MECS 4510

HOMEWORK 1

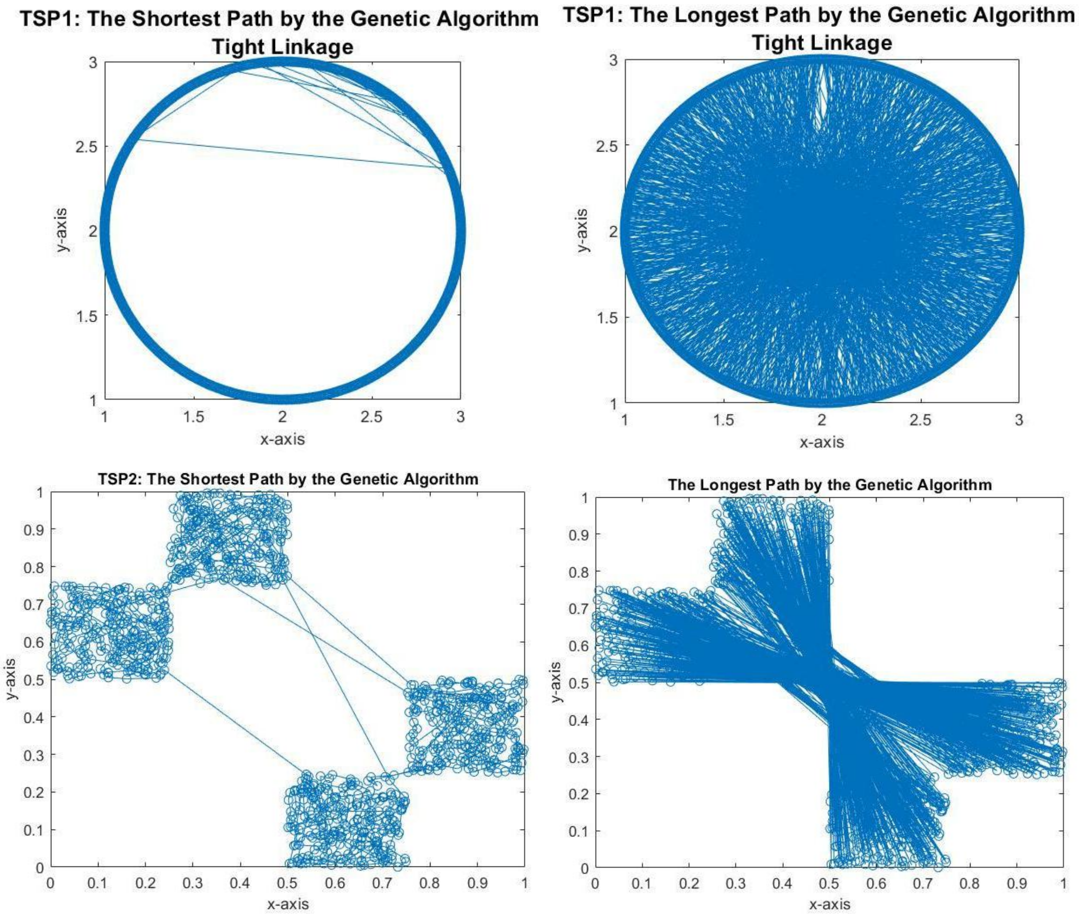
Name: Zhengdong Liu, Jianfei Pan
UNI: zl2957, jp4201
Course Name: Evolutionary Computation and Design Automation
Instructor: Hod Lipson
Date Submitted: 2021-10-04
Grace Hours Used: 0
Grace Hours Gained: 2
Grace Hours Remaining: 98 for each

Section 1: Result summary

Table 1. Result Summary of the Best Overall Performance

	Type	Distance	Evaluation	Population
Ga_TightLinkage_TSP1	Shortest Path	21.6828	E06	10
Ga_TightLinkage_TSP1	Longest Path	1.9694e+03	E06	10
Ga_TightLinkage_TSP2	Shortest Path	27.0531	E07	30
Ga_50%2points_TSP2	Longest Path	798.8363	6894778	30

The team is limited by the computational resource and time and thus we tried to balance our resource and time for completing assignments. Thus, some values here might not be the absolute best value if given more time, but were acquired from our best model running at maximum capacity.



Section 2: Method

2.1. Representation

2.1.1. Random search and hill climber

For random search and hill climber algorithms, the index of each city will be stored in an array with random numbers from 1 to 1000. The index of this array represents the order of the travelling path.

2.1.2. EA

Priority encoding is applied to this travelling salesman problem. Initially, an array of randomly generated priority values will be created. The index of the array represents each city, and will not change during iterations, while the priority values in the array will update after each iteration. The priority array will be used for crossover and mutation operators. Then the travelling path will be determined by the priority value of each city.

2.2. Description

2.2.1. Random search

For random search, a random travelling order will be generated for each iteration, and its total distance will be calculated and compared with the distance of the previous path. The travelling order with shorter distance will be recorded during iterations. Besides, instead of random points, 100 equally distributed points that form a circle are used for the second scheme. Theoretically, the shortest path would be a pure circle that connects all points from beginning to end. Therefore, this scheme can be used to detect if our selection method is capable of finding the best solution.

2.2.2. Hill climber

Instead of creating a random travelling path each time, the method of hill climber will randomly swap two adjacent points in the path array during each iteration. Then a rank based selection of selecting the top 50% shortest or longest traveling sequence will be kept in the population pool, which will be used to repeat the swapping process at the next iteration. The shortest the distance of each run and each iteration is tracked in a list that will be used for plotting in the end.

2.2.3. Beam search

Beam search is a parallel sorting method, where multiple travelling paths will be examined for one iteration. For example, 5 arrays with random path orders will be initially created, then two random adjacent points in each array will be swapped, creating 5 new arrays. Then these arrays will be combined and sorted together, to find the top 50% shortest paths. At last, the newly selected paths will repeat the above procedure to find the shortest path.

2.2.4. Description of EA variations and selection methods

In this assignment, the team used tight linkage in finding the shortest solution, where we used the built in k-means clustering function to organize the random points in two four groups, for which we should maintain their linkages during crossover. We assigned randomly priority weights to different clusters with slightly different average weights to enforce an efficient traveling sequence. We introduced mutation to our population pool that at each evaluation the existing population pool will have offsprings that are a product of random mutation and crossover. The mutation will randomly switch two priority values over the entire 1000 genes. We used 2 point crossover in our model, as we will randomly swap the middle 2 groups in bulk with other genes to create new offsprings. During this process, we will not break the linkages of the previously grouped clusters. For selection methods, the team tried different selection methods and eventually used a rank based selection method, where we will firstly rank their performance based on the total travelled distance and then select the top 50 percent of the population with the best performance to maintain in the pool. These populations will breed the next generation and stay in the evaluation cycle, whereas the bottom 50 percent population will be discarded.

2.2.5. Analysis of performance

The team has tried 1 point, 2 points, and 3 points crossover, and eventually used the 2 point crossover in our final model because it has the best performance. We think 1 point crossover introduces too much bias and is very inefficient and thus creates poor results. We think 3 point crossover introduced too much randomness and thus led to poor convergence. The team tried both poor linkage and tight linkage and found that tight linkage can significantly improve the performance of the model in finding the shortest distance, which makes it converge faster and to the optimal solution. The tight linkage ensures that the cities that are close to each other are kept together during crossover, which as a result reduces the inefficient travelling between sparsely located points. The team also tried different selection methods and to our surprise that the top 25% rank based selection method works better than the 50%. We thought 25% may contain more bias, but it actually led to faster convergence. We suspect that in the long term 50% may converge faster to a more optimal solution but the team currently has limited computation power to prove that. Overall, the team is satisfied with the performance of our models.

2.2.6. Bar Chart comparison of two methods

The below bar chart compares the longest and shortest distance obtained by using a random search genetic algorithm (50% top selection and 2-point crossover) under E05 iterations. It can be seen that the genetic algorithm outperformed random search in finding the shortest distance, which is almost five times smaller than random search. In terms of the longest distance, the method of genetic algorithms is still 200 more than the longest distance found by random search.

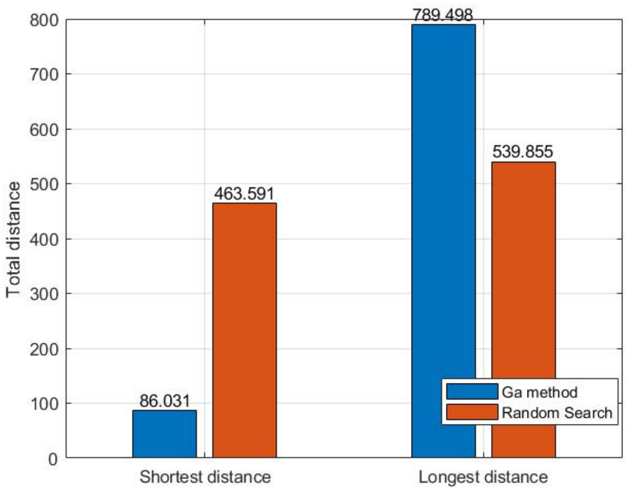


Figure 1. Bar Chart

Section 3: Performance Curves

In this section, the shortest path and longest path learning curves of various methods that we have used are all summarized in the following graph. The fitness for this problem is the total distance traveled as shown in the y axis. The dot plots are also provided as follows.

