Terence Berry, Bradley Beise, Sheng Bian
Project Group 51
CS 325 Spring 2018
6/8/18

# Traveling Salesperson Problem(TSP) Algorithm Project Report

As a group, we initially studied and researched different methods/algorithms to determine which one was the better algorithm to implement. Terence Berry chose to research the dynamic programming algorithm. Sheng Bian wanted to research a greedy algorithm and 2-opt algorithm. Bradley Beise wanted to research an algorithm based off of a Hamiltonian circuit. Below is the research and pseudocode associated with the algorithms.

### Dynamic Programming

For the Travelling Salesperson Problem (TSP), we will be using dynamic programming to see if we can solve the problem. With the TSP, we are trying to find the shortest path between n cities. We have to find the order of which city to visit to minimize the distance that we travel between the n cities and then return back to the first city. For the distance formula between cities, we will be using:

$d(c1,c2)$ = nearestint ($\sqrt{}$($x$1 − $x$2)^2 + ($y$1 − $y$^2)^2)

First, we should calculate the distances between each city. We would enter this in an adjacency matrix. Then, we would break this problem down into subproblems and make a recursive algorithm. Our recursive algorithm would be based on finding the minimum distance from a vertex to the remaining set of vertices. We would go through all the vertices. Once, we have visited each vertex once, our last distance would be from the last vertex or city to the first one. This should give us the optimal shortest path. We will use the following recursive algorithm:

T (i , s) = min ( ( i , j) + T ( j , S – { j }) ) ;

In doing this dynamic programming approach to the TSP, our time complexity would be exponential. Since we are solving this recursive equation and we have a total of (n-1)2^(n-2) subproblems, which is O(n2^n), each subproblem will take O(n) time to solve. So the time complexity to solve the subproblems and the algorithm in general will be **O(n^(2)2^(n))**.

**Pseudocode:**

```
TSP()
//Calculate distances between each city and save distances in adjacency matrix.
Vector <int> distancesMatrix
    For i to n do
        For j to n do
            d(i,j) = nearestint (√(x1 − x2)^2 + (y1 − y^2)^2)
            distanceMatrix = d(i,j)
//From the first city recursively calculate the distance to the remaining set of cities
Vector <int> matrix
Vector <int> cityList
For i to n cities do
    If matrix[j][i] !=0 and completed[i] == 0
    if(matrix[j][i] + matrix[i][j] < infinity
        Min = matrix[i][0] + matrix[j][i]
        cityList.push_back(i)
        Kmin = matrix[j][i]
    If min != infinity
        distance += kmin
//After the shortest path has been calculated, display distance and path taken
Print distance
For i to n do
    Print cityList[i]
```

**Greedy Algorithm**

The greedy algorithm used to solve Traveling Salesman Problem is called nearest neighbor algorithm. The nearest neighbor algorithm was one of the first algorithms used to determine a solution to the travelling salesman problem.

**Below are the steps of the nearest neighbor algorithm.**

1. Add all cities to available cities list.
2. Start on an arbitrary available city as “starting city” and set “starting city” to current city. Remove “starting city” from available cities list.
3. Find out the shortest path connecting current city and a city V in available cities
4. Set current city to V and remove V from available cities list.
5. Repeat Step 3.
6. After available cities list is empty, return to the “starting city”.

In simple words, In the nearest neighbor algorithm, salesman starts at a random city and visits the nearest city repeatedly until all cities are visited. Finally, it returns to the starting city.

**The pseudocode for nearest neighbor algorithm is written below:**

```
TSP()
   availableCities[] = all the cities
   startingCity = availableCities[0]
   currentCity = startingCity
   remove currentCity from availableCities
   path[]                    //array to store cities represent path
   totalDistance = 0
   while(availableCities not empty)
          currentDistance = infinite
          for each city V in availableCities:
         if(distance between cities< currentDistance)
         currentDistance = distance between cities
         nextCity = V
          totalDistance = totalDistance + currentDistance
          currentCity = nextCity
```

```
        path.append(currentCity)
        remove currentCity from availableCities
    path.append(startingCity)
    totalDistance = totalDistance + distance between currentCity and startingCity
    return totalDistance
```

**2-OPT Algorithm**

2-opt is a simple local search algorithm first proposed by Croes in 1958 for solving the traveling salesman problem. The main idea behind it is to take a route that crosses over itself and reorder it so that it does not. A complete 2-opt local search will compare every possible valid combination of the swapping mechanism.

