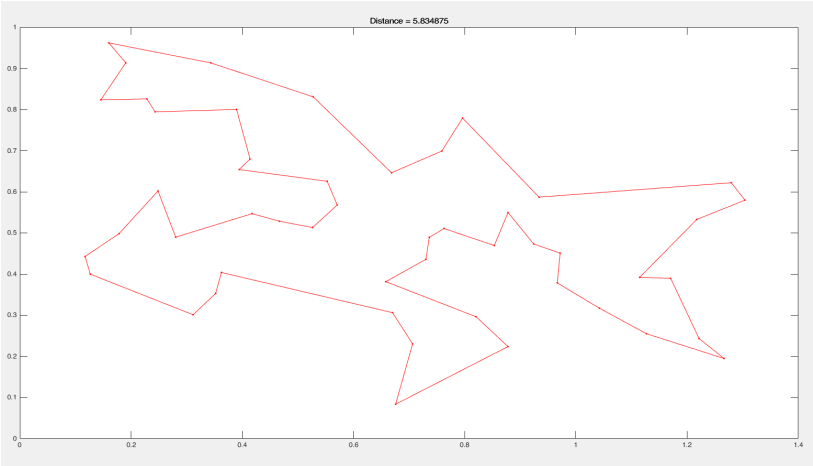

Figure 6: The shortest path found with Anoop's 2-opt algorithm.

Route Length: 5.8349

Run Time: 0.0096

Summary:

tsp_ga Distance	tsp_ga Time (s)	2opt Distance	2opt Time (s)	DeGroote Distance	DeGroote Time (s)
5.7140	3.4907	5.8349	0.0096	5.7094	3.9776

Table 2: A summary of the route lengths and run times for the algorithms tested in 1.4.

1.5. Scalability and Statistical analysis of your algorithm. Now develop 100 NEW and DIFFERENT cases for TSP using Matlab's random number generator for 50 cities each within the USA Map. Now compare your algorithm with tsp_ga.m and 2-opt_TSP.m for the above set-up (compare apples to apples) and with figures of merit including the tour distance and convergence time. (20 Points)

The data for each of the 100 runs is included with the project submission. The benchmark information for the SAME 100 randomly generated scenarios is shown in Table 3. The code for running the benchmark for the method developed in this assignment is included in benchmark.py.

Summary:

tsp_ga Average Distance	5.7865
tsp_ga Average Time	3.6631
2opt Average Distance	6.1345
2opt Average Time	0.012
DeGroote Average Distance	5.9400
DeGroote Average Time	4.0799

Table 3: A 100-run average comparing the solutions to the TSP using the TSP toolbox approach, a 2-opt approach and DeGroote's approach.

2.1. A Cluster first approach using K-Means followed by YOUR BEST GA applied to each cluster having 2 UAVs at the depot. Benchmark using mtsp_ga.m (10 points).

Algorithm 5: Cluster-First Genetic Algorithm

Result: An ordered list of cities to visit
procedure $clusterFirstGA(cityList, numClusters, depot)$
$routes \leftarrow$ empty array
for *each city in* $cityList$ **do**
  group city with others in its cluster
end
for *each* $cluster$ **do**
  $cluster \leftarrow cluster \cup depot$
  $routes[cluster] \leftarrow geneticAlgorithm(cluster)$
end
return $routes$

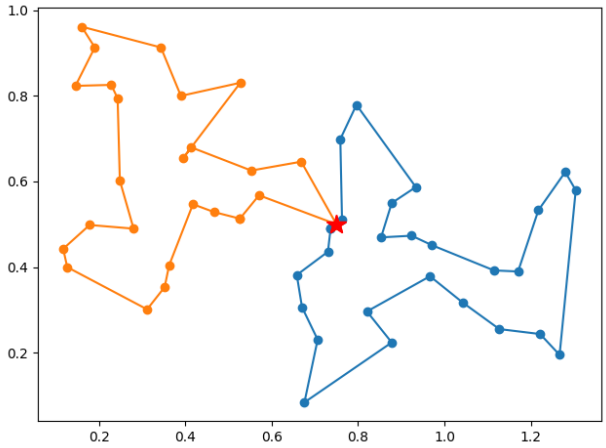

Figure 7: The solution found for the cluster-first MTSP using two salesman.

