TSP problem:

The 150 largest U.S. cities

Luis Steven Lin

CEE 512 – Logistics System Analysis
Homework Handout Set #2
Professor Y. Ouyang
April 2, 2012

Contents

1. Introduction

The objective of this report was to solve the travelling salesman problem (TSP) for the largest 150 US cities given their longitude and latitude coordinates. In other words, the goal was to find the shortest possible route that visits each city exactly once and returns to the origin city. Since the TSP is an NP-hard problem, the farthest insertion heuristic was implemented to find a feasible and reasonable solution.

The first step was to compute the pair-wise distance between cities using an appropriate formula and store the results in a 150x150 matrix, in which each entry represents the distance between cities (edges of the graph). This matrix was called the cost (or distance) matrix. Then a route was found using the mentioned heuristic, and finally a plot of the tour was made. The appendix shows all codes for all files used, as well as some references and resources related to coding and plotting in Matlab.

2. Data Input

Given the coordinates for the latitude and longitude of a pair of cities (located in file "data150cities.xls"), the distance between the cities was calculated using the *haversine* [1] formula, which computes the great-circle distance between two points (i.e. the shortest distance over the earth's surface). Define the following variables:

φ_1, λ_1 = geographical latitude and longitude of city 1 respectively (in radians)

φ_2, λ_2 = geographical latitude and longitude of city 2 respectively (in radians)

$\Delta\varphi$ = difference in latitude between the cities (in radians)

$\Delta\lambda$ = difference in longitude between the cities (in radians)

R = Earth's radius (mean radius = 6,371km)

a = square of half the chord length between the cities

c = angular distance (in radians)

d = distance between the cities (in Km)

Then, after converting the given latitude and longitude in degrees to radians, the distance between cities can be computed using the following formulas (see 6.A findDistance.m for code):

$$\Delta\varphi = \varphi_2 - \varphi_1$$

$$\Delta\lambda = \lambda_2 - \lambda_1$$

$$a = \sin^2(\Delta\varphi/2) + \sin^2(\Delta\lambda/2) \cdot \cos(\varphi_1) \cdot \cos(\varphi_2)$$

$$c = 2 \cdot atan2(\sqrt{a}, \sqrt{1-a})$$

$$d = R \cdot c$$

A more accurate and realistic estimate for the road distance could be obtained using the distance by a driving direction engine (e.g. Google map), which takes into account the actual roads between cities.

The formula was used to compute all the distances between pair of cities, resulting in a symmetric 150x150 matrix. The details for the code can be seen in the appendix (6.B costmatrixTSP.m). The distance matrix among Cities 1-10 is shown in table Table 2.1.

[1] http://www.movable-type.co.uk/scripts/latlong.html

Table 2.1 Distance matrix among Cities 1-10

	1	2	3	4	5	6	7	8	9	10
1	0	3953	1155	2285	125	3905	785	2206	3444	2547
2	3953	0	2810	2220	3857	187	3192	2010	589	1948
3	1155	2810	0	1510	1073	2774	383	1285	2325	1685
4	2285	2220	1510	0	2162	2090	1781	361	1632	303
5	125	3857	1073	2162	0	3806	717	2089	3342	2426
6	3905	187	2774	2090	3806	0	3157	1901	478	1809
7	785	3192	383	1781	717	3157	0	1605	2707	1991
8	2206	2010	1285	361	2089	1901	1605	0	1426	406
9	3444	589	2325	1632	3342	478	2707	1426	0	1363
10	2547	1948	1685	303	2426	1809	1991	406	1363	0

Just to check the correctness of the distances, the website listed in the footnote in the previous section was used to compute the distances between city 1 (New York) and city 2 (Los Angeles). The result (Figure 2.1) agrees with the distance (3953 km) found by the program from the appendix. As an additional check, the travel road distance found by Google Map was 4,469 km, indicating that the distance computed by the formula is a reasonable estimate of the actual road travel distance.