Table 2 Result summary for all performances

Methods	**Shortest path**	**Longest path**	**Iterations**	**Population**
RS	463.591	539.855	E05	10
Hill Climber	365.184	655.804	E05	10
Beam Search	108.051	774.175	E05	10
Ga_50%2points	98.344	793.260	E05	10
Ga_50%3points	113.625	787.935	E05	10
Ga_25%2points	86.031	789.498	E05	10
Ga_TightLinkage_TSP1	21.6828	N/A	E06	10

3.1. Performance Plots

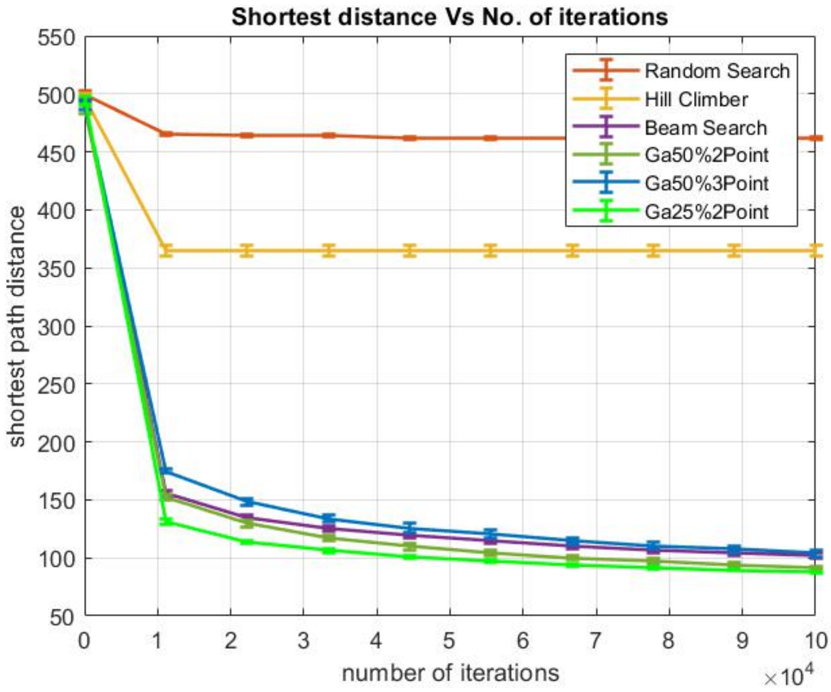


Figure 2. Shortest path learning curve

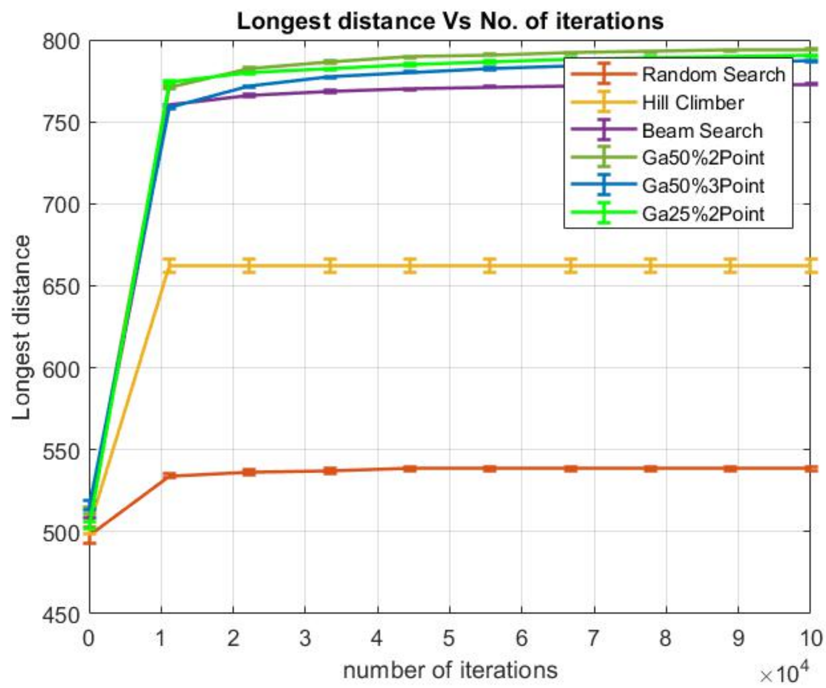


Figure 3. Longest Path Learning Curve

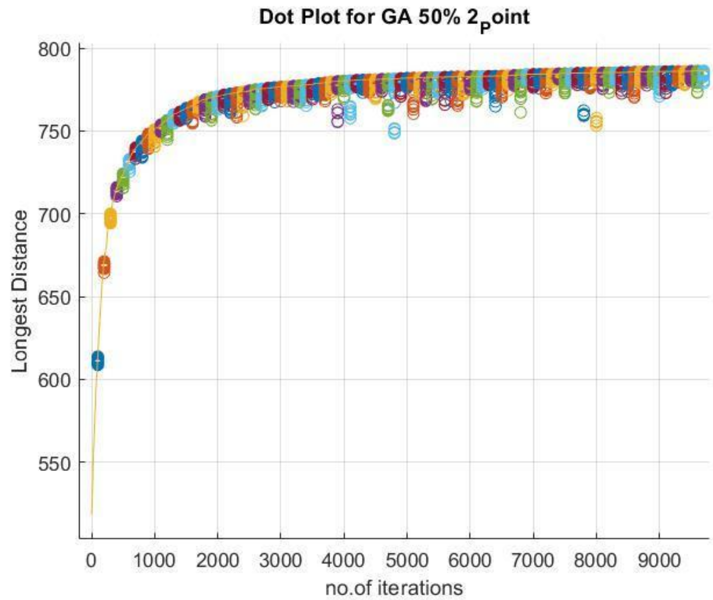


Figure 4. Dot Plots for finding the largest distance

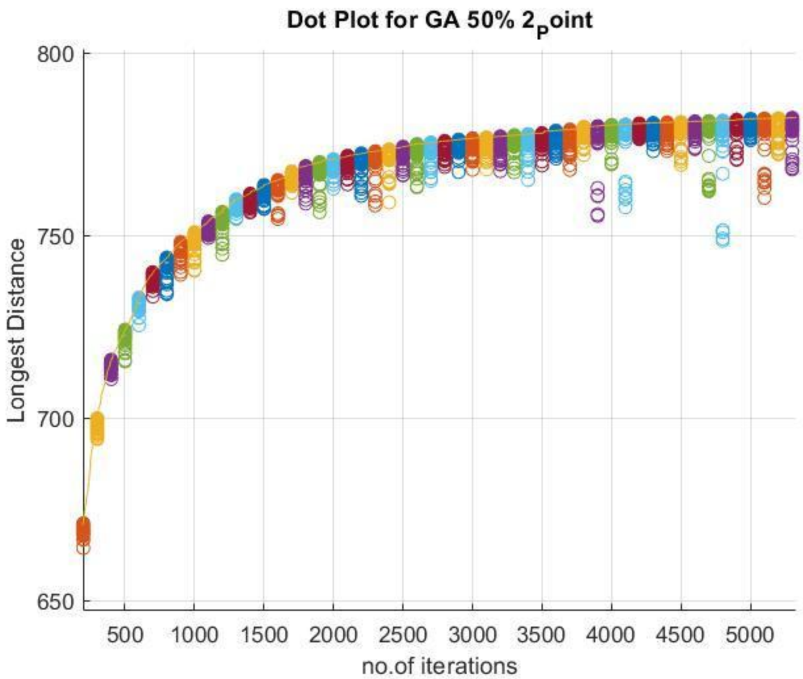


Figure 5. Zoomed-in Dot Plots for finding the largest distance

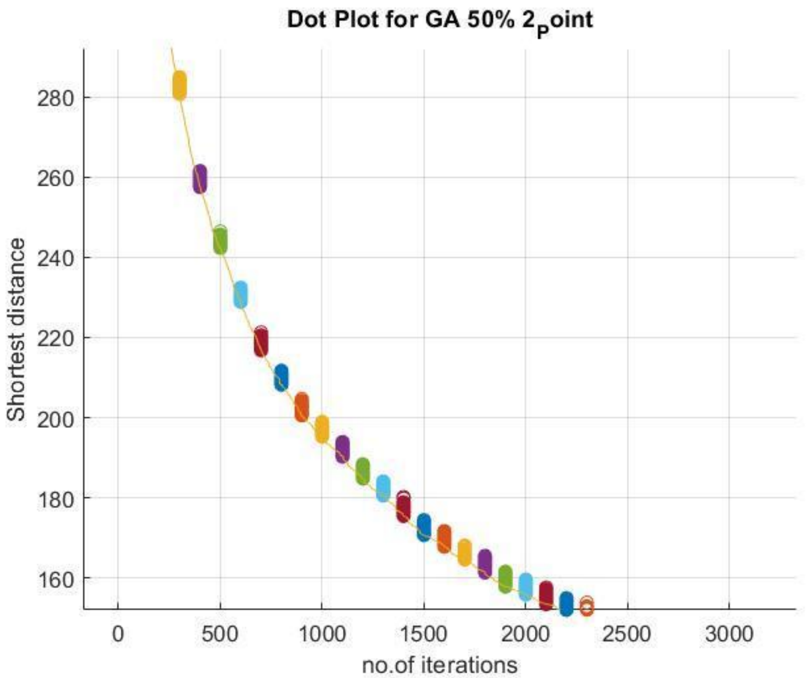


Figure 6. Dot Plots for finding the shortest distance

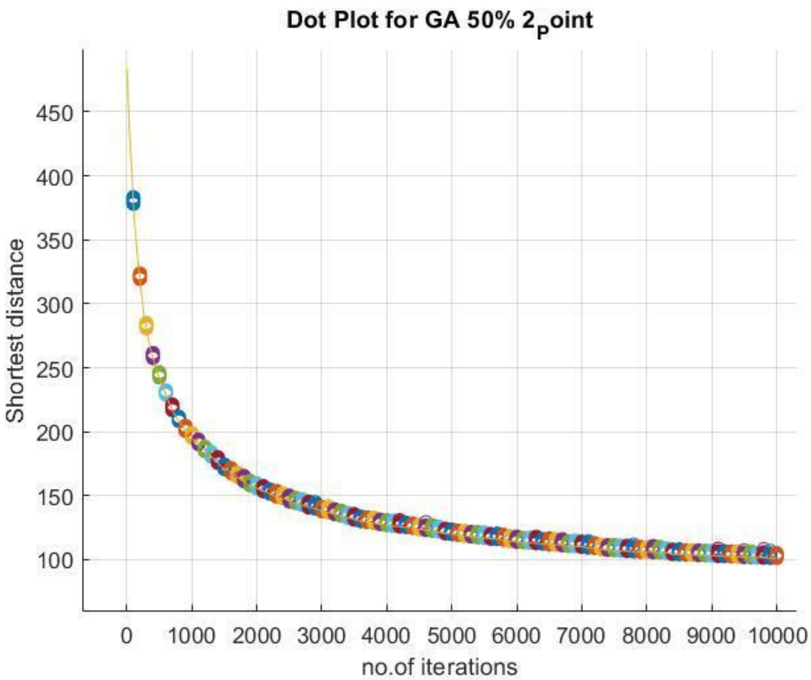


Figure 7. Zoomed-in Dot Plots for finding the shortest distance

3.2. Shortest path plot using the Christofides algorithm

The theoretical shortest path using the Christofides algorithm is plotted below with an estimate for the shortest distance of 15.097. The graph is plotted in python with codes obtained from Retsediv's Github page. [1]

