حل مسئله Bin Packing با الگوریتم کرم شب تاب FA در متلب

در مسئله Bin Packing اشیاء دارای حجم‌های مختلف باید در تعداد متناهی از جعبه از حجم V به شکلی که تعداد جعبه‌های استفاده‌شده کمینه شود قرار داده شود.

صورت مسئله:

۳۰ تا شی با حجم مشخص به این صورت داریم:

```
v = [6 3 4 6 8 7 4 7 7 5 5 6 7 7 6 4 8 7 8 8 2 3 4 5 6 5 5
7 7 12]
```

می خواهیم این اشیا را به گونه ای بسته بندی کنیم که هر بسته به اندازه حجم ۳۰ بیشتر جا نگیرد، همه اشیا باید بسته بندی شوند و هیچ شی نباید در دو بسته قرار بگیرد.

شرح کد:

این سورس کد شامل ۳ فایل می باشد که عبارتند از:

CreateModel.m: برای ایجاد مدل(همان اشیا) و مشخص کردن تعداد آن ها و حداکثر حجم هر بسته و مقدار دهی اولیه پارامتر های مدل مسیله از آن استفاده می شود.

```
function model = CreateModel()

    model.v = [6 3 4 6 8 7 4 7 7 5 5 6 7 7 6 4 8 7 8 8 2 3
4 5 6 5 5 7 7 12];
            ایجاد اشیا و مشخص کردن حجم آن ها و دادن آن به مدل

    model.n = numel(model.v);
                        مشخص کردن تعداد اشیا و دادن آن به مدل

    model.Vmax = 30;
                    مشخص کردن حداکثر حجم بسته و دادن آن به مدل

end
```

BinPackingFit.m:این تابع برای مشخص کردن میزان برازش راه حل می باشد.

```
function [z, sol] = BinPackingFit(x, model)

    n = model.n;                                تعداد اشیا
    v = model.v;                                حجم اشیا
    Vmax = model.Vmax;                  ماکزیمم حجم هر بسته

    [~, q]=sort(x,'descend');       مرتب سازی نزولی جمعیت

    Sep = find(q>n); پیدا کردن شی بزرگ تر از بزرگ ترین شی

    From = [0 Sep] + 1;          آرایه ای از ۰ تا بزرگترین شی
    To = [Sep 2*n] - 1; آرایه ای از بزرگترین شی تا یکی کمتر
                                                از ۲ برابر تعداد اشیا

    B = {};
    for i=1:n
        Bi = q(From(i):To(i)); برای همه اشیا اگر حجمشان
                                        بزرگتر از ۰ است قبول است
        if numel(Bi)>0
            B = [B; Bi]; %#قبوله
        end
    end

    nBin = numel(B);
                                                    B به تعداد nBin
    Viol = zeros(nBin,1);
                                        تخلف، یک آرایه به طول nBin

    for i=1:nBin
                                    حجم هر بسته با توجه به اشیا درونش

        Vi = sum(v(B{i}));
    ماکزیمم Vi تقسیم بر ماکزیمم حجم گنجایش هر بسته و ۰ به ما
                                                میزان تخلف را می دهد

        Viol(i) = max(Vi/Vmax-1, 0);
    end

                                                تخطی، میانگین Viol
    MeanViol = mean(Viol);
```

```
                یک ضریبی از تخطی می خواهیم که n تعداد اشیا است
    alpha = 2*n;
            تعداد بسته ها + الفا ضربدر تخطی هزینه می شود.

    K = nBin + alpha*MeanViol;

        برازندگی عکس هزینه با یک ضریب ۱ در مخرج جمع می شود.

    Z=1/(1+k);
                        این مقادیر را به راه حل می دهیم.
    sol.nBin = nBin;
    sol.B = B;
    sol.Viol = Viol;
    sol.MeanViol = MeanViol;

end
```

