حل مسئله فروشنده دوره گرد با الگوریتم قورباغه sfla در متلب

صورت مسئله:

یک فروشنده دوره گرد تمامی شهرها (اینجا ۲۰ شهر و هزینه پیمایش بین هر جفت از آنها مشخص است) را به گونه ای ملاقات کند که هر یک از این شهرها را فقط یک بارملاقات کرده باشد و دوباره به شهر آغازین برگردد با این شرط که با کمترین هزینه پیمایش این کار را انجام دهد.

به طور کلی هدف پیدا کردن کم هزینه ترین تور برای ملاقات همه شهرها و بازگشت به شهر آغازین حرکت است.

```
    x=[25 76 91 86 47 66 87 50 22 19 3 67 86 52 10 56 21 65
14 88];

    y=[28 1 37 100 57 10 32 56 97 32 27 43 39 89 12 34 79
56 6 21];
```

شرح کد:

این سورس کد شامل ۸ فایل می باشد که عبارتند از:

CreateModel.m: برای ایجاد فاصله و هزینه بین شهرها و مختصات هر یک از شهرها از این تابع استفاده می شود.

```
function model=CreateModel()

    x=[25 76 91 86 47 66 87 50 22 19 3 67 86 52 10 56 21 65
14 88];

    y=[28 1 37 100 57 10 32 56 97 32 27 43 39 89 12 34 79
56 6 21];

    n=numel(x);  تعداد شهرها

    d=zeros(n,n);ماتریس فاصله بین شهرها

    for i=1:n-1
        for j=i+1:n
```

```
            d(i,j)=sqrt((x(i)-x(j))^2+(y(i)-y(j))^2); محاسبه فاصله

            d(j,i)=d(i,j); گذاشتن فاصله بین شهرها به صورت متقارن

        end
    end

    model.n=n;          دادن متغیرهای مسئله به مدل
    model.x=x;
    model.y=y;
    model.d=d;

end
```

Myfitt: این تابع برای محاسبه برازندگی مسیرهای یک تور یا یک پیمایش کامل شهرها بکار می رود.

```
function [z sol]=Myfitt (s,model)

    d=model.d;

    [~, tour]=sort(s);
    sol.tour=tour; تور با هزینه کمتر

    n=numel(tour);

    tour=[tour tour(1)];

    L=0;
    for i=1:n
        L=L+d(tour(i),tour(i+1)); هزینه قبلی+هزینه انتخاب شهرجدید
    end

    sol.L=L;  راه حل

    z=1/(1+L); فیتنس

end
```

SortPopulation.m:مرتب سازی جمعیت قورباغه ها

```
function [pop, SortOrder] = SortPopulation(pop)

    دادن برازندگی
    fits = [pop.fit];

    بردارهای برازندگی را نزولی مرتب کنید
    [~, SortOrder] = sort(fits,'descend');

    ترتیب مرتب سازی بر روی جمعیت را اعمال کنید
    pop = pop(SortOrder);

end
```

IsInRange.m:چک کردن حدبالا و پایین مسیله برای موقعیت راه حل های جدید

```
function b = IsInRange(x, VarMin, VarMax)

    b = all(x >= VarMin) && all(x <= VarMax);

end
```

PlotSolution.m:کشیدن راه حل

```
function PlotSolution(tour,model)

    tour=[tour tour(1)];

    x=model.x;
    y=model.y;

    plot(x(tour),y(tour),'-sk',...
        'LineWidth',2,...
        'MarkerSize',12,...
        'MarkerFaceColor',[1 0.7 0.8]);

end
```

RandSample.m

```
تعداد لیدرها


function L = RandSample(P, q, replacement)

    if ~exist('replacement', 'var')
        replacement = false;
    end

    L = zeros(q, 1);
    for i = 1:q
        L(i) = randsample(numel(P), 1, true, P);
        if ~replacement
            P(L(i)) = 0;
        end
    end

end
```

