Description of Methods / Algorithms

This section will include a description of three methods to solve the traveling salesman problem (TSP), and pseudocode for each method.

Dynamic Programming (DP) Research

The Held-Karp DP algorithm to solve the TSP notes that every subpath of a solution with minimum distance is itself a path with minimum distance. This algorithm works in a bottom up fashion by starting with the smallest subpath and then incrementally constructing the shortest path from the start to the end node, finding the shortest path from node to node, and ensuring each node is visited exactly once.

However, the time complexity for this approach is $O(n^2 2^2)$, and is considered unfeasible for larger sets. Some sources also noted that the implementation of the algorithm includes some potential challenges, such as indexing each new subset. Other sources noted that approximation techniques, such as nearest neighbor, double spanning trees, and making use of probabilistic techniques, would be necessary as the number of cities increases.

DP Pseudocode

Let c be a vector of n cities, and distance(c[i], c[j]) be the distance between cities c[i] and c[j]. Let the first city c[1] be the start, and the c[end] be the end. Let path({S}, c[end]) be the shortest path for the set S, starting with c[1] and ending with c[end], and going to every city in S once. The pseudo-code for this DP algorithm is:

```
TSP(c, d, path):
   path({1} , 1) = 0
   for s = 2 to n
      for all subsets S, { 1, 2, 3, … n}, where the subset size is s and it contains 1
         path(S, 1) = infinity
         for j = 2 … S
            path(S, j) = min (path(S - {j}, i) + distance(i, j) for i != j and i an element of S

Return min path({1 … n}, j) + distance(j, i)
```

Nearest Neighbor Heuristic

Since the DP solution is not ideal for larger data sets, this section also explores the more practical nearest neighbor heuristic. The nearest neighbor heuristic is naive and greedy heuristic that it adds the city closest to the most recently added city. The first city is the starting point, and

then while there are still cities that have not been added, the next nearest city is added. This heuristic can lead a non-optimal final solution since it is a greedy algorithm.

Pseudocode

Let c be a vector of n cities, and distance(c[i], c[j]) be the distance between cities c[i] and c[j]. Let the first city c[1] be the start, and the c[end] be the end. Let path({S}, c[end]) be the shortest path for the set S, starting with c[1] and ending with c[end], and going to every city in S once. The pseudo-code for this DP algorithm is:

```
TSP(c, d, path):
Let the first city in the path be c[1]
While there are unvisited cities
   Add the next nearest city to the path
Return the path
```

2-Opt Research

2-Optimization, or just 2-opt, is a heuristic that takes an existing TSP route and optimizes it by removing 2 edges that cross and replacing them with 2 edges so that a cycle still exists but paths do not cross. However, for each swap, if the resultant tour has a length that is longer than or equal to the length of the tour before the swap, then we undo the swap. The 2-opt procedure has an upper bound of $O(n^3)$, but this is after a preliminary route has been created.

The 2-opt pseudocode is as follows:

2opt(route, x, y):

1. take route[0] to route[x-1] and add them in order to new_route
2. take route[x] to route[y] and add in reverse order to new_route
3. take route[y+1] to end and add them in order to

new_route return new_route

This and the route length comparison is done in a loop for all

eligible nodes that can be swapped – 1.

Particle Swarm / Genetic Research

Particle Swarm Optimization (PSO) is an algorithm inspired by biology. It was developed by computer scientists and biologists studying the movement of thousands of birds and insects in

the air. The algorithm itself has many uses, and can be modified to apply to our travelling salesman problem.

The basic functionality is to move (fly) a group (swarm) of problem solving nodes (particles) throughout the range of possible solutions to a given problem. The range is described as the problem space, and the movement of the particles has a random component with some exceptions. The particles will keep track of their position at all times, and they will have a personal best parameter in which they will remember. From a global perspective, the highest number of particles that have the same personal best data will contribute to the global best. There are some variations that use local best instead of global best.

At the beginning of our problem, the particles will be scattered throughout the map. As our particles move from city to city, they constantly share information with one another. Here is the original formula for the PSO algorithm:

Vid=vid*W+C1*rand(pid-xid)+C2*Rand(pgd-xid)

Vid is the newfound velocity of the given particle, and vid (lowercase v) is the current velocity. Velocity in our problem is actually the amount in which their position has changed on the graph.

W, C1 and C2 are constants. Rand and rand are randomly generated numbers that help spread the particles out in the beginning. Xid is the current location of the individual particles, pid is the personal best position for each particle, and pgd is the global best position for the entire system.

The particles will remember where they are now, where they were in the past, and the optimal position as communicated by the other particles. We can leverage this fact and correlate these three features to break the travelling salesman problem up into 3 decisions about the route. Once we have optimal routes for a region, we can assign those routes to that region.