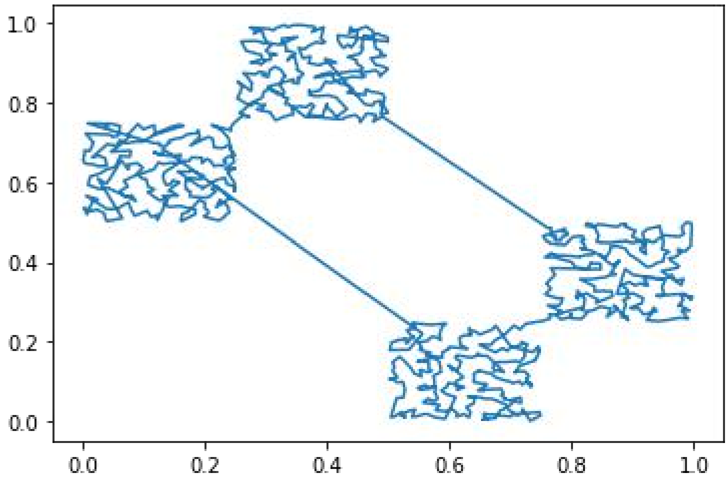


Figure 8. The theoretical shortest path using the Christofides algorithm

3.3. Convergence Plots

The convergence plots for the genetic algorithm with 50 percent rank based selection and 2 point crossover plus mutation variation in finding the shortest path is found. The threshold value is decided to be 85 in order to yield a more obvious yet less computationally expensive convergence plot.

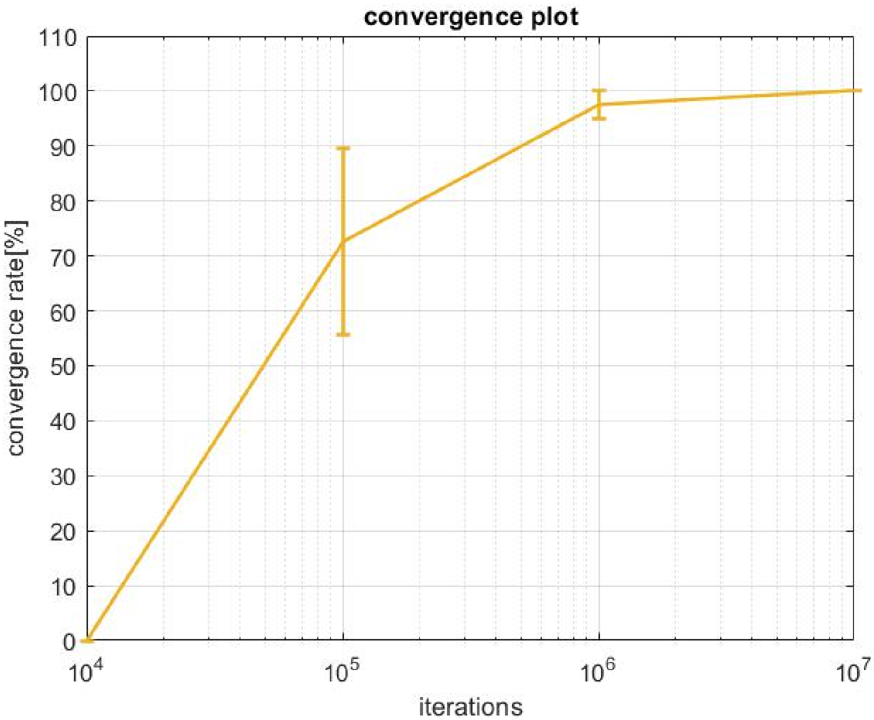


Figure 9. Convergence Plot for finding the longest distance

3.4. Path improvement video

Youtube link:https://youtu.be/-I6Q_B0OWGQ

This video shows the improvement of the shortest path of TSP by using a genetic algorithm. A combination of mutation and 2 point crossover is used, with 25% top results selected for the next iteration. The total number of iterations is 10000 for this video.

Citations

[1] Retsediv, “Retsediv/Christofidesalgorithm,” *GitHub*. [Online]. Available: https://github.com/Retsediv/ChristofidesAlgorithm. [Accessed: 05-Oct-2021].

