Analysis of Algorithms
Instructor: Julianne Schutfort

Final Project
Travelling Salesman Problem

Group 13
Edgar Marquez
Stephen Townsend
Nadia Viscovich

Group Project: Traveling Salesman

Traveling Salesman Problem (TSP) is classical and most widely studied problem in Combinatorial Optimization. It has been studied intensively in both Operations Research and Computer Science since 1950s as a result of which a large number of techniques have been developed to solve this problem.[1]

TSP can be described as: Given a set of cities, and the distances between each pair of the cities, the aim of the salesman is to find a minimum path that visits each city exactly once and gives the minimized the total distance travelled.

We focused our research on the following algorithms: nearest neighbor algorithm, the ant colony optimization and the 2 opt algorithm.

Nearest Neighbor Algorithm

The nearest neighbor algorithm is a greedy algorithm that finds the candidate solution to TSP using simple means. This method is a natural strategy for the TSP, because it mimics the way the traveling salesman selects a travel route. It selects a starting point and then always selects the nearest city to be added to the tour, it then "walks" to that city and repeats by choosing a new non-selected city, until all cities are in the tour. To complete the tour, an edge is added between the last selected city and the starting city. A general version of this heuristic has running time of $\Theta(N^2)$ [2].

Steps

1. Start with any random vertex, call it current vertex.
2. Find an edge which gives minimum distance between the current vertex and an unvisited vertex, call it V.
3. Now set that current vertex to unvisited vertex V and mark that vertex V as visited.
4. Terminate the condition, if all the vertices are visited at least once.
5. Go to step 2.

The output of the algorithm is the sequence of all visited vertices. This implies a short tour but not an optimal one. Because of its greediness nature the algorithm misses some shorter routes that can easily be detected with human insight. It is possible that nearest neighbor algorithm will not find a feasible tour even if there is one.

Advantages

- Simple to implement
- Executes quickly for small data sets.
- Very flexible decision boundaries

Disadvantages

- When data set is large it may take a lot of space
- Computational costs: A lot of time may be needed for testing. [4]

Pseudo Code

Input: a set of n points in d dimensions

P = $\{p_1, p_2, \dots p_n\}$

Desired Output: For each point p ∈ P the nearest point to p

Algorithm NN (P = $[p_1, p_2, \dots p_n]$)

for all $i \in [1, n]$, $j \in [1,n]$

compute $d\,[\,i, j\,] = \| p_i - p_j \|$

for $i = 1$ to n

dist [i] ← ∞

for $j = 1$ to n

if $i \neq j$ and d[i, j] < dist [i]

then

dist [i] ← d[i, j]; NN← j

return NN, dist

Ant Colony Optimization

Ant colony optimization (ACO) is an algorithm that is meant to simulate the real behavior of ants in nature. Ants are very good at searching for the shortest path from a food source to the next without having much information of the world around them. This is achieved by the help of pheromones that ants leave behind on their travels. The larger the quantity of pheromones along a specific path the more likely the ant would be to follow it and

excrete pheromones of its own. As time passes pheromones will evaporate and if ants stop using a path the quantity of pheromones will deplete resulting in ants following other paths. In the algorithm “ants” scan a graph G and their aim is to find the optimal solution. Every “ant” has its own memory which is used to record the information of the path it has traveled. The “ants” are distributed to the n nodes randomly and each “ant” chooses the next node to visit according to several rules [5]

Ant Colony Optimization Rules

- It must visit each node exactly once
- A distant node has less chance of being chosen
- The path with the larger quantity of “pheromones” laid on an edge between 2 nodes has a greater chance of being chosen
- Having completed its tour, the ant deposits more “pheromones on the edges it traveled, it the tour was short
- After each iteration, trails of pheromones evaporate

The overall result is that when an “ant” finds a short path, other ants are more likely to follow that path and with accumulation of “pheromones eventually leads to all the “ants” following a single path. The great advantage over the use of exact methods is that ACO algorithm provides relatively good results by a comparatively low number of iterations, and is therefore able to find an acceptable solution in a comparatively short time, so it is useable for solving problems occurring in practical applications. [6]

Pseudocode [7]

```
Procedure ACOMetaheuristicStatic
        Set parameters, initialize pheromone trails
        While (termination condition not met) do
                ConstructAntSolutions
                ApplyLocalSearch    %optimal
                UpdatePheromones
        end
end
```