Max Distance: 2.790

Run Time: 1.141

Algorithm 6: Multiple TSP MinMax GA (MATLAB)

Result: An ordered list of cities to visit
procedure $geneticAlgorithm(cityList, numSalesmen, depot)$
$population \leftarrow$ random array of ordered cities of size $populationSize$
$breaks \leftarrow$ indices of breaks in each individual, length of $numSalesmen$
while $iteration \leq numIterations$ **do**
  for *each* $individual$ *in* $population$ **do**
    $individual.calculateFitnessValue$ (maximum of all tour lengths, starting at $depot$)
  end
  $newPopulation \leftarrow$ empty array
  $newBreaks \leftarrow$ empty array
  for *each* $individual$ *in the top* $1/8$ *of fitness* **do**
    (Mutate both the route and the breaks in the route)
    $routeChildOne \leftarrow flipMutation(individual)$
    $routeChildTwo \leftarrow swapMutation(individual)$
    $routeChildThree \leftarrow slideMutation(individual)$
    $breakChildOne \leftarrow modifyBreaks(individual)$
    $breakChildTwo \leftarrow flipModifyBreaks(individual)$
    $breakChildThree \leftarrow swapModifyBreaks(individual)$
    $breakChildFour \leftarrow slideModifyBreaks(individual)$
    $newPopulation \leftarrow newPopulation \cup$
    $\{individual, routeChildOne, routeChildTwo, routeChildThree$
    $newBreaks \leftarrow newBreaks \cup$
    $\{individual, breakChildOne, breakChildTwo, breakChildThree, breakChildFour\}$
  end
  $population \leftarrow newPopulation$
  $breaks \leftarrow newBreaks$
end
return $fittestIndividual$, $breaks$

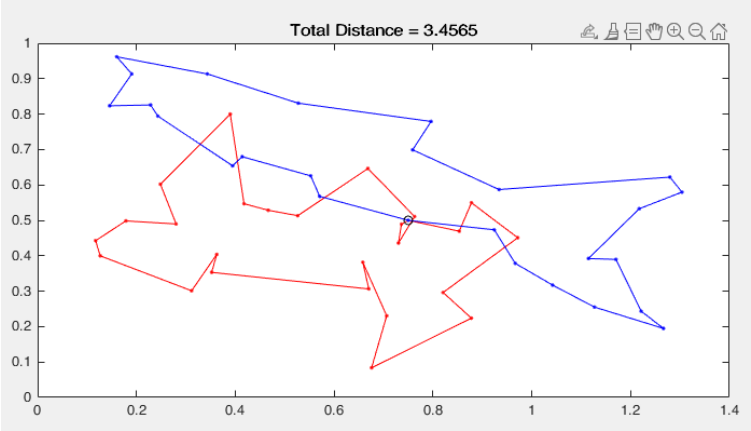

Figure 8: The solution found for the MATLAB mtsp function using two salesman.

Max Distance: 3.457

Run Time: 2.316

2.2. A Cluster first approach using K-Means followed by YOUR BEST GA applied to each cluster having 3 UAVs at the depot. Benchmark using mtsp_ga.m (10 points).

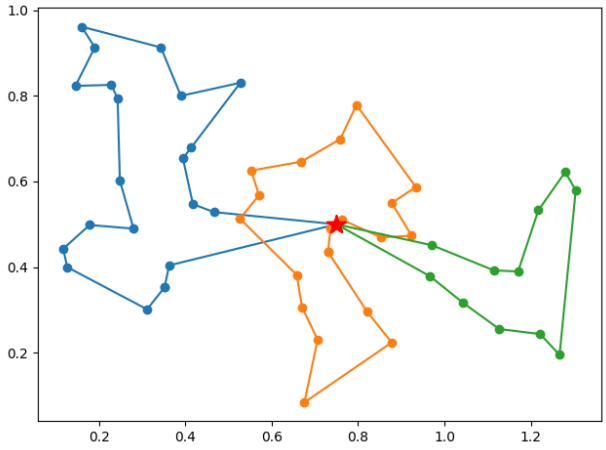

Figure 9: The solution found for the cluster-first MTSP using three salesman.