Appendix

```
% MECS 4510 HOMEWORK1
% Author: Zhengdong Liu Jianfei Pan  UNI:zl2957 jp4201
% This script will import the data of 1000 locations and then it will
% implement the random search method to find the shortest path through all
% points.

% INPUT:   run: number of runs                 evl: number of evalutaions
% OUTPUT:  path_x: x coordinate of path,       path_y:y coordinate of path
%          dx: x coordinate for evaluation     dy: shortest distance
%          derr: errorbar

function [path_x, path_y,dx,dy,derr]=RS_ShortestPath(runs,evl)

% import the randomly distributed samples and store them in terms of x and
% y coordinates
Sample=importdata('tsp.txt');
sample_x=Sample(:,1);
sample_y=Sample(:,2);

    % Run for 10 times to get the average
    for k=1:runs

        % set the initial path distance for checks
        dist_i=1e8;

    % loop over 3e6 iterations to find the shortest path
        for j=1:evl
            x1(j)=j;
            % set the initial distance
            dist=0;
            % create an array of the random sequence of the path
            %num=randperm(1000,1000);

            num=randperm(1000,1000);

            % loop over all points
            for i=1:1000
                if i==1000
                    dist=dist+sqrt( (sample_x(num(1000))-
sample_x((num(1))))^2+(sample_y(num(1000))-sample_y((num(1))))^2);
                else
                    % calculate and add up the total distance
                    dist=dist+sqrt( (sample_x(num(i+1))-
sample_x((num(i))))^2+(sample_y(num(i+1))-sample_y((num(i))))^2);
                end
            end
```

```
            % update the shortest distance
            if dist<dist_i
                path_order=num;
                dist_final(j)=dist;
                dist_i=dist;
            else
                dist_final(j)=dist_i;
            end
        end
        disp_finalY(:,k)=dist_final;
    end

        new_y=mean(disp_finalY,2);
        sd=std(disp_finalY,[],2);
        err=sd/sqrt(runs);
        dx=linspace(1,evl,10);
        dy=interp1(x1,new_y,dx);
        derr=interp1(x1,err,dx);

    % Loop over points to plot the path
     for i=1:1001
         if i==1001
             path_x(1001)=(sample_x(num(1)));
             path_y(1001)=(sample_y(num(1)));
         else
             path_x(i)=(sample_x(num(i)));
             path_y(i)=(sample_y(num(i)));
         end
     end
end



% MECS 4510 HOMEWORK1
% Author: Zhengdong Liu Jianfei Pan  UNI:zl2957 jp4201
% This script will import the data of 1000 locations and then it will
% implement the random search method to find the longest path through all
% points.

% INPUT:   run: number of runs                evl: number of evalutaions
% OUTPUT:  path_x: x coordinate of path,      path_y:y coordinate of path
%          dx: x coordinate for evaluation    dy: longest distance
%          derr: errorbar

function [path_x, path_y,dx,dy,derr]=RS_LongestPath(runs,evl)

    % import the randomly distributed samples and store them in terms of x and
    % y coordinates
    Sample=importdata('tsp.txt');
    sample_x=Sample(:,1);
    sample_y=Sample(:,2);

    % Run for 10 times to get the average
    for k=1:runs

        % set the initial path distance for checks
        dist_i=0;

    % loop over 3e6 iterations to find the shortest path
        for j=1:evl
            x1(j)=j;
            % set the initial distance
            dist=0;

            % create an array of the random sequence of the path
            %num=randperm(1000,1000);
            num=randperm(1000,1000);

            % loop over all points
            for i=1:1000
                if i==1000
                    dist=dist+sqrt( (sample_x(num(1000))-
sample_x((num(1))))^2+(sample_y(num(1000))-sample_y((num(1))))^2);
                else
                    % calculate and add up the total distance
                    dist=dist+sqrt( (sample_x(num(i+1))-
sample_x((num(i))))^2+(sample_y(num(i+1))-sample_y((num(i))))^2);
```

```
                end
            end

            % update the longest distance
            if dist>dist_i
                path_order=num;
                dist_final(j)=dist;
                dist_i=dist;
            else
                dist_final(j)=dist_i;
            end
        end
        disp_finalY(:,k)=dist_final;
    end

        new_y=mean(disp_finalY,2);
        sd=std(disp_finalY,[],2);
        err=sd/sqrt(runs);
        dx=linspace(1,evl,10);
        dy=interp1(x1,new_y,dx);
        derr=interp1(x1,err,dx);

    % Loop over points to plot the path
     for i=1:1001
         if i==1001
             path_x(1001)=(sample_x(num(1)));
             path_y(1001)=(sample_y(num(1)));
         else
             path_x(i)=(sample_x(num(i)));
             path_y(i)=(sample_y(num(i)));
         end
     end
end
```

```
% MECS 4510 HOMEWORK1
% Author: Zhengdong Liu Jianfei Pan  UNI:zl2957 jp4201
% This function will import the data of 1000 locations and then it will
% implement the hill climber method to find the shortest path through all
% points.

% INPUT:   run: number of runs                evl: number of evalutaions
% OUTPUT:  path_x: x coordinate of path,      path_y:y coordinate of path
%          dx: x coordinate for evaluation    dy: shortest distance
%          derr: errorbar

function [path_x,path_y,dx,dy,derr,dist_final]=HillClimber(runs,evl)
    % import the randomly distributed samples and store them in terms of x and
    % y coordinates
    Sample=importdata('tsp.txt');
    sample_x=Sample(:,1);
    sample_y=Sample(:,2);
    for k =1: runs

        % create the initial order of path
        num=randperm(1000,1000);

        % calculate the intial distance
        dist_i=0;
        for i=1:1000
             if i==1000
                 dist_i=dist_i+sqrt( (sample_x(num(1000))-
sample_x((num(1))))^2+(sample_y(num(1000))-sample_y((num(1))))^2);
             else
                 dist_i=dist_i+sqrt( (sample_x(num(i+1))-
sample_x((num(i))))^2+(sample_y(num(i+1))-sample_y((num(i))))^2);
             end
        end

        % loop over n evaluations to improve the result
        for j=1:evl
            % store the data for x coordinate
            x1(j)=j;

            % copy a new order array and randomly swap two adjacent cities
            mutate =num;
            swapidx=randperm(999,1); %Create random indices for swapping
```

```
            mutate(swapidx+1)=num(swapidx); % random swapping
            mutate(swapidx)=num(swapidx+1);
            dist=0;
            % loop over all points, calculate the new distance
            for i=1:1000
                if i==1000
                    dist=dist+sqrt( (sample_x(mutate(1000))-
sample_x((mutate(1))))^2+(sample_y(mutate(1000))-sample_y((mutate(1))))^2);
                else
                    dist=dist+sqrt( (sample_x(mutate(i+1))-
sample_x((mutate(i))))^2+(sample_y(mutate(i+1))-sample_y((mutate(i))))^2);
                end
            end
            % coompare with initial distance and update the shortest order
            if dist<dist_i
                num=mutate;
                dist_final(j,k)=dist;
                dist_i=dist;
            else
                dist_final(j,k)=dist_i(1);
            end
        end
    end

        % calculate the errorbars for these runs
        new_y=mean(dist_final,2);
        sd=std(dist_final,[],2);
        err=sd/sqrt(k);
        dx=linspace(1,evl,10);
        dy=interp1(x1,new_y,dx);
        derr=interp1(x1,err,dx);

    % Loop over points to plot the path
     for i=1:1001
         if i==1001
             path_x(1001)=(sample_x(num(1,1)));
             path_y(1001)=(sample_y(num(1,1)));
         else
             path_x(i)=(sample_x(num(1,i)));
             path_y(i)=(sample_y(num(1,i)));
         end
     end

end


% MECS 4510 HOMEWORK1
% Author: Zhengdong Liu Jianfei Pan  UNI:zl2957 jp4201
% This function will import the data of 1000 locations and then it will
% implement the hill climber method to find the longest path through all
% points.

% INPUT:   run: number of runs                  evl: number of evalutaions
% OUTPUT:  path_x: x coordinate of path,        path_y:y coordinate of path
%          dx: x coordinate for evaluation      dy: longest distance
%          derr: errorbar

function [path_x,path_y,dx,dy,derr,dist_final]=HillClimber_longest(runs,evl)
    % import the randomly distributed samples and store them in terms of x and
    % y coordinates
    Sample=importdata('tsp.txt');
    sample_x=Sample(:,1);
    sample_y=Sample(:,2);
    for k =1: runs

        % create the initial order of path
        num=randperm(1000,1000);

        % calculate the intial distance
        dist_i=0;
        for i=1:1000
             if i==1000
                 dist_i=dist_i+sqrt( (sample_x(num(1000))-
sample_x((num(1))))^2+(sample_y(num(1000))-sample_y((num(1))))^2);
             else
                 dist_i=dist_i+sqrt( (sample_x(num(i+1))-
sample_x((num(i))))^2+(sample_y(num(i+1))-sample_y((num(i))))^2);
```

```
            end
        end

        % loop over n evaluations to improve the result
        for j=1:evl
            % store the data for x coordinate
            x1(j)=j;

            % copy a new order array and randomly swap two adjacent cities