RunFLA.m:بروزرسانی هر دسته از جمعیت

```
function pop = RunFLA(pop, params)

   پارامترهای قورباغه
    q = params.q;            تعداد دسته
    alpha = params.alpha;    تعداد اعضا هر دسته
    beta = params.beta;      ماکزیمم تعداد تکرار
    sigma = params.sigma;
    fitFunction = params.fitFunction;
    VarMin = params.VarMin;
    VarMax = params.VarMax;
    VarSize = size(pop(1).Position);
    BestSol = params.BestSol;

    nPop = numel(pop);       اندازه جمعیت
    P = 2*(nPop+1-(1:nPop))/(nPop*(nPop+1));     احتمال
انتخاب

    محاسبه دامنه جمعیت
    LowerBound = pop(1).Position;
    UpperBound = pop(1).Position;
    for i = 2:nPop
        LowerBound = min(LowerBound, pop(i).Position);
```

```
        UpperBound = max(UpperBound, pop(i).Position);
    end

    حلقه اصلی قورباغه

    for it = 1:beta

        تعداد لیدرها
        L = RandSample(P, q);
        B = pop(L);

        تخصیص فرزندان
        for k = 1:alpha

                                                    مرتب سازی جمعیت
            [B, SortOrder] = SortPopulation(B);
            L = L(SortOrder);

             Flags
            ImprovementStep2 = false;
            Censorship = false;

            بهبود شماره ۱
            NewSol1 = B(end);
            Step = sigma*rand(VarSize).*(B(1).Position-
B(end).Position);
            NewSol1.Position = B(end).Position + Step;
            if IsInRange(NewSol1.Position, VarMin, VarMax)
                NewSol1.fit =
fitFunction(NewSol1.Position);
                if NewSol1.fit<B(end).fit
                    B(end) = NewSol1;
                else
                    ImprovementStep2 = true;
                end
            else
                ImprovementStep2 = true;
            end

            بهبود شماره ۲
            if ImprovementStep2
                NewSol2 = B(end);
                Step =
sigma*rand(VarSize).*(BestSol.Position-B(end).Position);
```

```
                NewSol2.Position = B(end).Position + Step;
                if IsInRange(NewSol2.Position, VarMin,
VarMax)
                    NewSol2.fit =
fitFunction(NewSol2.Position);
                    if NewSol2.fit<B(end).fit
                        B(end) = NewSol2;
                    else
                        Censorship = true;
                    end
                else
                    Censorship = true;
                end
            end

            بهبود شماره ۳
            if Censorship
                B(end).Position = unifrnd(LowerBound,
UpperBound);
                B(end).fit = fitFunction(B(end).Position);
            end

        end

                                        بازگشت دسته به جمعیت کل
        pop(L) = B;

    end

end
```

sfla.m:پیاده سازی الگوریتم قورباغه(برای اجرای برنامه)

```
clc;
clear;
close all;

تعریف پارامترهای مسیله

model=CreateModel();

fitFunction=@(s) Myfitt(s,model);        تابع برازندگی

nVar=model.n;                تعداد متغیرهای تصمیم
```

```
VarSize=[1 nVar];   اندازه ماتریس متغیرهای تصمیم

VarMin=0;          حدپایین متغیرها
VarMax=1;          حدبالا متغیرها

پارامترهای الگوریتم قورباغه

MaxIt = 50;         حداکثر تعداد تکرار الگوریتم

nPopMemeplex = nVar*2;                                  تعداد جمعیت
دسته ها
nPopMemeplex = max(nPopMemeplex, nVar+1);

nMemeplex = nVar;                    تعداد دسته ها
nPop = nMemeplex*nPopMemeplex;  اندازه جمعیت

I = reshape(1:nPop, nMemeplex, []);

پارامترهای قورباغه
fla_params.q = max(round(0.3*nPopMemeplex), 2);     تعداد
                                                    لیدرها
fla_params.alpha = 10;   تعداد فرزندان
fla_params.beta = 5;    ماکزیمم تعداد تکرار
fla_params.sigma = 1;    اندازه گام
fla_params.fitFunction = fitFunction;
fla_params.VarMin = VarMin;
fla_params.VarMax = VarMax;

مقداردهی اولیه

الگوی فردی خالی
empty_individual.Position = [];
empty_individual.fit = [];
empty_individual.Sol=[];
آرایه جمعیت را مقداردهی اولیه کنید
pop = repmat(empty_individual, nPop, 1);

اعضای جمعیت را اولیه کنید
for i = 1:nPop
    pop(i).Position = unifrnd(VarMin, VarMax, VarSize);
    [ pop(i).fit  pop(i).Sol] =
fitFunction(pop(i).Position);
end
```