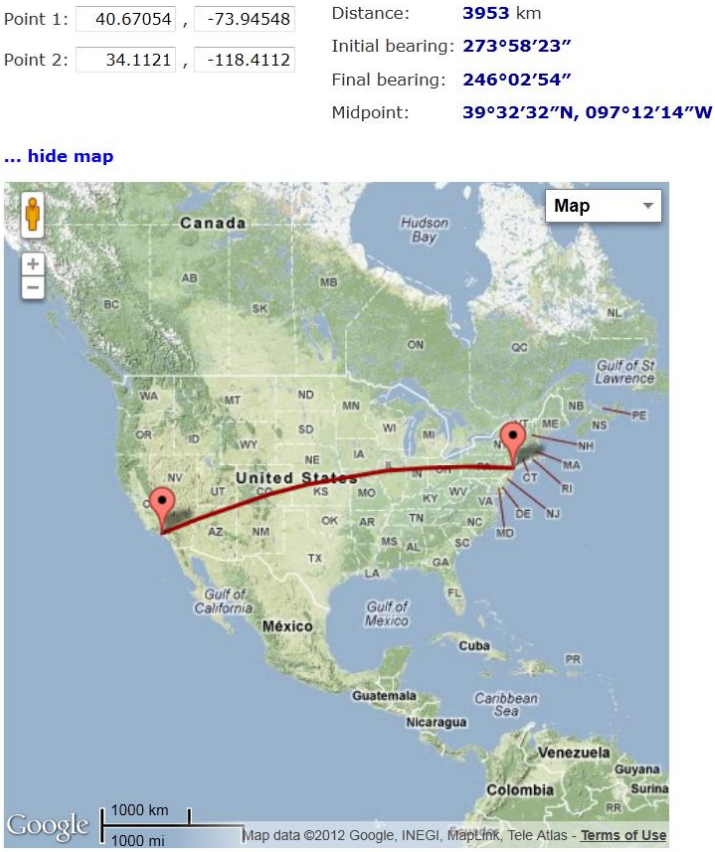

Figure 2.1 Distance between NY and LA computed by website

3. Algorithm Implementation

3.1 Overview

The farthest insertion heuristic was implemented in Matlab. The main file (6.F CEE512_hw5.m) calls all functions and used their outputs for inputs of other functions. The cost matrix found in the previous section, was used as input to find the TSP (6.D findTSP.m). This function also calls a function (6.C insertToTour.m) to insert a city into the appropriate location in the tour. The output for the function findTSP include the tour (the route in which cities are visited), and the tour length (total distance in km of route in km). These two outputs and the original coordinate of the cities were used as inputs to generate a plot of the tour (6.E plotTSP.m)

3.2 Farthest Insertion Heuristic

The Matlab code for the farthest insertion heuristic can be found in the appendix (6.D findTSP.m). The idea of the algorithm is to add the farthest city to the tour at each step such that the insertion cost (distance to any city in the tour) is the lowest. Choosing the farthest city might seem counterintuitive, but on average better solutions (lower route distance) are found compared to the nearest insertion heuristic because the former takes into account the global picture of the points (e.g. in the latter stage of the algorithm, it avoids choosing a very far city that might disrupt the tour). The general steps of the heuristic are as follows (based on lecture notes):

1. Select a node and its closest node and build a tour of two nodes
2. Select the farthest node k to the tour from set V (nodes not in tour), that is not in set T (nodes in tour). That is find k that maximizes $c(i,k), i \in T, k \in V$
3. Insert k into the tour such that the insertion cost is minimized. That is find the edge $(i,j), i,j \in T$ that minimizes $c(i,k) + c(k,j) - c(i,j)$. Replace edge (i,j) with (i,k) and (j,k). In other words, insert k between i and j.
4. Repeat step 2 and 3 until all cities are added to the tour (set V empty).

Note that in this report, the algorithm was run for every possible starting node (city) to determine the best solution given by this heuristic.

4. Results

The numerical results can be found in Figure 4.1, which displays the command window after running CEE512_hw5.m. The output shows that the computation time for the heuristic starting at city 3 (which gave the lowest cost/distance) was 0.056869 seconds, while the total computation time for the entire program (CEE512_hw5.m) was 9.984439 seconds. The total tour length was 23167.4 km and the tour path as shown in the output. The computer platform and software used are also shown in the output.