            mutate =num;
            swapidx=randperm(999,1); %Create random indices for swapping
            mutate(swapidx+1)=num(swapidx); % random swapping
            mutate(swapidx)=num(swapidx+1);
            dist=0;
            % loop over all points, calculate the new distance
            for i=1:1000
                if i==1000
                    dist=dist+sqrt( (sample_x(mutate(1000))-
sample_x((mutate(1))))^2+(sample_y(mutate(1000))-sample_y((mutate(1))))^2);
                else
                    dist=dist+sqrt( (sample_x(mutate(i+1))-
sample_x((mutate(i))))^2+(sample_y(mutate(i+1))-sample_y((mutate(i))))^2);
                end
            end
            % coompare with initial distance and update the longest order
            if dist>dist_i
                num=mutate;
                dist_final(j,k)=dist;
                dist_i=dist;
            else
                dist_final(j,k)=dist_i(1);
            end
        end
    end

        % calculate the errorbars for these runs
        new_y=mean(dist_final,2);
        sd=std(dist_final,[],2);
        err=sd/sqrt(k);
        dx=linspace(1,evl,10);
        dy=interp1(x1,new_y,dx);
        derr=interp1(x1,err,dx);

    % Loop over points to plot the path
     for i=1:1001
         if i==1001
             path_x(1001)=(sample_x(num(1,1)));
             path_y(1001)=(sample_y(num(1,1)));
         else
             path_x(i)=(sample_x(num(1,i)));
             path_y(i)=(sample_y(num(1,i)));
         end
     end

end


% MECS 4510 HOMEWORK1
% Author: Zhengdong Liu Jianfei Pan  UNI:zl2957 jp4201
% This function will import the data of 1000 locations and then it will
% implement the parallel-climber method to find the shortest path through all
% points.

% INPUT:   run: number of runs                evl: number of evalutaions
% OUTPUT:  path_x: x coordinate of path,      path_y:y coordinate of path
%          dx: x coordinate for evaluation    dy: shortest distance
%          derr: errorbar


function [path_x,path_y,dx,dy,derr]=BeamSearch(runs,evl)
    % import the randomly distributed samples and store them in terms of x and
    % y coordinates
    Sample=importdata('tsp.txt');
    sample_x=Sample(:,1);
    sample_y=Sample(:,2);

    % set the initial path distance for checks
```

```
    num_new = zeros(5,1000);
    num = zeros(10,1000);

    % create five random travelling sequence


    for k =1: runs
        % generate the five random path sequences
        for n=1:5
        num(n,:)=randperm(1000,1000);
        end
            % loop over n evaluations to improve the result
            for j=1:evl
                % store the data for x coordinate
                x1(j)=j;
                % Mutate the five sequences and add them into the total population
                % pool. This occurs iteratively.
                % make a copy of the first five sequences and mutate them afterwards
                mutate =num(1:5,:);
                num(6:10,:)=mutate;

                for n=1:5
                    swapidx=randperm(1000,2); %Create random indices for swapping
                    num(n+5,swapidx(1))=num(n,swapidx(2)); % random swapping
                    num(n+5,swapidx(2))=num(n,swapidx(1));
                end

                for m=1:10
                    dist=0;
                    % loop over all points, calculate and add up the total distance,
                    % and store them in dist_final
                    for i=1:1000
                        if i==1000
                            dist=dist+sqrt( (sample_x(num(m,1000))-
sample_x((num(m,1))))^2+(sample_y(num(m,1000))-sample_y((num(m,1))))^2);
                        else
                            dist=dist+sqrt( (sample_x(num(m,i+1))-
sample_x((num(m,i))))^2+(sample_y(num(m,i+1))-sample_y((num(m,i))))^2);
                        end
                    end
                    % update the shortest distance

                    dist_final(m)=dist;
                end
                rank_dist=sort(dist_final); % return the smallest value each row
                for n=1:5
                    for l=1:10
                        if rank_dist(n)==dist_final(l)
                            num_new(n,:)=num(l,:); % select top 5 smallest dist
                        end
                    end

                end
                num(1:5,:)=num_new; % update the sequence

                dist_finalNew(j)=rank_dist(1); % store the shortest value
            end
            dist_finalY(:,k)= dist_finalNew; % store thoe shortesst values  for each
run
    end
        % calculate the errorbars for these runs
        new_y=mean(dist_finalY,2);
        sd=std(dist_finalY,[],2);
        err=sd/sqrt(k);
        dx=linspace(1,evl,10);
        dy=interp1(x1,new_y,dx);
        derr=interp1(x1,err,dx);

    % Loop over points to plot the path
     for i=1:1001
         if i==1001
             path_x(1001)=(sample_x(num(1,1)));
             path_y(1001)=(sample_y(num(1,1)));
         else
             path_x(i)=(sample_x(num(1,i)));
             path_y(i)=(sample_y(num(1,i)));
         end
```

```
     end
     
end


% MECS 4510 HOMEWORK1
% Author: Zhengdong Liu Jianfei Pan  UNI:zl2957 jp4201
% This function will import the data of 1000 locations and then it will
% implement the parallel-climber method to find the longest path through all
% points.

% INPUT:   run: number of runs                 evl: number of evalutaions
% OUTPUT:  path_x: x coordinate of path,       path_y:y coordinate of path
%          dx: x coordinate for evaluation     dy: longest distance


function [path_x,path_y,dx,dy,derr]=BeamSearch_long(runs,evl)
    % import the randomly distributed samples and store them in terms of x and
    % y coordinates
    Sample=importdata('tsp.txt');
    sample_x=Sample(:,1);
    sample_y=Sample(:,2);
    
    % set the initial path distance for checks
    num_new = zeros(5,1000);
    num = zeros(10,1000);
    
    % create five random travelling sequence
    
    
    for k =1: runs
        % generate the five random path sequences
        for n=1:5
        num(n,:)=randperm(1000,1000);
        end
            % loop over n evaluations to improve the result
            for j=1:evl
                % store the data for x coordinate
                x1(j)=j;
                % Mutate the five sequences and add them into the total population
                % pool. This occurs iteratively.
                % make a copy of the first five sequences and mutate them afterwards
                mutate =num(1:5,:);
                num(6:10,:)=mutate;
                
                for n=1:5
                    swapidx=randperm(1000,2); %Create random indices for swapping
                    num(n+5,swapidx(1))=num(n,swapidx(2)); % random swapping
                    num(n+5,swapidx(2))=num(n,swapidx(1));
                end
                
                for m=1:10
                    dist=0;
                    % loop over all points, calculate and add up the total distance,
                    % and store them in dist_final
                    for i=1:1000
                        if i==1000
                            dist=dist+sqrt( (sample_x(num(m,1000))-
sample_x((num(m,1))))^2+(sample_y(num(m,1000))-sample_y((num(m,1))))^2);
                        else
                            dist=dist+sqrt( (sample_x(num(m,i+1))-
sample_x((num(m,i))))^2+(sample_y(num(m,i+1))-sample_y((num(m,i))))^2);
                        end
                    end
                    % update the longest distance
                    
                    dist_final(m)=dist;
                end
                rank_dist=sort(dist_final,'descend'); % return the largest value each
row
                for n=1:5
                    for l=1:10
                        if rank_dist(n)==dist_final(l)
                            num_new(n,:)=num(l,:); % select top 5 largest dist
                        end
                    end
```

```
                end
                num(1:5,:)=num_new; % update the sequence

                dist_finalNew(j)=rank_dist(1); % store the largest value
            end
            dist_finalY(:,k)= dist_finalNew; % store thoe largest values  for each run
    end
        % calculate the errorbars for these runs
        new_y=mean(dist_finalY,2);
        sd=std(dist_finalY,[],2);
        err=sd/sqrt(k);
        dx=linspace(1,evl,10);
        dy=interp1(x1,new_y,dx);
        derr=interp1(x1,err,dx);

    % Loop over points to plot the path
     for i=1:1001
         if i==1001
             path_x(1001)=(sample_x(num(1,1)));
             path_y(1001)=(sample_y(num(1,1)));
         else
             path_x(i)=(sample_x(num(1,i)));
             path_y(i)=(sample_y(num(1,i)));
         end
     end

end


% MECS 4510 HOMEWORK1
% Author: Zhengdong Liu Jianfei Pan  UNI:zl2957 jp4201
% This script will import the data of 1000 locations and then it will
% implement the G method(top 50% selection) to find the longest path
% through all points


% INPUT:   run: number of runs                evl: number of evalutaions
%          popluation_size: population size   k_point: crossover variation
%          scheme: 1 for shortest distance, 2 for longest distance
