پروژه حل مسئله فروشنده دوره گرد بکمک الگوریتم ژنتیک

نویسنده : علیرضا طباطبائیان 9723052

ابتدا توابع Crossover و Mutation را مطابق آنچه در صورت سوال خواسته شده پیاده سازی مینماییم.

Crossover.m

```
function Crossed = Crossover(Root) % Crossover Function Definition

    Number_of_Cities = length( Root ) ; % Find the Number of Cities
    K1 = 0 ; % K1 = The First Random Position
    K2 = 0 ; % K2 = The Second Random Position

    while K1==K2

          K1 = randi([1,Number_of_Cities]) ; % The First Random Position
          K2 = randi([1,Number_of_Cities]) ; % The Second Random Position
    end

    if K1 < K2 % Crossover
       Root(1,K1:K2) = fliplr(Root(1,K1:K2)); % fliplr is the best function for crossover
    else
        Root(1,K2:K1) = fliplr(Root(1,K2:K1));
    end

    Crossed = Root ;

end
```

Mutation.m

```
function Mutant = Mutation(Root) % Define The Mutation function

    Number_of_Cities = length( Root ) ; % Find the Number of Cities
    K1 = 0 ; % K1 = The First Random Position
    K2 = 0 ; % K2 = The Second Random Position

    while K1==K2

          K1 = randi([1,Number_of_Cities]) ; % The First Random Position
          K2 = randi([1,Number_of_Cities]) ; % The Second Random Position

    end

    Root([K1,K2])=Root([K2,K1]); % Mutate = Change The Numbers
    Mutant = Root;

end
```

سپس نقاط تصادفی را تولید کرده و اولین گروه از والدین را بصورت رندوم تولید میکنیم.

Random_Number_Generator.m

```
function Points = Random_Number_Generator( Start_Point , Stop_Point , Number_of_Cities) % Parent Coordinates

    X = ( ( Stop_Point(1) - Start_Point(1) ) * rand( Number_of_Cities , 1 ) ) + Start_Point(1) ; % Define X
    Y = ( ( Stop_Point(2) - Start_Point(2) ) * rand( Number_of_Cities , 1 ) ) + Start_Point(2) ; % Define Y
    Z = ( ( Stop_Point(3) - Start_Point(3) ) * rand( Number_of_Cities , 1 ) ) + Start_Point(3) ; % Define Z
    Points = [ X , Y , Z ] ;

end
```

Main.m

```
%% Clear

close all ; clear ; clc ;  % Close all

%% Initial Parameters Definition

Start_Point = [0,0,0] ; % Define Start Point
Stop_Point  = [3,4,5] ; % Define End Point
Number_of_Cities = 25 ; % Define Number of Cities
Number_of_FirstGeneration = 100 ; % Define The Number of First Generation Parents
Number_of_Iterations = 500 ; % Define Number of Iterations
Points = Random_Number_Generator( Start_Point , Stop_Point , Number_of_Cities) ; % Generate The Cities
Roots = zeros( Number_of_FirstGeneration , Number_of_Cities , Number_of_Iterations ) ; % Define a 3D Vect
Length = zeros( Number_of_FirstGeneration , 1 , Number_of_Iterations ) ; % Define a Vector for Length of

%% Generate the First Generation of Parents

for i = 1 : Number_of_FirstGeneration

    Roots ( i , : , 1 ) = randperm( Number_of_Cities ) ; % A Permutation of Cities
    Length( i , 1 , 1 ) = Distance(Points , Roots ( i , : , 1 )) ; % Find the Length of the Generated Roc

end

[Length( : , 1 , 1 ) , Positions ] = sort(Length( : , 1 , 1 )) ; % Sort the Lengths
Roots ( : , : , 1 ) = Roots ( Positions , : , 1 ) ; % Sort the Roots according to their Lengths
```

سپس آنها را مرتب کرده و اعمال Mutate و Crossover را بر روی آنها اعمال میکنیم :