The particles themselves might assign the same city more than once, and this is a problem. We need to list a city once only. We can do this by assigning initial cities and optimal routes, and if they city has already been listed we will check the route and see if it was faster or slower. If the route was faster, we replace the listed route for the new, optimal route.

But there is a problem with this approach. Cities can only be listed once and sections may contain cities that have already been listed in a previous route section. So there needs to be mechanism to ensure that every city is added to the route and that no city is duplicated in the process.

When swarm is combined with the genetic algorithm, it produces a product that improves the longer it runs.

Chosen Algorithm

We decided to implement several algorithms, including nearest neighbor and 2-Opt. We also explored 3-Opt, but runtime began to be an issue. To get a balance between runtime speed and optimization, we decided to implement a hybrid approach where Nearest Neighbor was used to prime the input file for a more optimal algorithm such as 2-Opt. We also ran different algorithms against the input sets to see which provided the best balance between optimization to runtime. We used Nearest Neighbor for relatively small solution sets, then switched to 2-Opt for larger input files.

Rationale for the Algorithms

We initially began by implementing Nearest Neighbor. However we soon realized that while Nearest Neighbor is fast for small input files, it failed to find a solution after over 30 minutes of running on the larger input sets (input files over 10000 bytes). We then decided to implement 2-Opt, 3-Opt, and a genetic algorithm. However the runtime for 3-Opt was n^3, and was also too slow for larger values of n. While the genetic algorithm was interesting in that the results became more accurate the longer it ran, runtime was also a challenge. Next we implemented 2-Opt, which was fast and fairly accurate as long as we used Nearest Neighbor in conjunction with it. Therefore we decided to use Nearest Neighbor as a primer algorithm, then run the 2-Opt algorithm on the larger data sets. The rationale for selecting these algorithms was we wanted to achieve a balance between algorithmic run time and optimal results.

Algorithm Pseudocode

This section includes the pseudocode for the primary algorithms that we implemented, including Nearest Neighbor, 2-Opt, and a genetic algorithm.

Nearest Neighbor

```
TSP(c, d, path):
        Let the first city in the path be c[1]
        While there are unvisited cities
           Add the next nearest city to the path
        Return the path
```

This algorithm can be run to set up an initial route. Then 2-Opt could be run on that semi-ordered route.

2-Opt

2opt(route, x, y):

1. take route[0] to route[x-1] and add them in order to new_route
2. take route[x] to route[y] and add in reverse order to new_route
3. take route[y+1] to end and add them in order to

new_route return new_route

Genetic

1. Pick a randomly chosen sub-population of individuals (ibest).
2. Generate the initial population P0 with the chosen individuals
3. Evaluate the fitness of the population
4. Pick the best individual from the population P0
5. Initialize the generation counter G
6. While termination criteria is not satisfied, do:
    a. Increase the generation counter G += G + 1
    b. Select above average chromosomes from PG-1 and from a new population
    c. Create offspring from randomly selected parent chromosomesPG:LGA
    d. Evaluate the fitness of the new population P1

    If newFitness of (chromo_best) < oldFitness of (chromo_best):
        Update chromo_best accordingly
    End while loop
    insert(chromo_best) into (chromo_best) Generation

End

Example Instances and Best Tours

Case Number	Tours	Time (secs)	Algorithm
1	130921	0.29	Nearest Neighbor
2	2975	13.51	Nearest Neighbor
3	1812934	827.7	Nearest Neighbor followed by 2-Opt

Competition Instances and Best Tours

Case Number	Tours	Time (secs)	Algorithm
1	5911	0.08	Nearest Neighbor
2	8011	0.63	Nearest Neighbor
3	14826	9.28	Nearest Neighbor
4	19711	74.24	Nearest Neighbor
5	26986	3.6	Nearest Neighbor followed by 2-Opt
6	37689	14	Nearest Neighbor followed by 2-Opt
7	58805	86.2	Nearest Neighbor followed by 2-Opt

Sources Consulted

1. https://en.wikipedia.org/wiki/Held%E2%80%93Karp_algorithm
2. https://www.cl.cam.ac.uk/teaching/1516/AdvAlgo/tsp_demo.pdf
3. https://www.tutorialspoint.com/design_and_analysis_of_algorithms/design_and_analysis_of_algorithms_travelling_salesman_problem.htm
4. https://www.quora.com/How-do-I-solve-the-traveling-salesman-problem-using-dynamic-programming
5. https://repub.eur.nl/pub/101691/ERS-2017-011-LIS.pdf
6. http://papers.philipshield.com/tsp.pdf
7. https://courses.engr.illinois.edu/cs598csc/sp2009/Lectures/lecture_2.pdf