% OUTPUT:  path_x: x coordinate of path,      path_y:y coordinate of path
%          dx: x coordinate for evaluation    dy:  distance
%          derr: errorbar

function [path_x, path_y,dx,dy,derr]=Ea_50(runs,evl,population_size,scheme,k_point)

    % import the randomly distributed samples and store them in terms of x and
    % y coordinates

    Sample=importdata('tsp.txt');
    sample_x=Sample(:,1);
    sample_y=Sample(:,2);



    % set the initial path distance for checks
    num_new = zeros(population_size/2,1000);
    num = zeros(population_size,1000);
    priorities_of_cities = zeros(population_size,1000);

    for k =1: runs

        % NEW: Add weights to all cities and initialize them with random
        % priorities and normalize them. Each priority value corresponds to the
        % city in the smae canonical order. High priority means the particular city
        % will be visited earlier than the city with low priority.

        for n=1:population_size/2
        priorities_of_cities(n,:) = randperm(1000,1000);
        priorities_of_cities(n,:) =
priorities_of_cities(n,:)/max(priorities_of_cities(n,:)) ;% Normalize
        %num(n,:)=randperm(1000,1000);
        end

            % loop over n evaluations to improve the result
            for j=1:evl
                % store the data for x coordinate
                x1(j)=j;
```

```
                % Recombination
                mutate =priorities_of_cities(1:population_size/2,:);
                priorities_of_cities(population_size/2 +1:population_size,:)=mutate;
                % Mutation Starts
                % Mutate the five sequences and add them into the total population
                % pool. This occurs iteratively.
                % make a copy of the first five sequences and mutate them afterwards
                for n=1:population_size/2
                    swapidx=randperm(1000,2); %Create random indices for swapping

priorities_of_cities(n+5,swapidx(1))=priorities_of_cities(n,swapidx(2)); % random
swapping

priorities_of_cities(n+5,swapidx(2))=priorities_of_cities(n,swapidx(1));
                end
                %Mutation Ends
                %Crossover starts, here we use three-point crossover on the priorities
to generate new travel plans. Using
                %three-point is because we identify that the cities are from four
major regions.

                if k_point == 2
                [priorities_of_cities(population_size/2 + 1 :population_size,:)] =
cross_over_and_recombined_mutate_cross_2p(priorities_of_cities(population_size/2 +
1 :population_size,:), k_point);
                elseif k_point==3
                [priorities_of_cities(population_size/2 + 1 :population_size,:)] =
cross_over_and_recombined_mutate_cross_3p(priorities_of_cities(population_size/2 +
1 :population_size,:), k_point);
                end
                % Find the travel plan (an 10*1000 array of indices) based on
                % the priorities, get_travelPlan returns the indices that
                % determine the traversing sequence
                [travel_plan] = get_travelPlan(priorities_of_cities);

                for m=1:population_size
                    dist=0;
                    % loop over all points, calculate and add up the total distance,
                    % and store them in dist_final
                    for i=1:1000
                        if i==1000
                            dist=dist+sqrt( (sample_x(travel_plan(m,1000))-
sample_x((travel_plan(m,1))))^2+(sample_y(travel_plan(m,1000))-
sample_y((travel_plan(m,1))))^2);
                        else
                            dist=dist+sqrt( (sample_x(travel_plan(m,i+1))-
sample_x((travel_plan(m,i))))^2+(sample_y(travel_plan(m,i+1))-
sample_y((travel_plan(m,i))))^2);
                        end
                    end
                    % update the shortest distance

                    dist_final(m)=dist;
                end
                if scheme==1
                [rank_dist,rank_dist_index]=sort(dist_final); % return the smallest
value each row
                else
                [rank_dist,rank_dist_index]=sort(dist_final,'descend');
                end

                %priorities_of_cities
                for n = 1:population_size/2 + 1

priorities_of_cities_selected(n,:)=priorities_of_cities(rank_dist_index(n),:); %
select top 5 smallest dist
                end
                %priorities_of_cities
                priorities_of_cities(1:(population_size/2 +
1),:)=priorities_of_cities_selected; % update the sequence
                dist_finalNew(j)=rank_dist(1); % store the shortest value
            end
            dist_finalY(:,k)= dist_finalNew; % store thoe shortesst values  for each
run
    end
        % calculate the errorbars for these runs
        new_y=mean(dist_finalY,2);
```

```
        sd=std(dist_finalY,[],2);
        err=sd/sqrt(k);
        dx=linspace(1,evl,10);
        dy=interp1(x1,new_y,dx);
        derr=interp1(x1,err,dx);

    % Loop over points to plot the path
     for i=1:1001
         if i==1001
             path_x(1001)=(sample_x(travel_plan(1,1)));
             path_y(1001)=(sample_y(travel_plan(1,1)));
         else
             path_x(i)=(sample_x(travel_plan(1,i)));
             path_y(i)=(sample_y(travel_plan(1,i)));
         end
     end

% plot the shortest path achieved by Parallel Climber
% figure(1)
% plot(path_x,path_y,'-o');
% xlabel('x-axis')
% ylabel('y-axis')
% title('The Shortest Path by Parallel Climber')
end




% MECS 4510 HOMEWORK1
% Author: Zhengdong Liu Jianfei Pan  UNI:zl2957 jp4201
% This script will import the data of 1000 locations and then it will
% implement the G method(top 25% selection) to find the longest path
% through all points


% INPUT:   run: number of runs                evl: number of evalutaions
%          population_size: population size   k_point: crossover variation
%          scheme: 1 for shortest distance, 2 for longest distance
% OUTPUT:  path_x: x coordinate of path,      path_y:y coordinate of path
%          dx: x coordinate for evaluation    dy:  distance
%          derr: errorbar

function [path_x, path_y,dx,dy,derr]=Ea_25(runs,evl,population_size,scheme)


% import the randomly distributed samples and store them in terms of x and
% y coordinates
Sample=importdata('tsp.txt');
sample_x=Sample(:,1);
sample_y=Sample(:,2);
% set the initial path distance for checks
num_new = zeros(population_size/2,1000);
num = zeros(population_size,1000);
priorities_of_cities = zeros(population_size,1000);

    for k =1: runs

        % NEW: Add weights to all cities and initialize them with random
        % priorities and normalize them. Each priority value corresponds to the
        % city in the smae canonical order. High priority means the particular city
        % will be visited earlier than the city with low priority.

        for n=1:population_size*0.2
        priorities_of_cities(n,:) = randperm(1000,1000);
        priorities_of_cities(n,:) =
priorities_of_cities(n,:)/max(priorities_of_cities(n,:)) ;% Normalize
        %num(n,:)=randperm(1000,1000);
        end

            % loop over n evaluations to improve the result
            for j=1:evl
                % store the data for x coordinate
                x1(j)=j;
                % Recombination

                mutate
=[priorities_of_cities(1:population_size*0.2,:);priorities_of_cities(1:population_size
```

```
*0.2,:);priorities_of_cities(1:population_size*0.2,:);priorities_of_cities(1:populatio
n_size*0.2,:)];

                priorities_of_cities(population_size*0.2 +1:population_size,:)=mutate;
                % Mutation Starts
                % Mutate the five sequences and add them into the total population
                % pool. This occurs iteratively.
                % make a copy of the first five sequences and mutate them afterwards
                for n=1:population_size*0.80
                    swapidx=randperm(1000,2); %Create random indices for swapping

priorities_of_cities(n+population_size*0.2,swapidx(1))=priorities_of_cities(n,swapidx(
2)); % random swapping

priorities_of_cities(n+population_size*0.2,swapidx(2))=priorities_of_cities(n,swapidx(
1));
                end
                %Mutation Ends
                %Crossover starts, here we use three-point crossover on the priorities
to generate new travel plans. Using
                %three-point is because we identify that the cities are from four
major regions.

                k_point = 2;
                [priorities_of_cities(population_size*0.80 + 1 :population_size,:)] =
cross_over_and_recombined_mutate_cross_2p(priorities_of_cities(population_size*0.80 +
1 :population_size,:), k_point);

                % Find the travel plan (an 10*1000 array of indices) based on
                % the priorities, get_travelPlan returns the indices that
                % determine the traversing sequence
                [travel_plan] = get_travelPlan(priorities_of_cities);

                for m=1:population_size
                    dist=0;
                    % loop over all points, calculate and add up the total distance,
                    % and store them in dist_final
                    for i=1:1000