fa.m: که فایل اصلی برنامه است که الگوریتم کرم شب تاب در آن نوشته شده و پارامترهایش تعریف و مقداردهی شده است و دو تابع قبلی برای تعریف مسیله و محاسبه برازندگی در آن فراخوانی شده است.

```
clc;
clear;
close all;

                                                تعریف مسیله

model = CreateModel();                  ایجاد مدل Bin Packing
                                        تابع محاسبه فیتنس

fitFunction = @(x) BinPackingFit(x, model);

nVar = 2*model.n-1;       تعداد متغیرهای تصمیم(تعداد خانه های
آرایه که اشیا را درآن قرار می دهیم)

VarSize = [1 nVar];       اندازه ماتریس متغیرهای تصمیم(اندازه
آرایه حاوی اشیا)

VarMin = 0;       حد پایین مقادیر اشیا
VarMax = 1;        حد بالا مقادیر اشیا

                                                پارامترهای fa

MaxIt = 150;      ماکزیمم تعداد تکرار الگوریتم
```

```
nPop=20;              تعداد کرم شب تاب اندازه جمعیت

gamma=1;              ضریب جذب نور

beta0=2;              مقدار پایه ضریب جذب


alpha=0.2;            ضریب جهش


alpha_damp=0.98;      نسبت میرایی ضریب جهش


delta=0.05*(VarMax-VarMin);      محدوده جهش یکنواخت

m=2;

if isscalar(VarMin) && isscalar(VarMax)
    dmax = (VarMax-VarMin)*sqrt(nVar);
else
    dmax = norm(VarMax-VarMin);
end

nMutation = 3;        تعداد عملیات جهش اضافی


جمعیت اولیه مقداردهی

 ساختار کرم شب تاب خالی
firefly.Position=[];
firefly.Fit=[];
firefly.Sol=[];

آرایه جمعیت را مقداردهی اولیه کنید
pop=repmat(firefly,nPop,1);

بهترین راه حل را که تاکنون پیدا کرده اید اولیه کنید
BestSol.Fit=-inf;

کرم شب تاب اولیه ایجاد کنید
for i=1:nPop
   pop(i).Position=unifrnd(VarMin,VarMax,VarSize);
   [pop(i).Fit, pop(i).Sol]=FitFunction(pop(i).Position);
```