49 درصد باز تولید

1 درصد جهش

50 درصد کراس اُور

New_Roots_Generation.m

```
function Children = New_Roots_Generation(Parents) % Children Generation function

    [Number_of_Parents , Nember_of_Cities] = size(Parents); % Find the number of parents and cities
    temp = zeros(Number_of_Parents , Nember_of_Cities); % Children temporary variable Definition

    for i = 1 : 0.49*Number_of_Parents % Reproduction
        temp(i,:) = Parents(i,:);
    end

    for i = 0.49*Number_of_Parents+1 : 0.5*Number_of_Parents % Mutation
        temp(i,:) = Mutation(Parents(i-0.49*Number_of_Parents,:)) ;
    end

    for i = 1 : Number_of_Parents/4 % Crossover
        for j = i-1 : i
            temp(0.5*Number_of_Parents+i+j,:) = Crossover(Parents(i,:)) ;
        end
    end

    Children = temp ;
end
```

در نهایت این حلقه مرتب سازی و پیاده سازی الگوریتم های ژنتیک را ادامه میدهیم تا به جواب مطلوب برسیم.

```
%% Generate Children from Parents

figure(1) % Plot the Roots
plot3([0;Points(Roots( 1 , : , 1 ),1);0],[0;Points(Roots( 1 , : , 1 ),2);0],[0;Points(Roots( 1 , : , 1 ),
title('Best Root in each Iteration','color','r');
xlabel('X','color','r');
ylabel('Y','color','r');
zlabel('Z','color','r');

for i = 2 : Number_of_Iterations

    Roots ( : , : , i ) = New_Roots_Generation(Roots( : , : , i-1 )); % Generate new Roots from Parents

    for j = 1 : Number_of_FirstGeneration

        Length( j , 1 , i ) = Distance(Points , Roots ( j , : , i )) ; % Find The Lengths of new Children

    end

    [Length( : , 1 , i ) , Positions ] = sort(Length( : , 1 , i )) ; % Sort the Lengths
    Roots ( : , : , i ) = Roots ( Positions , : , i ) ; % Sort the Roots according to their Lengths
    pause(0.001); % Pause order so as to Have a Better Insight into Plots
    plot3([0;Points(Roots( 1 , : , i ),1);0],[0;Points(Roots( 1 , : , i ),2);0],[0;Points(Roots( 1 , : ,
    title('Best Root in each Iteration','color','r');
    xlabel('X','color','r');
    ylabel('Y','color','r');
    zlabel('Z','color','r');

end

figure(2) % Plot the Best Length answer of each Iteratin
X = 1 : 1 : Number_of_Iterations ;
Y(:) = Length(1,1,:) ;
plot(X , Y);
title('Y : Best Length / X : Iteration','color','b');
xlabel('Iteration','color','r');
ylabel('Best Length','color','r');
```

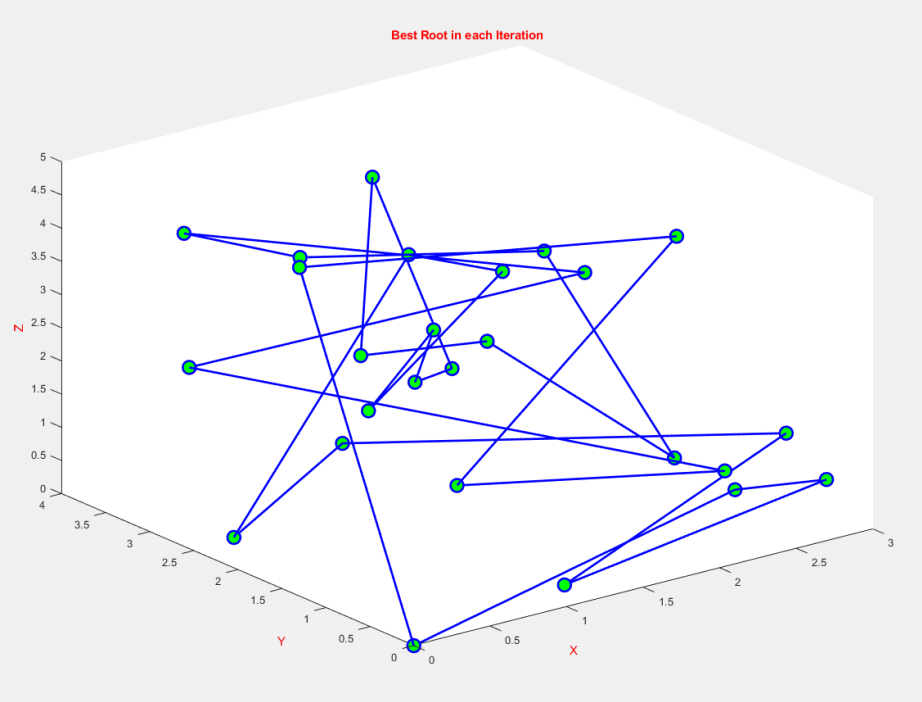
*** در اینجا تابع برازندگی همان تابع فاصله است که در زیر قابل مشاهده است : (هر چقدر طول کمتر ، برازندگی بیشتر) (تابع هزینه هم همان است)