                        if i==1000
                            dist=dist+sqrt( (sample_x(travel_plan(m,1000))-
sample_x((travel_plan(m,1))))^2+(sample_y(travel_plan(m,1000))-
sample_y((travel_plan(m,1))))^2);
                        else
                            dist=dist+sqrt( (sample_x(travel_plan(m,i+1))-
sample_x((travel_plan(m,i))))^2+(sample_y(travel_plan(m,i+1))-
sample_y((travel_plan(m,i))))^2);
                        end
                    end
                    % update the shortest distance

                    dist_final(m)=dist;
                end
                if scheme==1
                [rank_dist,rank_dist_index]=sort(dist_final); % return the smallest
value each row
                %priorities_of_cities
                else
                [rank_dist,rank_dist_index]=sort(dist_final,'descend');
                end
                for n = 1:population_size*0.2 + 1

priorities_of_cities_selected(n,:)=priorities_of_cities(rank_dist_index(n),:); %
select top 5 smallest dist
                end
                %priorities_of_cities
                priorities_of_cities(1:(population_size*0.2 +
1),:)=priorities_of_cities_selected; % update the sequence
                dist_finalNew(j)=rank_dist(1); % store the shortest value
            end
            dist_finalY(:,k)= dist_finalNew; % store thoe shortesst values  for each
run
    end

        % calculate the errorbars for these runs
        new_y=mean(dist_finalY,2);
        sd=std(dist_finalY,[],2);
        err=sd/sqrt(k);
```

```
        dx=linspace(1,evl,10);
        dy=interp1(x1,new_y,dx);
        derr=interp1(x1,err,dx);

    % Loop over points to plot the path
     for i=1:1001
         if i==1001
             path_x(1001)=(sample_x(travel_plan(1,1)));
             path_y(1001)=(sample_y(travel_plan(1,1)));
         else
             path_x(i)=(sample_x(travel_plan(1,i)));
             path_y(i)=(sample_y(travel_plan(1,i)));
         end
     end
end

% plot the shortest path achieved by Parallel Climber
% figure(1)
% plot(path_x,path_y,'-o');
% xlabel('x-axis')
% ylabel('y-axis')
% title('The Shortest Path by Parallel Climber')


% MECS 4510 HOMEWORK1
% Author: Zhengdong Liu Jianfei Pan  UNI:zl2957 jp4201
% This function implement the crossover operator and recombine priority
% matrix

% INPUT:   priorities_of_cities:priorities_of_cities
%          k_point:crossover points
% OUTPUT:  priorities_of_cities: new prioirty matrix

function [priorities_of_cities] =
cross_over_and_recombined_mutate_cross_2p(priorities_of_cities, k_point)
%cross_over_and_recombined Summary of this function goes here
% randomly cross over points and then recombine. Crossover can happen among
% two or three parents

number_of_elements = size(priorities_of_cities);
number_of_plans = number_of_elements(1);
cross_over_segments = zeros(number_of_plans,333); % 250 elements for 20 segments

% chopped the inputs into pieces and prepare for the crossover and
% recombination
for i = 1:number_of_plans
    cross_over_segments(i,:) = priorities_of_cities(i,333:665);
end

% randomize the segments and ready to crossover and recombine
cross_over_segments = cross_over_segments(randperm(size(cross_over_segments, 1)), :);

% The next step is to perform cross_over and recombination
random_cross = randperm(number_of_plans);

for i = 1:(number_of_plans*0.7)
    priorities_of_cities(random_cross(i),333:665) =
cross_over_segments(random_cross(i),:);
end
end



% MECS 4510 HOMEWORK1
% Author: Zhengdong Liu Jianfei Pan  UNI:zl2957 jp4201
% This function implement the crossover operator and recombine priority
% matrix

% INPUT:   priorities_of_cities:priorities_of_cities
%          k_point:crossover points
% OUTPUT:  priorities_of_cities: new prioirty matrix


function [priorities_of_cities] =
cross_over_and_recombined_mutate_cross_3p(priorities_of_cities,k_point)
%cross_over_and_recombined Summary of this function goes here
% randomly cross over points and then recombine. Crossover can happen among
```

```
% two or three parents

number_of_elements = size(priorities_of_cities);
number_of_plans = number_of_elements(1);
priorities_of_cities_crossovered = zeros(number_of_plans,1000);
cross_over_segments = zeros(10,250); % 250 elements for 20 segments

count = 1;
for i = 1:2:2*number_of_plans
    cross_over_segments(i,:) = priorities_of_cities(i-(count-1),251:500);
    cross_over_segments(i+1,:) = priorities_of_cities(i-(count-1),501:750);
    count = count + 1;
end

% randomize the segments and ready to crossover and recombine
cross_over_segments = cross_over_segments(randperm(size(cross_over_segments, 1)), :);

random_cross = randperm(5);

for i = 1:(number_of_plans-2)
    priorities_of_cities(random_cross(i),251:500) =
cross_over_segments(2*random_cross(i) -1,:);
    priorities_of_cities(random_cross(i),501:750) =
cross_over_segments(2*random_cross(i),:);
end
    %priorities_of_cities(i,:) =
priorities_of_cities_crossovered(i,:)/max(priorities_of_cities_crossovered(i,:));
end



% MECS 4510 HOMEWORK1
% Author: Zhengdong Liu Jianfei Pan  UNI:zl2957 jp4201
% This function get the lastest travel plan based on priority of cities

% INPUT:   priorities_of_cities:priorities_of_cities

% OUTPUT:  traversing_sequence: travel plan


function [traversing_sequence] = get_travelPlan(priorities_of_cities)
% get_travelPlan Summary of this function goes here
% Detailed explanation goes here
number_of_elements = size(priorities_of_cities);
number_of_plans = number_of_elements(1);
traversing_sequence = zeros(number_of_plans,1000);

for i = 1:number_of_plans

[~,traversing_sequence(i,:)] = sort(priorities_of_cities(i,:),'descend');

end


% MECS 4510 HOMEWORK1
% Author: Zhengdong Liu Jianfei Pan  UNI:zl2957 jp4201
% This script will do :
% 1: plot the shortest path by Ga method (25%_2points)
% 2?Plot the learning curve for shortest distance for following methods:
% random search, hill  climber, beam search, Ga_50%_3points,
% Ga_50%_2points, Ga_25%_2points

% clear workspace and command window
clear;
clc;

% set up the parameters, we run 5 times with a population of 10 and
% iterations of 1e5.
% scheme =2 represents the method finding the longest distance, while
% scheme =1 represents the method finding hte shortest distance
runs=5;
evl=1e5;
population_size=10;
scheme=1;

%plot the learning curve for shortest distance for following methods:
% random search, hill  climber, beam search, Ga_50%_3points,
```

```
% Ga_50%_2points, Ga_25%_2points
figure (1)

[~,~,dxRs,dyRs,derrRs]=RS_ShortestPath(runs,evl);
 output_graphHC= errorbar(dxRs,dyRs,derrRs);
output_graphHC.LineWidth = 1.5;
output_graphHC.Color = '#D95319';
hold on

[~,~,dxHc,dyHc,derrHc]=HillClimber(runs,evl);
 output_graphHC= errorbar(dxHc,dyHc,derrHc);
output_graphHC.LineWidth = 1.5;
output_graphHC.Color = '#EDB120';
hold on

[~,~,dxBs,dyBs,derrBs]=BeamSearch(runs,evl);
 output_graphHC= errorbar(dxBs,dyBs,derrBs);
output_graphHC.LineWidth = 1.5;
output_graphHC.Color = '#7E2F8E';

k_point=2;  % set the crossover variation, this time 2 points crossover
 [~, ~,dxEa50,dyEa50,derrEa50]=Ea_50(runs,evl,population_size,scheme,k_point);
output_graphHC= errorbar(dxEa50,dyEa50,derrEa50);
output_graphHC.LineWidth = 1.5;
output_graphHC.Color = '#77AC30';
hold on

 k_point=3; % set the crossover variation, this time 2 points crossover
  [~,~,dx2Ea50,dy2Ea50,derr2Ea50]=Ea_50(runs,evl,population_size,scheme,k_point);
 output_graphHC= errorbar(dx2Ea50,dy2Ea50,derr2Ea50);
output_graphHC.LineWidth = 1.5;
output_graphHC.Color = '#0072BD';
hold on


[path_xShort,path_yShort,dxEa25,dyEa25,derrEa25]=Ea_25(runs,evl,population_size,scheme)
;
 output_graphHC= errorbar(dxEa25,dyEa25,derrEa25);
output_graphHC.LineWidth = 1.5;
output_graphHC.Color = '#00FF00';
hold on

xlabel('number of iterations')
ylabel('shortest path distance')
title('Shortest distance Vs No. of iterations')
grid on

legend('Random Search','Hill Climber','Beam
Search','Ga50%2Point','Ga50%3Point','Ga25%2Point')

 % figure 2 plots the longest path achieved by Ga_25%_2points
figure (2)
plot(path_xShort,path_yShort,'-o');
xlabel('x axis')
ylabel('y axis')
title('Shortest path')


% MECS 4510 HOMEWORK1
% Author: Zhengdong Liu Jianfei Pan  UNI:zl2957 jp4201
% This script will do :
% 1: plot the longest path by Ga method (25%_2points)
% 2?Plot the learning curve for longest distance for following methods:
% random search, hill  climber, beam search, Ga_50%_3points,
% Ga_50%_2points, Ga_25%_2points


% clear workspace and command window
clear;
clc;

% set up the parameters, we run 5 times with a population of 10 and
% iterations of 1e5.
% scheme =2 represents the method finding the longest distance, while
% scheme =1 represents the method finding hte shortest distance
runs=5;
evl=1e5;
```

```
population_size=10;
scheme=2;

%plot the learning curve for longest distance for following methods:
% random search, hill  climber, beam search, Ga_50%_3points,
% Ga_50%_2points, Ga_25%_2points
figure (1)

[~,~,dxRs,dyRs,derrRs]=RS_LongestPath(runs,evl);
 output_graphHC= errorbar(dxRs,dyRs,derrRs);
output_graphHC.LineWidth = 1.5;
output_graphHC.Color = '#D95319';
hold on

[~,~,dxHc,dyHc,derrHc]=HillClimber_longest(runs,evl);
 output_graphHC= errorbar(dxHc,dyHc,derrHc);
