CS 325 Project 3 Report
The Travelling Salesman Problem (TSP)

Corey Savage

Table of Contents

Section 1: Algorithm Research

Greedy Algorithm

Description: The Greedy Algorithm for the TSP that I am exploring is based on Kruskal's Algorithm. Kruskal's Algorithm is a minimum-spanning-tree algorithm, used to find the shortest possible path. Like all other Greedy Algorithms, the solution for the TSP is usually suboptimal.

Sort the edges, increasing by weights.
Starting with the least cost edge, compare each edge and select an edge if it:

1. Does not cause a vertes to have a degree of three or more.
2. Does not form a cycle, unless the number of selected edges equals the number of vertices in the graph

Pseudocode:

```
TopologicalSort(G)
Int i = 0
While G.edgeNum is less than G.vertexNum {
        If G.vertex(i).degree and G.vertex(i+1).degree are greater than 1 or
           G.testEdge(G.vertex(i), G.vertex(i+1)) creates circle {
                i increases by 1
        }
        Else {
                G.addEdge(G.vertex(i), G.vertex(i+1))
        }
}
```

Work Cited:

http://lcm.csa.iisc.ernet.in/dsa/node186.html
https://en.wikipedia.org/wiki/Kruskal%27s_algorithm

Christofides' Algorithm

Description: The Christofides' Algorithm for the TSP attempts to find an optimal solution by combining the minimum spanning tree with a minimum-weight perfect matching solution. From Christofides' approach, the TSP tour will be no larger than 1.5 of the optimal solution. With an Eulerian graph the algorithm will use a Eulerian path to find an optimal solution. Once a Eulerian path has been found, convert the TSP by creating shortcuts when cities are visited twice.

Convert problem into Eulerian graph

1. Find minimum spanning tree
2. Convert all vertices of odd order to even

Find an Eulerian tour for the graph
Convert to TSP and find shortcuts

Pseudocode:

```
Graph G=(v,w)
T = minSpanningTree(G)
O = odd(T.vertices)
M = minWeightPerfectMatch(O)
for (Each edge in M and T) {
        Multigraph H = combine(M.edge, T.edge)
}
EC = eulerianCircuit(H)
OptimalTour = hamiltonianCircuit(EC)
```

Work Cited:

https://en.wikipedia.org/wiki/Travelling_salesman_problem#Christofides.27_algorithm_for_the_TSP
https://en.wikipedia.org/wiki/Christofides_algorithm

2-Opt

Description: The 2-opt algorithm was created specifically for solving the TSP. With a 2-opt algorithm, an optimal solution is found by taking a route that crosses over itself and reorder until there are no more possible switches left that can improve/shorten the tour. A complete 2-opt local search will compare every possible valid combination of swapping.

Find Improvement (continue until there are no possible improvements)

1. For each city(i=0, k=i+1)
   a. Swap Route
      i. Take route[1] to route[i-1] and add them in order to new route
      ii. Take route[i] to route[k] and add them in reverse order to new route
      iii. Take route[k+1] to end and add them in order to new rout
   b. If new route is better, make main route

Pseudocode:

```
Graph G = (x, y)
While (possible swap) {
        Distance  = calcTotalDistance(G)
        For (i = 0; in each possible city -1) {
                For (k = i+1; in each possible city) {
                new_G = Swap(g, i, k)
                New_distance = calcTotalDistance(new_G)
                        If (New_distance is less than distance) {
                                G = new_G
                        }
                }
        }
}
Swap(graph g, int i, int k) {
  for (c = 0; each city until i - 1 )
     newG.addCity(g[c])
  for (c = i; each city until k)
     newG.addCity(g[k decreasing to i])
  for (c = k + 1; until last city)
     newG.addCity(g[c])
  return newG
}
```

Work Cited:

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

Section 2: My Algorithm

Description: The Algorithm I chose to implement to solve the TSP was a 2-opt algorithm, very similar to the algorithm researched above. The main difference being, instead of implementing a complete 2-opt local search my algorithm will perform a 2-opt swap over the entire problem set 20 times before exporting the tour results. Each improved route found resets the counter tracking the number of iterations.

```
The algorithm is passed a list of cities, called v[city][coordinates]
Create a new list of cities, newV, and set it equal to v
Initialize  c, which represents a counter to stop swapping
While c is less than 20:
    Initialize d and set it equal to calculate_total_distance(v)
    Loop(i=0) through every city -1
        Beginning with previous loop city + 1, loop(k=i+1) through every city
            Create a new route, newV, by swapping routes i and k in list v:
                Take route[1] to route[i-1], add them in order to newV
                Take route[i] to route[k], add them in reverse order to newV
                Take route[k+1] to end, add them in order to newV
            Initialize newD and set it equal to calculate_total_distance(newV)
            If newD is less than d:
                Reset c to 0
                Set d to newD
                Set v to newV
    If loop has completed without finding an improved route, increase c
```

Work Cited:
https://en.wikipedia.org/wiki/2-opt
www.technical-recipes.com/2012/applying-c-implementations-of-2-opt-to-travelling-salesman-problems/

Pseudocode:

```
two_opt (v[cities][coordinates] {
        newV = v
        c = 0
        d = calculate_total_distance(v)
        While (c < 20) {
                For (i = 0; in each possible city -1) {
                        For (k = i+1; in each possible city) {
                                newV = Swap(v, i, k)
                                newD = calcTotalDistance(newV)
                                If (NewD is less than d) {
                                        v = newV
                                        d = newD
                                        c = 0
                                }
                        }
                }
                c++
        }
        Return v
}
Swap(graph g, int i, int k) {
  for (c = 0; each city until i - 1 )
      newG.addCity(g[c])
  for (c = i; each city until k)
      newG.addCity(g[k decreasing to i])
  for (c = k + 1; until last city)
      newG.addCity(g[c])
  return newG
}
```

Selection Choice:

Selecting a 2-opt algorithm appeared to be the optimal choice considering the guidelines we were provided. With a 2-opt algorithm, the required ratio needed for the three example problems provided can be achieved. Additionally, with the 2-opt algorithm the counter to break the algorithm can be modified to trigger after less loops without finding improvements. **This** customizability allows me to use the 2-opt algorithm for both the example and competition problems

Work Cited:

https://en.wikipedia.org/wiki/2-opt
www.technical-recipes.com/2012/applying-c-implementations-of-2-opt-to-travelling-salesman-problems/

Section 3: Example-Problem Results

tsp_example_1.txt.tour:

Distance = 111366

Run Time = 19.671123 seconds

Optimal Accuracy = 111366 / 108159 = 1.02

tsp_example_2.txt.tour:

Distance = 2760

Run Time = 314.440000 seconds

Optimal Accuracy = 2760 / 2579 = 1.07

Tsp_example_3.txt.tour: (Ran overnight, I attempted to lower the runtime and tour file got overwritten)

Distance = 1919162

Run Time = 39367.277013 seconds

Optimal Accuracy = 2760 / 2579 = 1.22

Section 4:Competition-Problem Results

Not sure why, but when testing these files my algorithm had all sorts of problems. Either, the first city would jump around, the distance would be wrong, or negatives would pop out.

test-input-1.txt:

Time Limit = 3 Minutes

Best Distance:1682

Time Limit = Unlimited

Best Distance:1682

test-input-2.txt:

Time Limit = 3 Minutes

Best Distance: 1928

Time Limit = Unlimited

Best Distance: 1836

test-input-3.txt:

Time Limit = 3 Minutes

Best Distance: 7326

Time Limit = Unlimited

Best Distance: 7326

test-input-4.txt:

Time Limit = 3 Minutes

Best Distance: 167106

Time Limit = Unlimited

Best Distance: 155023

test-input-5.txt: (Files 5, 6, and 7 could not improve from the starting path within the timeframe)

Time Limit = 3 Minutes

Best Distance: 343683

Time Limit = Unlimited

Best Distance: 186364

test-input-6.txt:

Time Limit = 3 Minutes

Best Distance: 674896

Time Limit = Unlimited

Best Distance: 10877

est-input-7.txt:

Time Limit = 3 Minutes

Best Distance: 1630355

Time Limit = Unlimited

Best Distance: 1607900