Distance.m

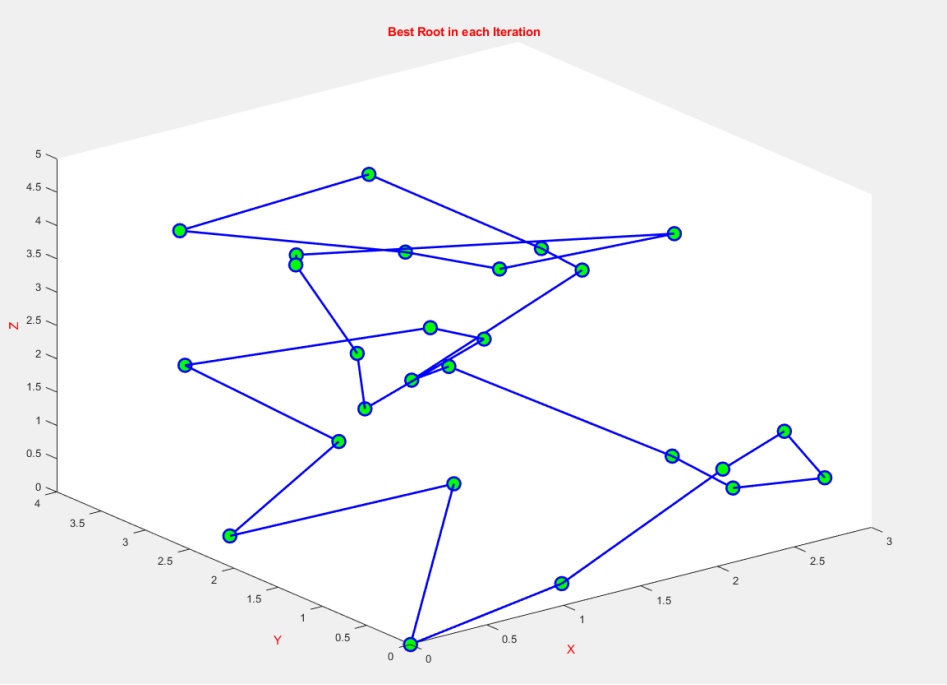
```
function d = Distance(Points , Root) % Length Function

    Number_of_cities = length( Points ) ;
    Length = zeros(1 , Number_of_cities + 1) ;

    Length(1)    = sqrt(sum((Points(Root(1),:)).^2)) ; % Length of the First Point from the Origin

    for i = 2 : Number_of_cities

        Length(i) = sqrt(sum((Points(Root(i),:)-Points(Root(i-1),:)).^2)) ; % Length of the points with e

    end

    Length(Number_of_cities+1) = sqrt(sum((Points(Root(Number_of_cities),:)).^2)) ; % Length of the Last

    d = sum( Length ) ; % Summation of these lengths

end
```