output_graphHC.LineWidth = 1.5;
output_graphHC.Color = '#EDB120';
hold on

[~,~,dxBs,dyBs,derrBs]=BeamSearch_long(runs,evl);
 output_graphHC= errorbar(dxBs,dyBs,derrBs);
output_graphHC.LineWidth = 1.5;
output_graphHC.Color = '#7E2F8E';

k_point=2;   % set the crossover variation, this time 2 points crossover
 [~, ~,dxEa50,dyEa50,derrEa50]=Ea_50(runs,evl,population_size,scheme,k_point);
output_graphHC= errorbar(dxEa50,dyEa50,derrEa50);
output_graphHC.LineWidth = 1.5;
output_graphHC.Color = '#77AC30';
hold on

 k_point=3;  % set the crossover variation, this time 3 points crossover
  [~,~,dx2Ea50,dy2Ea50,derr2Ea50]=Ea_50(runs,evl,population_size,scheme,k_point);
 output_graphHC= errorbar(dx2Ea50,dy2Ea50,derr2Ea50);
output_graphHC.LineWidth = 1.5;
output_graphHC.Color = '#0072BD';
hold on

[path_xLong,path_yLong,dxEa25,dyEa25,derrEa25]=Ea_25(runs,evl,population_size,scheme);
 output_graphHC= errorbar(dxEa25,dyEa25,derrEa25);
output_graphHC.LineWidth = 1.5;
output_graphHC.Color = '#00FF00';
hold on

xlabel('number of iterations')
ylabel('Longest distance')
title('Longest distance Vs No. of iterations')
grid on

 legend('Random Search','Hill Climber','Beam
Search','Ga50%2Point','Ga50%3Point','Ga25%2Point')

 % figure 2 plots the longest path achieved by Ga_25%_2points
figure (2)
plot(path_xLong,path_yLong,'-o');
xlabel('x axis')
ylabel('y axis')
title('Longest path')

 % MECS 4510 HOMEWORK1
% Author: Zhengdong Liu Jianfei Pan  UNI:zl2957 jp4201
% This function will plot the shortest distance path using GA method with
% clustering feature


% import the randomly distributed samples and store them in terms of x and
% y coordinates
clear;
clc;
Sample=importdata('tsp.txt');
sample_x=Sample(:,1);
sample_y=Sample(:,2);
%cluster points and them recombine them, use kmeans clustering with k = 4.
group1_x = [];
group1_y = [];
group2_x = [];
```

```
group2_y = [];
group3_x = [];
group3_y = [];
group4_x = [];
group4_y = [];

[idx, C] = kmeans(Sample,4);
for i = 1:1000
    if idx(i) == 1
        group1_x = [group1_x,sample_x(i)];
        group1_y = [group1_y,sample_y(i)];
    elseif idx(i) == 2
        group2_x = [group2_x, sample_x(i)];
        group2_y = [group2_y, sample_y(i)];
    elseif idx(i) == 3
        group3_x = [group3_x, sample_x(i)];
        group3_y = [group3_y, sample_y(i)];
    else
        group4_x = [group4_x, sample_x(i)];
        group4_y = [group4_y, sample_y(i)];
    end
end

sample_x = [group1_x, group2_x, group3_x, group4_x];
sample_y = [group1_y, group2_y, group3_y, group4_y];
size_of_cities = [size(group1_x),size(group2_x),size(group3_x),size(group4_x)]
k1 = size(group1_x,2);
k2 = size(group1_x,2) + size(group2_x,2) + size(group3_x,2);

% Set some hyperparameters
runs=1;
evl=1e6;
population_size = 50;

% set the initial path distance for checks
num_new = zeros(population_size/2,1000);
num = zeros(population_size,1000);
priorities_of_cities = zeros(population_size,1000);

for k =1: runs

    % NEW: Add weights to all cities and initialize them with random
    % priorities and normalize them. Each priority value corresponds to the
    % city in the smae canonical order. High priority means the particular city
    % will be visited earlier than the city with low priority.

    for n=1:population_size/2
    priorities_of_cities(n,:) = [randperm(1000,250),randperm(750,250),
randperm(500,250), randperm(250,250)];
    priorities_of_cities(n,:) =
priorities_of_cities(n,:)/max(priorities_of_cities(n,:)) ;% Normalize
    %num(n,:)=randperm(1000,1000);
    end

        % loop over n evaluations to improve the result
        for j=1:evl
            % store the data for x coordinate
            x1(j)=j;
            % Recombination
            mutate =priorities_of_cities(1:population_size/2,:);
            priorities_of_cities(population_size/2 +1:population_size,:)=mutate;
            % Mutation Starts
            % Mutate the five sequences and add them into the total population
            % pool. This occurs iteratively.
            % make a copy of the first five sequences and mutate them afterwards
            for n=1:population_size/2
                swapidx=randperm(1000,2); %Create random indices for swapping

priorities_of_cities(n+population_size/2,swapidx(1))=priorities_of_cities(n,swapidx(2))
; % random swapping

priorities_of_cities(n+population_size/2,swapidx(2))=priorities_of_cities(n,swapidx(1))
;
            end
            %Mutation Ends
            %Crossover starts, here we use three-point crossover on the priorities to
generate new travel plans. Using
```

```
            %three-point is because we identify that the cities are from four major
regions.

            k_point = 2;
            [priorities_of_cities(population_size/2 + 1 :population_size,:)] =
cross_over_and_recombined_mutate_cross_2pk(priorities_of_cities(population_size/2 +
1 :population_size,:), k1,k2);

            % Find the travel plan (an 10*1000 array of indices) based on
            % the priorities, get_travelPlan returns the indices that
            % determine the traversing sequence
            [travel_plan] = get_travelPlan(priorities_of_cities);

            for m=1:population_size
                dist=0;
                % loop over all points, calculate and add up the total distance,
                % and store them in dist_final
                for i=1:1000
                    if i==1000
                        dist=dist+sqrt( (sample_x(travel_plan(m,1000))-
sample_x((travel_plan(m,1))))^2+(sample_y(travel_plan(m,1000))-
sample_y((travel_plan(m,1))))^2);
                    else
                        dist=dist+sqrt( (sample_x(travel_plan(m,i+1))-
sample_x((travel_plan(m,i))))^2+(sample_y(travel_plan(m,i+1))-
sample_y((travel_plan(m,i))))^2);
                    end
                end
                % update the shortest distance

                dist_final(m)=dist;
            end

            [rank_dist,rank_dist_index]=sort(dist_final); % return the smallest value
each row
            %priorities_of_cities
            for n = 1:population_size/2 + 1

priorities_of_cities_selected(n,:)=priorities_of_cities(rank_dist_index(n),:); %
select top 5 smallest dist
            end
            %priorities_of_cities
            priorities_of_cities(1:(population_size/2 +
1),:)=priorities_of_cities_selected; % update the sequence
            dist_finalNew(j)=rank_dist(1); % store the shortest value
        end
        dist_finalY(k,:)= dist_finalNew; % store thoe shortesst values  for each run
end


% Loop over points to plot the path
 for i=1:1001
     if i==1001
         path_x(1001)=(sample_x(travel_plan(1,1)));
         path_y(1001)=(sample_y(travel_plan(1,1)));
     else
         path_x(i)=(sample_x(travel_plan(1,i)));
         path_y(i)=(sample_y(travel_plan(1,i)));
     end
 end

% plot the shortest path achieved by Parallel Climber
figure(1)
plot(path_x,path_y,'-o');
xlabel('x-axis')
ylabel('y-axis')
title('The Longest Path by the Genetic Algorithm')

% MECS 4510 HOMEWORK1
% Author: Zhengdong Liu Jianfei Pan  UNI:zl2957 jp4201
% This function will get the upadated travelplan using priority encoding

function [priorities_of_cities] =
cross_over_and_recombined_mutate_cross_2pk(priorities_of_cities,k1,k2)
    %cross_over_and_recombined Summary of this function goes here
    % randomly cross over points and then recombine. Crossover can happen among
    % two or three parents
```

```
    % k1 is the first cutoff point and k2 is the second. The segment between k1
    % and k2 will be used for crossover.

    number_of_elements = size(priorities_of_cities);
    number_of_plans = number_of_elements(1);
    cross_over_segments = zeros(number_of_plans,k2-k1); % middle 2 fourth segments

    % chopped the inputs into pieces and prepare for the crossover and
    % recombination
    for i = 1:number_of_plans
        cross_over_segments(i,:) = priorities_of_cities(i,k1+1:k2);
    end

    % randomize the segments and ready to crossover and recombine
    cross_over_segments = cross_over_segments(randperm(size(cross_over_segments,
1)), :);

    % The next step is to perform cross_over and recombination
    random_cross = randperm(number_of_plans);

    for i = 1:(number_of_plans*0.7)
        priorities_of_cities(random_cross(i),k1+1:k2) =
cross_over_segments(random_cross(i),:);
    end
end
```

```
% MECS 4510 HOMEWORK1
% Author: Zhengdong Liu Jianfei Pan  UNI:zl2957 jp4201
% This function will plot the bar graph comparing the results obtained by
% using Random search and Ga_25%_2points

% clear command window and workspace
clear;
clc;

% import the collected data

vals = [86.031 463.591 ; 789.498 539.855 ];
x = categorical({'Shortest distance','Longest distance'});
x = reordercats(x,{'Shortest distance','Longest distance'});
b = bar(x,vals);




xtips1 = b(1).XEndPoints;
ytips1 = b(1).YEndPoints;
labels1 = string(b(1).YData);
text(xtips1,ytips1,labels1,'HorizontalAlignment','center',...
    'VerticalAlignment','bottom')

xtips2 = b(2).XEndPoints;
ytips2 = b(2).YEndPoints;
labels2 = string(b(2).YData);
text(xtips2,ytips2,labels2,'HorizontalAlignment','center',...
    'VerticalAlignment','bottom')
legend('Ga method','Random Search')
grid on
ylabel('Total distance')
```