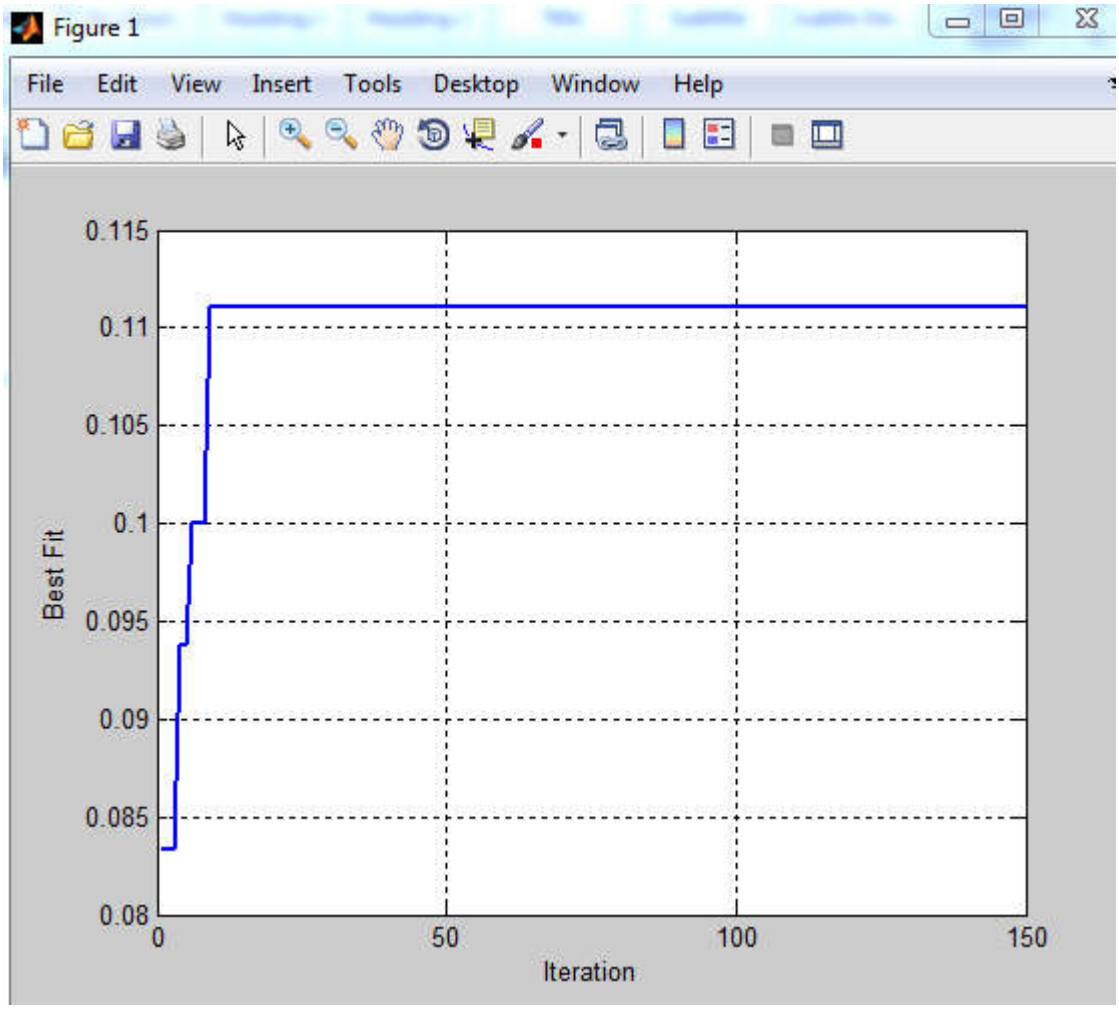
```
    if pop(i).Fit>=BestSol.Fit
        BestSol=pop(i);
    end
end

آرایه برای نگه داشتن بهترین مقادیر متناسب
BestFit=zeros(MaxIt,1);

حلقه اصلی الگوریتم کرم شب تاب
for it=1:MaxIt

    newpop=repmat(firefly,nPop,1);
    for i=1:nPop
        newpop(i).Fit = -inf; مقدار اولیه برازندگی جمعیت
جدید را منفی بی نهایت میداریم
        for j=1:nPop
            if pop(j).Fit > pop(i).Fit || i==j
                rij=norm(pop(i).Position-
pop(j).Position)/dmax;
                beta=beta0*exp(-gamma*rij^m);
                e=delta*unifrnd(-1,+1,VarSize);
                %e=delta*randn(VarSize);

                newsol.Position = pop(i).Position ...
                                  +
beta*rand(VarSize).*(pop(j).Position-pop(i).Position) ...
                                  + alpha*e;


newsol.Position=max(newsol.Position,VarMin);

newsol.Position=min(newsol.Position,VarMax);

                [newsol.Fit,
newsol.Sol]=FitFunction(newsol.Position);

                if newsol.Fit >= newpop(i).Fit
                    newpop(i) = newsol;
                    if newpop(i).Fit>=BestSol.Fit
                        BestSol=newpop(i);
                    end
                end

            end
```

```
        end

                                                    جهش را انجام دهید
        for k=1:nMutation
            newsol.Position = Mutate(pop(i).Position);
            [newsol.Fit,
newsol.Sol]=FitFunction(newsol.Position);
            if newsol.Fit >= newpop(i).Fit
                newpop(i) = newsol;
                if newpop(i).Fit>=BestSol.Fit
                    BestSol=newpop(i);
                end
            end
        end

    end

    ادغام
    pop=[pop
         newpop];   خوبه

    حلقه
    [~, SortOrder]=sort([pop.Fit],'descend');
    pop=pop(SortOrder);

    کوتاه کردن
    pop=pop(1:nPop);

    بهترین برازندگی موجود را ذخیره کنید
    BestFit(it)=BestSol.Fit;

    نمایش اطلاعات تکرار
    disp(['Iteration ' num2str(it) ': Best Fit = '
num2str(BestFit(it))]);

   تغیرات ضریب جهش
    alpha = alpha*alpha_damp;

end

نتایج

figure;
plot(BestFit,'LineWidth',2);
```

```
xlabel('Iteration');
ylabel('Best Fit');
grid on;
```