In simple words, 2-opt algorithm is to continuously change part of the route to find if we can get smaller distance. If we can make the improvement, keep it and continue changing the route.

**The pseudocode for 2-OPT algorithm is written below:**

```
2optSwap(route, i, k) :
    1. take route[0] to route[i-1] and add them in order to new_route
    2. take route[i] to route[k] and add them in reverse order to new_route
    3. take route[k+1] to end and add them in order to new_route
    return new_route;


TSP(existing_route) :

        repeat until no improvement is made
            start_again:
            best_distance = calculateTotalDistance(existing_route)
            for (i = 1; i < number of nodes eligible to be swapped - 1; i++)
                for (k = i + 1; k < number of nodes eligible to be swapped; k++)
                    new_route = 2optSwap(existing_route, i, k)
                    new_distance = calculateTotalDistance(new_route)
                    if (new_distance < best_distance)
                        existing_route = new_route
                        Go to start_again
```

**Algorithm based off Hamiltonian Circuit**

The Hamiltonian circuit is another algorithm we researched. In this algorithm, certain items are stored as they would be on a graph. The number of cities are stored as vertices, the route is stored as links, and the distance is stored as the weight for the links. In the following pseudo code, the min ckt length is stored as a large (inf) number. For each city in the graph or file, it will compare each other cities distance. It will then, in order of the for loop, store the smallest distance if and only if it is smaller than the current min length. Once that min length is found it will continue to the next city and mark that path as visited. The program will then continue the smallest paths until the final stop has been made or the route has made it all the way back.

```
MIN_CKT_LENGTH=INFINITY
for each vertex V in the graph G
        for each vertex V_N in the graph G such that V and V_N are different
                mark all vertices unvisited
                mark V as visited
                        for staring vertex as V and succeeding vertex as V_N, find circuit
                                such that path staring from V_N in that circut yeilds minimum
                                pathlength from V_N for all unvisited vertices by
                                visiting each vertex.( this path is obtained by greedy method).
                          if currently obtained circuit length <= MIN_CKT_LENGTH then
                                set MIN_CKT_LENGTH=newly obtained value
                                copy the new circuit as hamiltonion circuit
                          end if
                        end for V_N
                end for V
```

**Algorithm Decision**

Our decision for the algorithm that we chose to implement was based off of the running time and simplicity of the algorithm. We wanted to make sure that we achieved the fastest run time possible and hopefully get an algorithm that would output a correct answer or within 1.25 of the answer within 3 minutes. We also want to make the implementation as easy as possible. Therefore, We chose to combine greedy algorithm and 2-opt algorithm to implement. For the program, we choose to first find the path using greedy algorithm. Then we can use 2-op algorithm to get shorter distance within 3 minutes. Finally, we write the results to the output file. The pseudocode for both algorithms are provided above. We wrote the program using python.

**Results**

The following are the "best" tours for the three example instances:

| Example Case | Min Distance | Time (sec) |
|---|---|---|
| 1 | 115057 | 0.24 |
| 2 | 2835 | 1.94 |
| 3 | 1964917 | 321.96 |

The following are the best solutions for the competition test instances.

| Test Case | Min Distance | Time (sec) |
|---|---|---|
| 1 | 5639 | 0.05 |
| 2 | 7710 | 0.27 |
| 3 | 12751 | 1.56 |
| 4 | 17734 | 6.53 |
| 5 | 25230 | 21.46 |
| 6 | 35240 | 98.80 |
| 7 | 57489 | 179.17 |

**Reference:**


https://en.wikipedia.org/wiki/2-opt

https://www.thecrazyprogrammer.com/2017/05/travelling-salesman-problem.html

https://en.wikipedia.org/wiki/Nearest_neighbour_algorithm

http://people.ku.edu/~jlmartin/courses/math105-F11/Lectures/chapter6-part3.pdf