The graphical representation of the tour overlaid with the US map can be found in Figure 4.2. After inspecting the tour, the solution seems reasonable (e.g. the tour does not have edges that crosses each other).

```
File Name: CEE512_hw5.m
Date: 3/27/12
Author: Luis S Lin
Description: TSP solution for 150 cities using farthest insertion heuristic
Inputs: data150cities.xls

ans =

--Best Tour found by Farthest Insertion Heuristic--
Computation TIme for heuristic starting at city 3: 0.056869 seconds
Tour Length: 23167.4 km
Tour Path:
3-17-80-140-123-116-7-47-95-13-86-43-16-23-68-39-48-64-103-108-104-20-98-125-136-
121-81-122-54-65-1-5-12-19-96-73-97-131-60-36-111-72-128-85-117-34-67-56-143-33-
124-42-28-18-25-99-110-127-15-101-114-148-44-82-53-63-145-35-106-58-83-76-24-113-70-
75-92-74-150-4-62-147-10-27-8-88-138-109-59-84-107-37-22-32-51-120-137-9-115-146-
61-129-6-139-66-102-132-134-57-50-118-87-31-133-142-135-89-2-119-93-45-11-94-14-38-
40-71-100-130-29-91-21-90-144-105-141-26-69-52-49-149-112-30-78-46-77-41-55-79-126-3


Computer Platform: HP Z210, Intel Core i5-2400 3.1Ghz Processors, 8GB of RAM
Software: 7.12.0.635 (R2011a) , 64-bit (win64)
Total computation time for program CEE512_hw5.m
Elapsed time is 9.984439 seconds.
```