نتایج

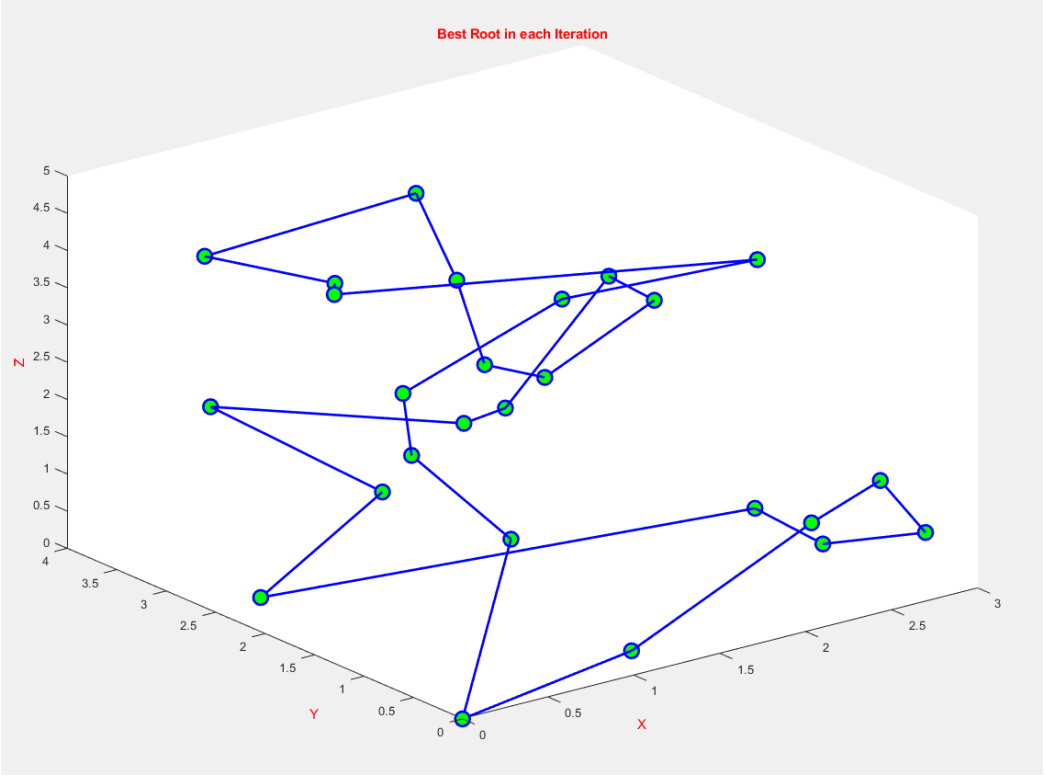
تکرار اول



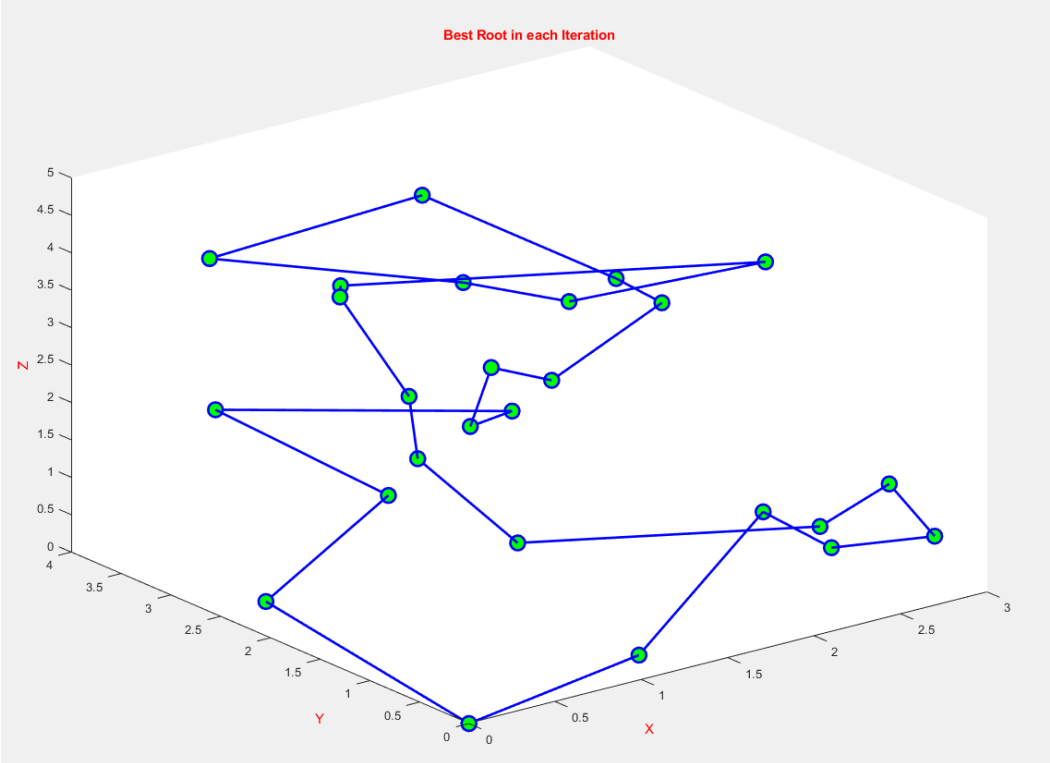
تکرار صدم



تکرار پانصدم



تکرار هزارم



طول بهینه در هر تکرار :