Max Distance: 2.661

Run Time: 1.776

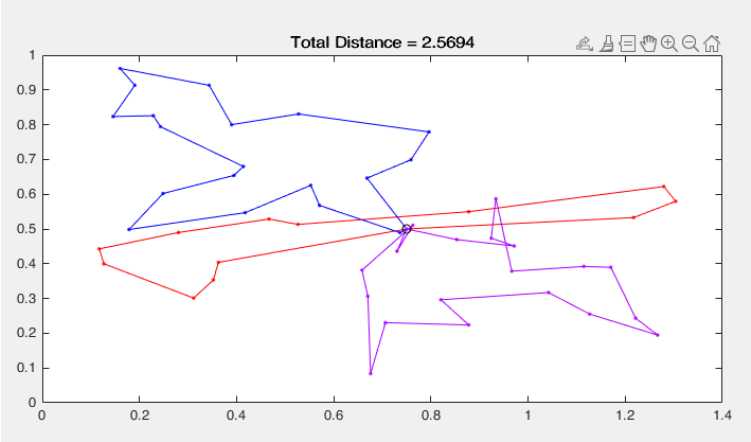

Figure 10: The solution found for the MATLAB mtsp function using three salesman.

Max Distance: 2.569

Run Time: 2.915

2.3. A Cluster first approach using K-Means followed by YOUR BEST GA applied to each cluster having 4 UAVs at the depot. Benchmark using mtsp_ga.m (10 points).

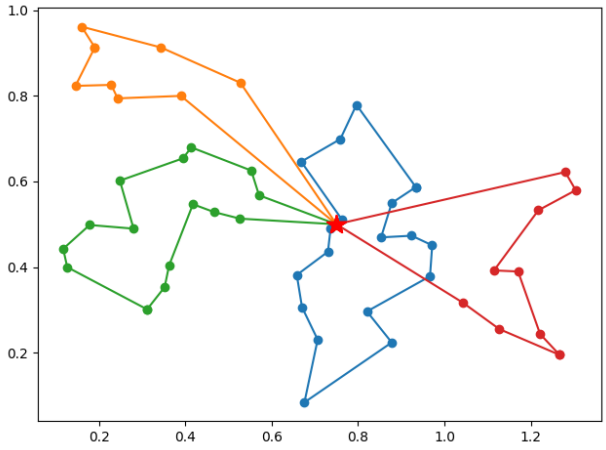

Figure 11: The solution found for the cluster-first MTSP using four salesman.

Max Distance: 1.936

Run Time: 2.264

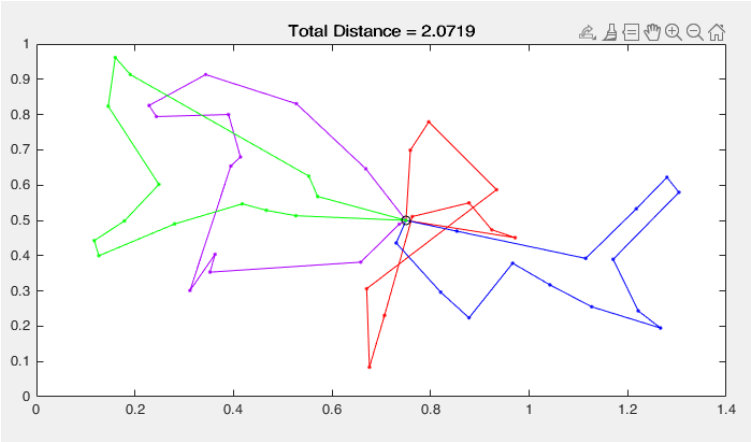

Figure 12: The solution found for the MATLAB mtsp function using four salesman.

Max Distance: 2.072

Run Time: 2.915

Summary:

Method	Number of Depots	MinMax Cost	Run Time (s)
DeGroote	2	2.790	1.141
MATLAB	2	3.457	2.316
DeGroote	3	2.661	1.776
MATLAB	3	2.569	2.915
DeGroote	4	1.936	2.264
MATLAB	4	2.072	2.915

Table 4: A summary of the MinMax costs and the run times for the techniques used in 2.3

Part 3 – Reflections and Lessons Learned Write a well-articulated, 2-3 page (font 12, 1.5 line spacing), perspective on the insights gained while developing this project. Clearly describe your observations, challenges, conclusions and lessons learned. What do you think of GA as an optimization tool? (20 Points)

This section is used to reflects and analyze some of the lessons learned from completing this project. We begin by analyzing the genetic algorithm (GA) developed from scratch and discussing its strengths and weaknesses. We then proceed to a comparison with the GA implementation provided by MATLAB, the the 2-opt algorithm provided by Anoop. Next, the results for the multiple TSP are discussed along with the lessons learned from this segment of the project. Finally, the capabilities of the GA as an optimization tool in general are reflected upon.