Figure 4.1 Matlab Command Window Output

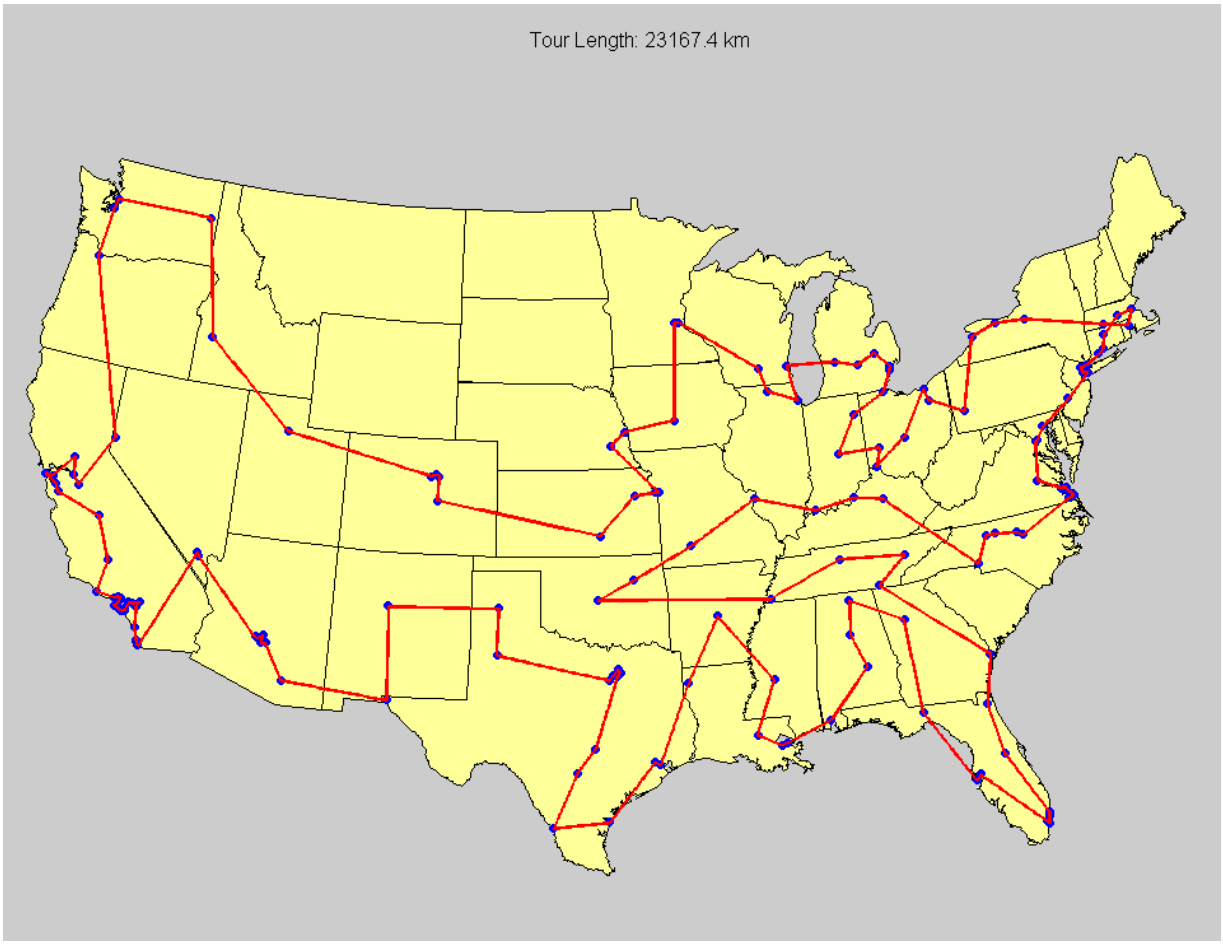

Figure 4.2 Matlab output: US Map with TSP tour

5. Summary

This report implemented the farthest insertion heuristic in Matlab to determine a route to visit the largest 150 cities in the United States. The outputs of the program included the tour length, the tour and the graphical representation of the tour. For future work, the nearest insertion heuristic and minimum spanning tree heuristic can be implemented to compare their results to the found in this report. In addition, the linear program can be implemented to compare the results with the optimal solution.

6. Appendix

6.A findDistance.m

```
%% Function : findDistance

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% File Name: findDistance.m
% Date: 3/27/12
% Author: Luis S Lin
% Description: Compute the distance between two points (in km) given the
%              latitude and longitude coordinates in degrees
%              Calculations based on great circle distance.
%
% Inputs: 1) latitude 1
%         2) latitude 2
%         3) longitude 1
%         4) longitude 2
%
% Ouput: 1) distance in km bewteen points
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%% Function : findDistance

function[d] = findDistance(lat1,lat2,lon1,lon2)
    % Get coordinates and convert to radians
    lat1=degtorad(lat1);
    lat2=degtorad(lat2);
    lon1=degtorad(lon1);
    lon2=degtorad(lon2);
    % Compute coordinate differences
    dlat=lat2-lat1;
    dlon=lon2-lon1;
    % Square of half the chord length between the cities
    a=sin(dlat/2)^2+cos(lat1)*cos(lat2)*sin(dlon/2)^2;
    % angular distance (in radians)
    c=2*atan2(sqrt(a),sqrt(1-a));
    % Earth's radius (mean radius = 6,371km)
    R=6371;
    % Compute distance
    d=R*c;
```

6.B costmatrixTSP.m

```
%% Function: costmatrixTSP

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% File Name: costmatrixTSP.m
% Date: 3/27/12
% Author: Luis S Lin
% Description: read matrix (latitude and longitude coordinates)
%              and create a distance (cost) matrix by compuing the
%              distances between each pair of cities
%
% Files used: findDistance.m
%
% Inputs: 1) matrix containing latitude and
%         longitude (absolute value, so N and W assumed) of cities
%
% Outputs: 1) cost matrix
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%% Create distance matrix
function[c]=costmatrixTSP(coords)
n = length(coords(:,1));
c = zeros(n,n); % cost matrix
% iterate through coords
for i=1:(n)
    for j=(i+1):n
        % Get coordinates of two cities
        lat1=coords(i,1);
        lat2=coords(j,1);
        lon1=coords(i,2);
        lon2=coords(j,2);
        % Compute disance and save in matrix (symmetric)
        c(i,j)=findDistance(lat1,lat2,lon1,lon2);
        c(j,i)=c(i,j);
    end
end
```