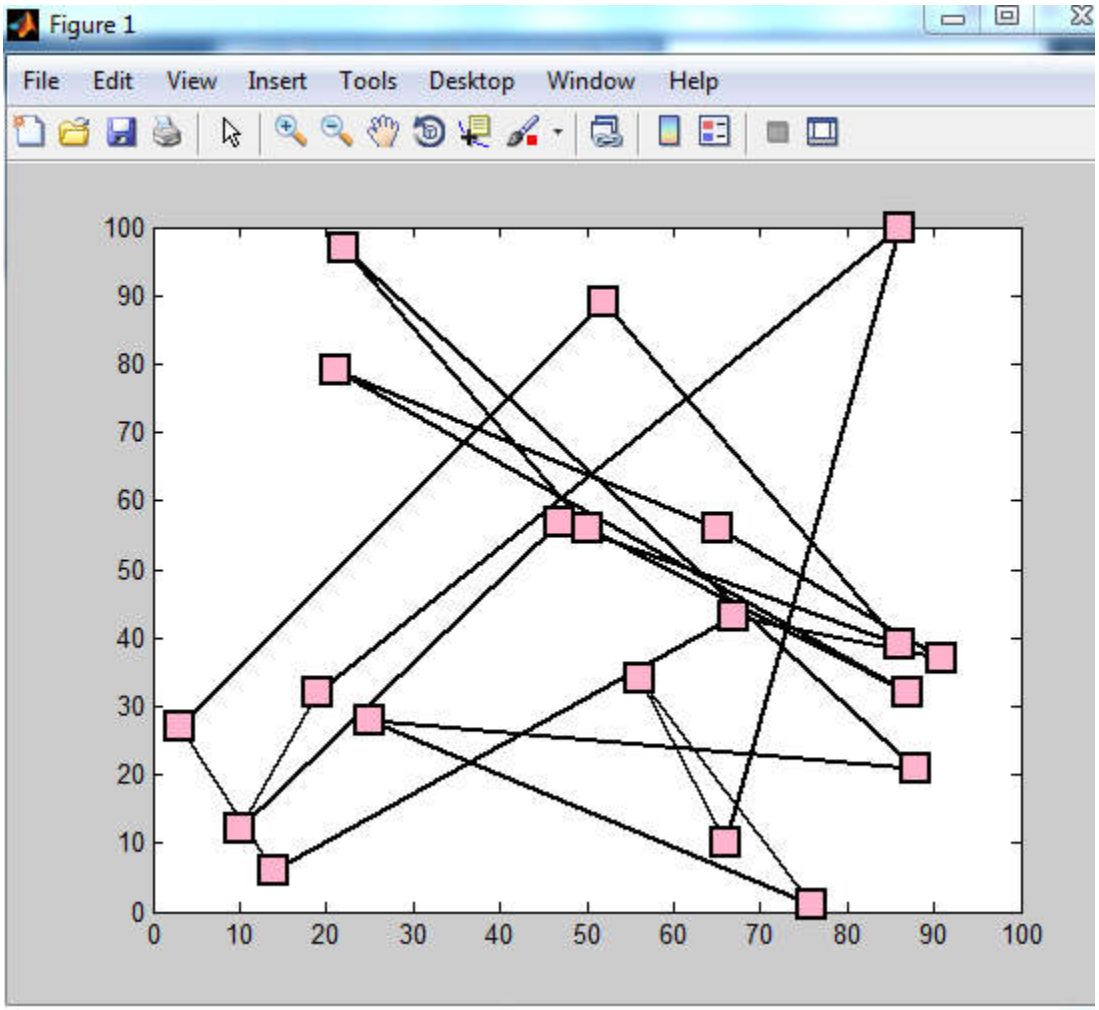
```
مرتب سازی بر جمعیت
pop = SortPopulation(pop);

به روز رسانی بهترین راه حل که تاکنون پیدا شده است
BestSol = pop(1);

مقداردهی اولیه بهترین آرایه ضبط
Bestfits = nan(MaxIt, 1);

حلقه اصلی قورباغه

for it = 1:MaxIt

    fla_params.BestSol = BestSol;

  آرایهMemeplexes دسته های هر قورباغه را مقداردهی اولیه کنید
    Memeplex = cell(nMemeplex, 1);

  دسته های هر قورباغه Memeplexes را تشکیل دهید و FLA را اجرا
                                                        کنید
    for j = 1:nMemeplex
        قرار دادن قورباغه ها در دسته ها بر اساس فیتنس
        Memeplex{j} = pop(I(j, :));

        یروزرسانی هر دسته از جمعیت
        Memeplex{j} = RunFLA(Memeplex{j}, fla_params);

        قرار دادن دسته بروزرسانی شده در جمعیت
        pop(I(j, :)) = Memeplex{j};
    end

                                                مرتب سازی جمعیت
    pop = SortPopulation(pop);

                                بروزرسانی بهترین راه حل پیدا شده
    BestSol = pop(1);

                                ذخیره سازی بهترین برازندگی پیدا شده
    Bestfits(it) = BestSol.fit;

                                                نمایش اطلاعات تکرار
    disp(['Iteration ' num2str(it) ': Best fit = '
num2str(Bestfits(it))]);
                                                کشیدن راه حل
    figure(1);
```

