**Project Report**

Group 41

Anthony De La Torre

Kaiyuan Fan

Cory Melendez

**Algorithms Researched**

1.Metric TSP

This algorithm calls that the graph satisfies the Triangle Inequality, that is: One side of a triangle is at least as short as the sum of the other two sides.

The algorithm is as follows:

1. Create a minimum spanning tree (MST) from a graph *G*, we will call *S*, such that total weight of S which we will call *w(s)* is less than or equal to all other spanning trees in a graph *G* which we will call *S’*. That is: *w(S) <= w(S’)*. We must have an MST in *G*, otherwise, that would imply that we have a vertex in G that is does not have a neighbor, in which case there is no solution for the TSP of *G*, *TSP(G).*
   1. We now claim that the weight of the MST of G must be less than the weight of the optimal solution of *TSP(G),* or *w(TSP(G)).*

*Proof:*

Take the optimal solution, *TSP(G)*, we will call Opt and remove an edge e from it. *Opt-e* is now a spanning tree. We know that the MST *S* is less than any spanning tree including *Opt-e*, so: *w(S) <= w(Opt-e) = w(Opt) – w(e) for any edge e greater than 0.*

1. We then create a cycle *W* on *S*, by doing a depth first search yielding twice the weight of S (since we descend into the tree then ascend back up), so *w(W) = 2\*w(M) < 2 \* Opt.* However, this is not the optimal solution and it does not solve the problem since we visit vertices multiple times.
2. Using the triangle inequality, we can create our solution by using the vertices only the first time that they are seen, allowing us to remove intermediate vertices in a path and not increase the path weight.

This algorithm requires that our graph obeys the triangle inequality, but since so many problems such as air flights and the costs associated with them do not behave this rule, it is not an algorithm that can be used on every TSP problem.

From ii) we see that we get a solution at least as good as two times the optimal solution in polynomial time since we can find an MST in polynomial time using an algorithm such as Kruskal’s MST Algorithm.

2.Christofides Algorithm

The first algorithm that can be considered to solve the TSP is the Christofides algorithm, named after Nicos Christofides who published it in 1976. This algorithm also requires the graph to be in the metric space, that is they are symmetric and obey the triangle inequality.

1. Find a minimum spanning tree *T* of a graph *G*.
2. Take the subgraph of *G*, *G’*, restricted to the vertices of odd degree in T which is complete since we have not removed any edges
3. Find a minimum weight Matching *M* on *G’*.
   1. We know *w(M)* *<= .5\** *Optimal Solution*. [1]
4. Join the edges of M with those of the *MST* T to create a graph H w/all vertices having even degree. This subgraph has a most weight *w(T) + w(M) <= 1.5 \* optimal solution*
5. Create a Eulerian Tour (A path that visits every edge once) on H and reduce like we did in 1iii)

Again, this algorithm requires that the graph obey the triangle inequality as well. The most time consuming of these are the min weight matching which cost O(n^3).

3. Greedy Neighbor Algorithm

The simplest algorithm is the Greedy Neighbor Algorithm which gradually constructs a tour by repeatedly selecting the shortest edge, it is not as precise as the other algorithms we have discussed or implemented however.

1. Sort all the edges
2. Select the shortest edge and add it to the tour if it doesn’t create a cycle with less than N edges and does not increase the degree of a vertex to more than 2.
3. Do we have N edges in our tour? If not, repeat step 2 until we do.

The complexity of this algorithm is O(n^2lg(n))

4. 2-opt Algorithm

The 2-opt algorithm is a simple local search algorithm that has application to solve the TSP. This algorithm improves on an existing tour, so we use the solution from greedy neighbor approach as above. The main idea behind it is to take a route that crosses over itself and reorder it so that it does not.

1. Do a greedy neighbor algorithm to find a tour of existing neighbors
2. Swap two edges so that the path is still connected and see if the overall distance get shorter
3. Repeat step ii) until the swapping of any edges result in no further improvement

Finally, we selected to implement 2-opt algorithm because it seemed relatively simple to conceptualize. 2-opt algorithm is efficient to compute the path in a limited amount of time and it is complex enough to get close to the optimal results.

**Pseudo Code**

void twoOptSwap(int i, int j, int cityIndex, City\* tour, City\* tempTour)

int count = 0;

for (int a = 0; a <= i - 1; ++a)

tempTour[a] = tour[a];

for (int a = i; a <= j; ++a)

tempTour[a] = tour[j - count];

count++;

for (int a = j + 1; a < cityIndex; ++a)

tempTour[a] = tour[a];

int twoOpt(City\* cities, int cityIndex)

int optomize = 0;

int bestDistance;

time\_t end;

end = time(NULL) + 180;

bestDistance = findTotalDistance(cities, cityIndex);

for (int i = 0; i < cityIndex - 1; i++) {

if (time(NULL) > end)

break;

for (int j = i + 1; j < cityIndex; j++) {

if (time(NULL) > end)

break;

City\* tempTour = new City[cityIndex];

twoOptSwap(i, j, cityIndex, cities, tempTour);

int tempDistance = findTotalDistance(tempTour, cityIndex);

if (tempDistance < bestDistance) {

optomize = 0;

bestDistance = tempDistance;

copyTour(tempTour, cities, cityIndex);

}

delete[] tempTour;

}

}

return bestDistance;

}

**Testing Results**

Best Tours for Three Example

|  |  |  |  |
| --- | --- | --- | --- |
| File | Best Distance | Running Time | Optimal Distance |
| tsp\_example\_1 | 108873 | 0.010 seconds | 108159 |
| tsp\_example\_2 | 2779 | 0.568 seconds | 2579 |
| tsp\_example\_3 | 1935563 | 1800 seconds (stopped) | 1573084 |

Best Solutions for The Competition Test

|  |  |  |
| --- | --- | --- |
| File | Best Distance (3 minute limit) | Best Distance (Unlimited time) |
| test-input-1 | 5119 | 5119 |
| test-input-2 | 7666 | 7666 |
| test-input-3 | 13292 | 13292 |
| test-input-4 | 18243 | 18243 |
| test-input-5 | 25748 | 25748 |
| test-input-6 | 36989 | 36574 |
| test-input-7 | 62599 | 60788 |