6.C insertToTour.m

```
%% Function: insertToTour

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% File Name: insertToTour.m
% Date: 3/27/12
% Author: Luis S Lin
% Description: given an array, the function inserts a k value bewteen
%              values i and j in the array. Values i and j have to be
%              located consecutively in the array. Note that the function
%              works also when the first and last value of the array are
%              the same.
%
% Inputs: 1) i=value 1
%         2) j=value 2
%         3) k=value to be inserted
%         4) array
%
% Outputs: 1) cost matrix
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%% Function: insertToTour

function[arrayNew]=insertToTour(i,j,k,array)

%% Determine position of k to be inserted

% find the position of i and j
position_i=find(array==i);
position_j=find(array==j);

% if both i and j are not located at the end points
if (length(position_i)==1)&(length(position_j)==1)
    if(position_i==position_j+1)
        position_k=position_i;
    elseif(position_i==position_j-1)
        position_k=position_j;
    else
    disp('Error1')
    end

% if j is located in the endpoints
elseif (length(position_i)==1)&(length(position_j)==2)
    if(position_i==position_j(1)+1)
        position_k=position_i;
    elseif(position_i==position_j(2)-1)
        position_k=position_j(2);
    else
    disp('Error2')
```

```
% if i is located in the endpoints
elseif(length(position_i)==2)&(length(position_j)==1)
     if(position_j==position_i(1)+1)
        position_k=position_j;
    elseif(position_j==position_i(2)-1)
        position_k=position_i(2);
    else
    disp('Error3')
    end
else

disp('Error: either i or j have more than 2 entries in array, or both have 2 or more↙
entries')

end

%% Insert k

array; % initial array;
k; %  value to insert
position_k;% index of current array where the value k is to be inserted.

% Create a new array
arrayNew = zeros(1,length(array)+length(k));
% Insert k
arrayNew(position_k +(0:length(position_k)-1)) = k;
% Fill with the rest of original entries
arrayNew(~arrayNew) = array;
```

6.D findTSP.m

```
%% Function: findTSP

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% File Name: findTSP.m
% Date: 3/27/12
% Author: Luis S Lin
% Description: find a TSP given the distance matrix of cities by appying the
%              farthest insertion heuristic
%
% Files used: insertToTour.m
%
% Inputs: 1) matrix containing distances between cities
%
% Outputs: 1) TSP tour (path with city numbers ordered)
%          2) length of tour in kM
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%% Function: findTSP
function[tourTSP,lengthTSP] = findTSP(c)

n = length(c(:,1));          % Number of cities
maxcost=max(max(c))*3;       % Upper bound for insertion cost
lengthTSP=max(max(c))*n;      % Upper bound for TSP length
% repeat algorithm for all possible starting cities
for startCity=1:n

tstart=tic; % Start computation time

%% Step 1: Create a staring tour of two cities
T=[];   % Create array containing list of cities in tour
V= 1:n; % Create array containing loist of cities not in tour
TOUR = zeros(n,n); % Create cost matrix of cities' edges in tour
                   % (Adjacency matrix with entries equal to distance
                   %  between cities in tour)
% First city
first=startCity;  % Select first city
T(1)=first ;      % Add to array containing cities in tour
V(V==first) = []; % Remove selected city from array not in tour

% Second city
minvalue=min(c(first,c(first,:)~=0));  % Find min distance from first city
mincity=find(c(first,:)==minvalue);    % Find correponding city (index)
mincity=mincity(1);                    % If tie, pick first listed city
T(2)=mincity;                          % Add city to tour
V(V==mincity) = [];                    % Remove city from non-tour

% Add edge first-second to TOUR
TOUR(first,mincity)=c(first,mincity);
TOUR(mincity,first)=TOUR(first,mincity);
```

```
% Add cities to tour path
tourTSPtemp=[first,mincity,first];

%% Step 2: Add cities to tour

% Repeat until all cities included in tour
while length(V)~=0

% Step 2.1: Find farthest city from V (non-tour) to T (tour)

maxvalue=0; % set current maxvalue (distance) to minimum

% Repeart for all cties not in tour
for i=V
    maxvaluetemp=max(c(i,T)); % Find farthest city i not in tour to cities in tour
    if(maxvaluetemp>maxvalue) % If distance exceeds previous maximum distance
        maxvalue=maxvaluetemp;% Set current distance as new max distance
        k = i;                % Set current city i as current farthest city to
    end                       % be inserted
end

% Step 2.2: Find existing edge (i,j) in tour to be removed
increase=maxcost; % Set sarting increase cost to max
% Check for all possible combinations i and j of cities in tour
for counter1=1:(length(T)-1)
    i=T(counter1);
    for counter2=(counter1+1):length(T)
        j=T(counter2);
        if(TOUR(i,j)~=0) %check existing edge
            % compute insertion cost of adding k between i and j
            increasetemp=c(i,k)+c(j,k)-c(i,j);
            if(increasetemp<increase) % if insertion cost is less than previous one
                increase=increasetemp;% set as current one as new insertion cost
                i_cut=i; % set current edge (i,j) to be cut
                j_cut=j;
            end
        end
    end
end

% Step 2.3: Replace exiting edge by new edges

if(length(T)==2) % if only 2 cities in tour

% add edges (i,k) and (j,k)
TOUR(i_cut,k)=c(i_cut,k);
TOUR(k,i_cut)=TOUR(i_cut,k);
TOUR(j_cut,k)=c(j_cut,k);
TOUR(k,j_cut)=TOUR(j_cut,k);
```

```
else  % for more than 2 cities in tour
% insert city k into tour by replacing edge (i,j) with (i,k) and (j,k)
% remove edge (i,j)
TOUR(i_cut,j_cut)=0;
TOUR(j_cut,i_cut)=0;
% add edges (i,k) and (j,k)
TOUR(i_cut,k)=c(i_cut,k);
TOUR(k,i_cut)=TOUR(i_cut,k);
TOUR(j_cut,k)=c(j_cut,k);
TOUR(k,j_cut)=TOUR(j_cut,k);
end

% update city k to T and V
T(length(T)+1)=k; % Add city to tour
V(V==k) = []; % Remove city from non-tour

% update tour by adding k in corresponding location
tourTSPtemp=insertToTour(i_cut,j_cut,k,tourTSPtemp);
end

tElapsedtemp=toc(tstart);  % End computation time and store elapsed time

%% Determine best tour so far
lengthTSPtemp=sum(sum(TOUR))/2; % sum all edges in tour matrix and divide by 2
                                % (sum of edges in tour = tour distance)

if(lengthTSPtemp<lengthTSP)  % if current TSP length is less than previous
    lengthTSP=lengthTSPtemp; % set current TSP length as new one
    tourTSP=tourTSPtemp;     % set current TSP tour as new one
    tElapsed=tElapsedtemp;   % set current elapsed time as new one

end


end

%% Display Results (for 150 cities)
sprintf('%s\n',...
    '--Best Tour found by Farthest Insertion Heuristic--',...
    sprintf('Computation TIme for heuristic starting at city %d: %f seconds',...
             tourTSP(1),tElapsed),...
    sprintf('Tour Length: %g km',lengthTSP),...
    'Tour Path: ',...
    sprintf('%g-',tourTSP(1:n/6*1)),...
    sprintf('%g-',tourTSP(n/6*1+1:n/6*2)),...
    sprintf('%g-',tourTSP(n/6*2+1:n/6*3)),...
    sprintf('%g-',tourTSP(n/6*3+1:n/6*4)),...
    sprintf('%g-',tourTSP(n/6*4+1:n/6*5)),...
    strcat(sprintf('%g-',tourTSP(n/6*5+1:n/6*6)),sprintf('%g',tourTSP(n+1))))
```