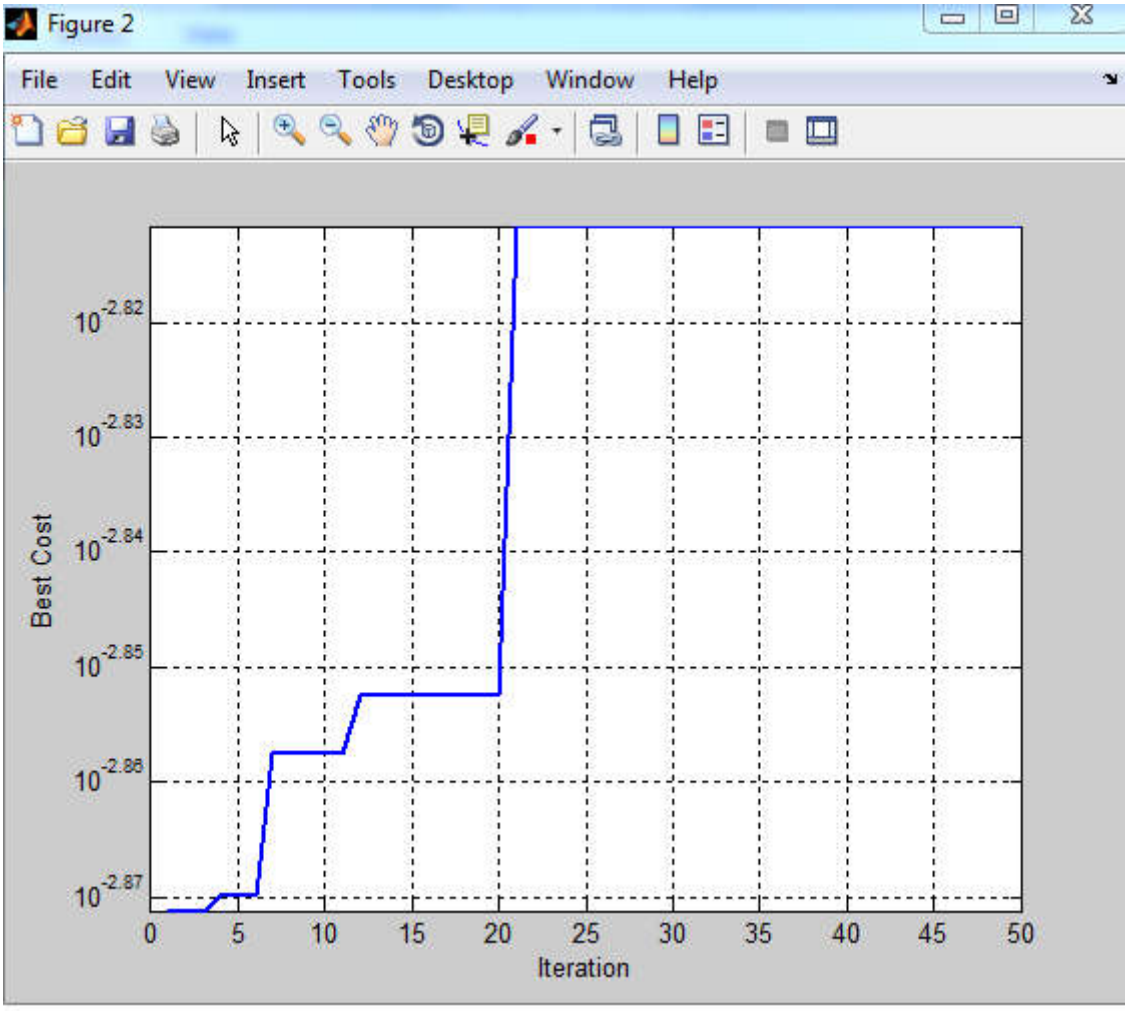
```
    PlotSolution(BestSol.Sol.tour,model);
end
```