The GA used in this project was developed from scratch using Python. It designed with an emphasis on readability and abstracting the genetic operators away from the end user. The first lesson learned was that the GA is very sensitive to tuning the stochastic parameters for crossover probability, mutation probability, and elitism ratio. Setting the crossover probability too high resulted in the algorithm converging too quickly and becoming stuck in a local minimum. Conversely, setting the crossover percentage too low does not allow for adequate sharing of genetic information, resulting in very slow convergence. Setting the mutation rate too low results in the algorithm becoming easily stuck in local minima. Setting the mutation rate too high results in too much noise in the population to converge.

After setting these parameters appropriately, the GA produces acceptable results. The primary visual indicator is a tour which visits all cities and returns to its starting location without crossing over its path at any point. When comparing this GA to the one from MATLAB, we can see that the performances are similar. For the 100 tests run in this project, the MATLAB GA produces shorter tours by approximately 2.7% at a reduced computation time of approximately 11.4% on average. Even though the custom GA was outperformed, the results are still promising due to the limited development time. If more

time were spent on optimizing the algorithm, the average tour length and run time could likely be brought in line with the the MATLAB GA.

The GA solution was also compared to that found by the two-opt algorithm. The most striking difference between the GA and the two-opt is run time. While both the custom GA and the one provided by MATLAB typically execute in between 3 and 4 seconds for a 50 city TSP, the two-opt algorithm runs in approximately 0.01 seconds. While there is an increase in run-time performance of three orders of magnitude, the solution quality does not suffer significantly, increasing by only 5.97% over the 100-run average for the tests. For this scale of problem, the two-opt algorithm performs quite well, but for larger problem sizes it tends to become stuck in local minima. There is unfortunately not much that can be done to improve the solution at this point. With a GA however, the population size and number of generations can be increased in an attempt to find a better solution, but of course at the cost of increased run time. The choice between a two-opt and GA is therefore very dependent on the problem size, available time, and importance of solution quality.

The same genetic algorithm was also used to solve a single-depot multiple TSP. To solve this problem, both the cities that each salesman will visit, along with the ordering of the cities for each salesman need to be determined. Using k-means clustering provides a way to decouple the task assignment portion of this problem from the ordering of cities. The cities are first broken into several different clusters using the k-mean algorithm, then the TSP is solved for each cluster of cities. The function of the GA is identical for this problem, it simply runs once for each cluster.

The cluster-first solution to the multiple TSP is again compared to a solution provided by MATLAB. It is however worth noting that MATLAB's solution does not use a cluster-first approach, instead using an additional chromosome to mark breaks in the tour representing the route for each individual salesman. Each method has some distinct

advantages over the other. Using clustering first makes the problem significantly easier to solve by decreasing the number of possible solutions. Clustering completely decouples the task assignment from the TSP. On the other hand, the cluster first method could lead to sub-optimal solutions because the algorithm does not actually take into account the global MinMax objective function. K-means clustering does not necessarily lead to clusters which can be visited in a minimum amount of time and the GA is only able to minimize the total distance for each cluster. The MATLAB algorithm however, does take the global objective function into account. But as stated previously, this results in a much more difficult problem, leading to the GA struggling to find a solution.

In general, GAs are powerful global optimization tools that can be applied to a wide variety of difficult combinatorial optimization problems. They can be used effectively to find near-optimal solutions at a greatly reduced computation time when compared to exact methods. They are however, not a perfect solution in all cases. The quality of the solution is highly dependent on several stochastic parameters which need to be tuned for the specific problem. There are also algorithms which can produce near-optimal solutions much more quickly for larger problem sizes. Ultimately, the GA is very use-case dependent. It functions well in certain situations but its application needs to be analyzed for the particular use-case.