2-OPT Algorithm

2 optimal or 2-OPT is an algorithm that takes a route (2 edges) that crosses over each other and reorders them so that they do not cross. To do this, the 2-OPT algorithm is simply an iterative approach where in each iteration it strategically replaces two old edges with two new ones, so that all edges still form a tour (the total number of edges remains the same) and the operation decreases the tour cost [8] What 2-OPT is looking for are 2 adjacent edges that have a lower cost than the current (crossing) edges. If a solution is found, then the algorithm will swap those edges to achieve a more optimal solution. To achieve the swap 2-OPT actually uses another, standalone, algorithm that is also potentially used to solve the TSP. That algorithm is the nearest neighbor (Please see above to read more about this algorithm)

Pseudocode [9]

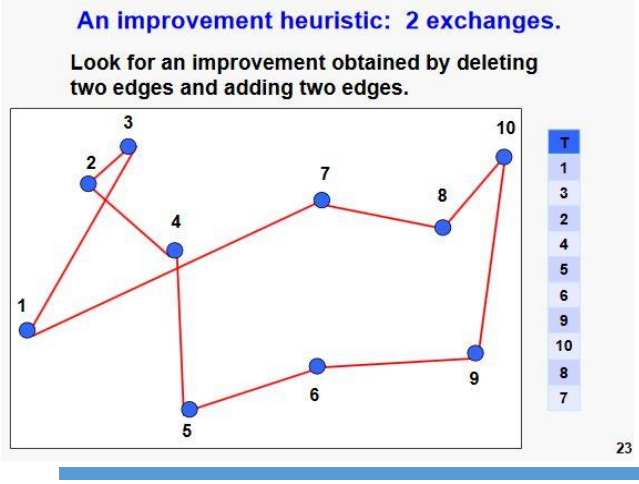
2optSwap(route i,k) {

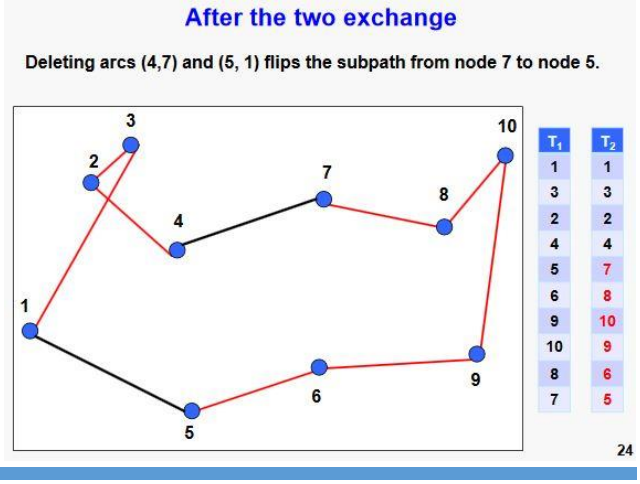
1. Take route[0] to route[i-1] and add them in order to new_route
2. Take route[i] to route[k] and add them in reverse order to new_route
3. Take route[k+1] to end and add them in order to new_route

return new_route;

}

Below are two images that demonstrate the process above. The slide on the left is going to swap out edges at vertices [1,7] & [4,5].





2-OPT from MIT PDF Slide, showing the 2-OPT swap function [10]

The time complexity of 2-OPT is a complex situation. I found a resource that stated that "[t]he 2-opt algorithm has a time complexity of O($N \cdot n^2$) time, where N is the number of stages that depends on the stop criteria." [11] I have also found a report from the Electronic Colloquium in Computational Complexity that stated that "there are numerous experimental studies on the performance of 2-Opt. However, the theoretical knowledge about this heuristic is still very limited. Not even its worst case running time on Euclidean instances was known so far." [12]

Overall, even with the uncertainty of the time complexity, according to an article from the International Journal of Applied Operational Research, "[The 2-OPT] algorithm remains the key ingredient in the most successful approaches for finding high-quality tours and is widely used to generate initial solutions for other algorithms." [13]

Implementation

As a group we decided to implement the 2-Opt Algorithm as it was understood by all three members of the group and could potentially include the use of the nearest neighbor algorithm, which we also researched. 2-OPT is a local search heuristic, meaning it needs a path to improve upon. The constructive algorithm we chose to provide this path was nearest neighbor. Our algorithm works in the following way:

Once the initialization of the problem is done (cities read, arrays created, distances calculated, etc…) nearestNeighbor() is called. The nearestNeighbor() makes the initial tour which is then passed to twoOpt(). The algorithm twoOpt() makes use of a nested loop to send pairs of cities to the twoOptSwap() function. The twoOptSwap() function checks if a better solution exists than that passed to it. If the case is that a better solution does exist it then invokes makeABetterTour(). The makeABetterTour() function is implemented to (as it states in the name) make a better tour than the initial tour. This is accomplished by makeABetterTour() by rearranging the path before, between and after the cities it was passed.This tour is then passed back to twoOptSwap() that then passes back the improved tour length to twoOpt(). The steps repeat in this fashion until the best solution is found, after which the program will write out the solution in the specified format.

We believed this algorithm to be conceptually easy to implement, would be fast, and

resulted in the most optimum tour lengths than simply running the nearest neighbor algorithm alone.

The first implementation did not include the nearest neighbor algorithm and the times we were getting for the competition test input files where over three minutes. We decided that using a nearest neighbor implementation along with the 2-Opt algorithm would speed up the algorithm. Following are the results:

File Name	Tour Length	Time in seconds	Ratio
Tsp_Example_1.txt	112,947	0.003	1.04
Tsp_Example_2.txt	2,741	0.091	1.06
Tsp_Example_3.txt	1,671,166	52,833.400	1.06

Results from Competition Test Inputs

File Name	Tour Length	Time in seconds
Test-input-1.txt	5,639	0.001
Test-input-2.txt	7,660	0.006
Test-input-3.txt	12,724	0.092
Test-input-4.txt	17,797	0.778
Test-input-5.txt	24,831	6.642
Test-input-6.txt	34,720	64.684
Test-input-7.txt	53,843	1,401.563

References

[1] Applegate, D. L., Bixby, R. E., Chvátal, V. & Cook, W. J. (2006). The Traveling Salesman Problem: A Computational Study, Princeton University Press, Princeton, New Jersey.

[2] The Traveling Salesman Problem: A case study in local optimization by David S. Johnson and Lyle A. MuGeoch 1995

[3] G. Laporte, “The Traveling Salesman Problem: An overview of exact and approximate algorithms”, Elsevier Science Publishers, vol. 59, pp.231–247, July 1995.

[4] http://dataaspirant.com/2016/12/23/k-nearest-neighbor-classifier-intro/

[5] https://en.wikipedia.org/wiki/Ant_colony_optimization_algorithms#Convergence

[6] http://www.ef.uns.ac.rs/mis/archive-pdf/2011%20-%20No4/MIS2011_4_2.pdf

[7] http://www.swarmbots.org/~mdorigo/HomePageDorigo/thesis/master/DarquennesMASTER.pdf

[8] *An Adaptive k-opt Method for Solving TSP*. http://iurobotics.net/publications/cdc16-tsp.pdf Pg1.

[9] *2-op*t. https://en.wikipedia.org/wiki/2-opt.

[10] *MIT 2013 Lecture*. Slides 23 & 24 https://ocw.mit.edu/courses/sloan-school-of-management/15-053-optimization-methods-in-management-science-spring-2013/lecture-notes/MIT15_053S13_lec17.pdf.

[11] *An Adaptive k-opt Method for Solving TSP*. http://iurobotics.net/publications/cdc16-tsp.pdf Pg5.

[12] Electronic Colloquium in Computational Complexity. *Worst Case and Probabilistic Analysis of the 2-Opt Algorithm for the TSP*. Report No. 92 (2006). https://www.google.com/url?sa=t&rct=j&q=&esrc=s&source=web&cd=2&ved=0ahUKEwjPxu7ZrtDXAhWKjFQKHe-GCw8QFggsMAE&url=https%3A%2F%2Feccc.weizmann.ac.il%2Freport%2F2006%2F092%2Fdownload%2F&usg=AOvVaw2-8xkuTAJsmofwVeW_kkUH

[13] International Journal of Applied Operational Research. An Approach for Solving Traveling Salesman Problem. http://ijorlu.liau.ac.ir/article-1-225-en.pdf.