نتایج

```
figure;
%plot(Bestfits, 'LineWidth', 2);
semilogy(Bestfits, 'LineWidth', 2);
xlabel('Iteration');
ylabel('Best fit');
grid on;
```

نتایج:






```
Iteration 1: Best Cost = 0.0013454
Iteration 2: Best Cost = 0.0013454
Iteration 3: Best Cost = 0.0013454
Iteration 4: Best Cost = 0.0013493
Iteration 5: Best Cost = 0.0013493
Iteration 6: Best Cost = 0.0013493
Iteration 7: Best Cost = 0.0013882
Iteration 8: Best Cost = 0.0013882
Iteration 9: Best Cost = 0.0013882
Iteration 10: Best Cost = 0.0013882
Iteration 11: Best Cost = 0.0013882
Iteration 12: Best Cost = 0.0014044
Iteration 13: Best Cost = 0.0014044
Iteration 14: Best Cost = 0.0014044
Iteration 15: Best Cost = 0.0014044
Iteration 16: Best Cost = 0.0014044
Iteration 17: Best Cost = 0.0014044
```

```
Iteration 18: Best Cost = 0.0014044
Iteration 19: Best Cost = 0.0014044
Iteration 20: Best Cost = 0.0014044
Iteration 21: Best Cost = 0.001543
Iteration 22: Best Cost = 0.001543
Iteration 23: Best Cost = 0.001543
Iteration 24: Best Cost = 0.001543
Iteration 25: Best Cost = 0.001543
Iteration 26: Best Cost = 0.001543
Iteration 27: Best Cost = 0.001543
Iteration 28: Best Cost = 0.001543
Iteration 29: Best Cost = 0.001543
Iteration 30: Best Cost = 0.001543
Iteration 31: Best Cost = 0.001543
Iteration 32: Best Cost = 0.001543
Iteration 33: Best Cost = 0.001543
Iteration 34: Best Cost = 0.001543
Iteration 35: Best Cost = 0.001543
Iteration 36: Best Cost = 0.001543
Iteration 37: Best Cost = 0.001543
Iteration 38: Best Cost = 0.001543
Iteration 39: Best Cost = 0.001543
Iteration 40: Best Cost = 0.001543
Iteration 41: Best Cost = 0.001543
Iteration 42: Best Cost = 0.001543
Iteration 43: Best Cost = 0.001543
Iteration 44: Best Cost = 0.001543
Iteration 45: Best Cost = 0.001543
Iteration 46: Best Cost = 0.001543
Iteration 47: Best Cost = 0.001543
Iteration 48: Best Cost = 0.001543
Iteration 49: Best Cost = 0.001543
Iteration 50: Best Cost = 0.001543
>>

BestSol =

    Position: [1x20 double]
        Cost: 0.0015
         Sol: [1x1 struct]

>> BestSol.Position

ans =
```

```
  Columns 1 through 10

    0.7703    0.3679    0.5140    0.3363    0.8769
0.6709    0.3959    0.9364    0.7437    0.8476

  Columns 11 through 20

    0.8389    0.3578    0.4068    0.2730    0.8349
0.6841    0.3343    0.7189    0.8672    0.6276

>>
```