6.E plotTSP.m

```
%%  Function: plotTSP

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% File Name: plotTSP.m
% Date: 3/27/12
% Author: Luis S Lin
% Description: Plot the TSP tour for cities in the US
%
% Inputs: 1) latitude (N) and longitudes (W) of US cities
%            (absolute values in matrix form, order by ascending order
%             of city number)
%         2) order of tour by city number (array as column vector)
%         3) tour length (scalar value in km units)
%
% Output: US map with TSP tour and tour length displayed
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%% Function

function[tourMAP] = plotTSP(coords,tourTSP,lengthTSP)

%%  Reorder coordinates by tour
n=length(tourTSP);
coordsTSP=zeros(n,2);
for i=1:n
    coordsTSP(i,1)= coords(tourTSP(i),1);
    coordsTSP(i,2)= coords(tourTSP(i),2);
end

% Add - sign for W coordinate
coordsTSP(:,2)=-coordsTSP(:,2);

%% Build a point geostruct

% The first field by convention is Geometry (dimensionality).
% As Geometry is the same for all elements, assign it with deal:
[Cities(1:n).Geometry] = deal('Point');

% Add the latitudes and longitudes to the geostruct:
for i=1:n
    Cities(i).Lat = coordsTSP(i,1);
    Cities(i).Lon = coordsTSP(i,2);
    Cities(i).Name = i;
end

%% Plot US Map
figure; ax = usamap('conus');
states = shaperead('usastatelo', 'UseGeoCoords', true,...
  'Selector',...
  {@(name) ~any(strcmp(name,{'Alaska','Hawaii'})), 'Name'});
```

```
geoshow(ax, states, 'FaceColor', [1 1 0.6])
framem off; gridm off; mlabel off; plabel off

%% Plot points
% Map the City locations with filled circular markers
geoshow(Cities,'Marker','o',...
    'MarkerFaceColor','b','MarkerEdgeColor','b','MarkerSize',5);

%% Plot connections
% Line Type: Great Circle
[lttrk,lntrk] = track('gc',coordsTSP,'degrees');
geoshow(lttrk,lntrk,'DisplayType','line','color','r','LineWidth',2)
% Display tour length
str = sprintf('Tour Length: %g km',lengthTSP);
title(str,'fonts',12)
```

6.F CEE512_hw5.m

```
%% CEE 512 - Homework 5

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% File Name: CEE512_hw5.m
% Date: 3/27/12
% Author: Luis S Lin
% Description: immplement the farthest insertion heuristic to solve for
%              a TSP tour given the latitude and longitdue of the 150
%              largest cities in the US
%
% Files used:     1) costmatrixTSP.m
%                    1.1) findDistance.m
%                 2) findTSP.m
%                    2.1) insertToTour.m
%                 3) plotTSP.m
%
% Inputs: data150cities.xls : excel data file containing latitude and
%         longitude (absolute value, so N and W assumed) of the largest
%         150 cities in the US
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

disp('File Name: CEE512_hw5.m')
disp('Date: 3/27/12')
disp('Author: Luis S Lin')
disp('Description: TSP solution for 150 cities using farthest insertion heuristic')
disp('Inputs: data150cities.xls ')

%% Set folder path wehre all data files located
tic
folderPath='E:\2011-2012 ~ UIUC\SPRING 2012\CEE 512 - Logistics Systems↙
Analysis\Homework\Hw5';
path(path,folderPath);

%% Read data file
filePath ='E:\2011-2012 ~ UIUC\SPRING 2012\CEE 512 - Logistics Systems↙
Analysis\Homework\Hw5\data150cities.xls';
coords = xlsread(filePath,'B4:C153');

%% Create a distance/cost matrix
c=costmatrixTSP(coords);

%% Find TSP
[tourTSP,lengthTSP]=findTSP(c);

%% Plot TSP
plotTSP(coords,tourTSP,lengthTSP)
disp('Computer Platform: HP Z210, Intel Core i5-2400 3.1Ghz Processors, 8GB of RAM')
disp('Software: 7.12.0.635 (R2011a) , 64-bit (win64)')
disp('Total computation time for program CEE512_hw5.m')
toc
```

6.G Resources and References

TSP heuristic

http://blogs.mathworks.com/pick/2011/10/14/traveling-salesman-problem-genetic-algorithm/

http://www.dabi.temple.edu/~vucetic/cis3223spring2008/lab3.htm

http://www.mathworks.com/matlabcentral/fileexchange/13680

http://tsp.r-forge.r-project.org

Map/Plot

http://www.mathworks.com/help/toolbox/map/f20-15124.html#bsgssd6-3

http://www.mathworks.com/help/toolbox/map/ref/usamap.html

http://www.mathworks.com/help/toolbox/map/f1-6030.html

http://www.mathworks.com/matlabcentral/fileexchange/24134-gaimc-graph-algorithms-in-matlab-code/content/gaimc/demo/html/airports.html

http://www.mathworks.com/matlabcentral/fileexchange/35178-tspsearch/content/tspplot.m

http://www.mathworks.com/matlabcentral/fileexchange/33448-ant-system-tsp-solver/content/plot_path.m

http://www.mathworks.com/help/toolbox/map/ref/usamap.html

http://stackoverflow.com/questions/5804468/matlab-drawing-network-of-nodes-in-circular-formation-with-links-between-nodes/5806123#5806123

http://www.mathworks.com/help/techdoc/ref/gplot.html

http://www.mathworks.com/help/techdoc/data_analysis/bridg3r-1.html

http://stackoverflow.com/questions/7824778/plot-arrowhead-on-the-line-between-two-geospatial-points-on-matlab

http://www.mathworks.com/help/toolbox/map/ref/track.html