بعد از اجرای کد fa.m نتایج به شرح زیر است:

نتایج:




```
Iteration 1: Best Fit = 0.083333
Iteration 2: Best Fit = 0.083333
Iteration 3: Best Fit = 0.083333
Iteration 4: Best Fit = 0.09375
Iteration 5: Best Fit = 0.09375
Iteration 6: Best Fit = 0.1
Iteration 7: Best Fit = 0.1
Iteration 8: Best Fit = 0.1
Iteration 9: Best Fit = 0.11111
```

```
Iteration 10: Best Fit = 0.11111
Iteration 11: Best Fit = 0.11111
Iteration 12: Best Fit = 0.11111
Iteration 13: Best Fit = 0.11111
Iteration 14: Best Fit = 0.11111
Iteration 15: Best Fit = 0.11111
Iteration 16: Best Fit = 0.11111
Iteration 17: Best Fit = 0.11111
Iteration 18: Best Fit = 0.11111
Iteration 19: Best Fit = 0.11111
Iteration 20: Best Fit = 0.11111
Iteration 21: Best Fit = 0.11111
Iteration 22: Best Fit = 0.11111
Iteration 23: Best Fit = 0.11111
Iteration 24: Best Fit = 0.11111
Iteration 25: Best Fit = 0.11111
Iteration 26: Best Fit = 0.11111
Iteration 27: Best Fit = 0.11111
Iteration 28: Best Fit = 0.11111
Iteration 29: Best Fit = 0.11111
Iteration 30: Best Fit = 0.11111
Iteration 31: Best Fit = 0.11111
Iteration 32: Best Fit = 0.11111
Iteration 33: Best Fit = 0.11111
Iteration 34: Best Fit = 0.11111
Iteration 35: Best Fit = 0.11111
Iteration 36: Best Fit = 0.11111
Iteration 37: Best Fit = 0.11111
Iteration 38: Best Fit = 0.11111
Iteration 39: Best Fit = 0.11111
Iteration 40: Best Fit = 0.11111
Iteration 41: Best Fit = 0.11111
Iteration 42: Best Fit = 0.11111
Iteration 43: Best Fit = 0.11111
Iteration 44: Best Fit = 0.11111
Iteration 45: Best Fit = 0.11111
Iteration 46: Best Fit = 0.11111
Iteration 47: Best Fit = 0.11111
Iteration 48: Best Fit = 0.11111
Iteration 49: Best Fit = 0.11111
```

```
Iteration 50: Best Fit = 0.11111
Iteration 51: Best Fit = 0.11111
Iteration 52: Best Fit = 0.11111
Iteration 53: Best Fit = 0.11111
Iteration 54: Best Fit = 0.11111
Iteration 55: Best Fit = 0.11111
Iteration 56: Best Fit = 0.11111
Iteration 57: Best Fit = 0.11111
Iteration 58: Best Fit = 0.11111
Iteration 59: Best Fit = 0.11111
Iteration 60: Best Fit = 0.11111
Iteration 61: Best Fit = 0.11111
Iteration 62: Best Fit = 0.11111
Iteration 63: Best Fit = 0.11111
Iteration 64: Best Fit = 0.11111
Iteration 65: Best Fit = 0.11111
Iteration 66: Best Fit = 0.11111
Iteration 67: Best Fit = 0.11111
Iteration 68: Best Fit = 0.11111
Iteration 69: Best Fit = 0.11111
Iteration 70: Best Fit = 0.11111
Iteration 71: Best Fit = 0.11111
Iteration 72: Best Fit = 0.11111
Iteration 73: Best Fit = 0.11111
Iteration 74: Best Fit = 0.11111
Iteration 75: Best Fit = 0.11111
Iteration 76: Best Fit = 0.11111
Iteration 77: Best Fit = 0.11111
Iteration 78: Best Fit = 0.11111
Iteration 79: Best Fit = 0.11111
Iteration 80: Best Fit = 0.11111
Iteration 81: Best Fit = 0.11111
Iteration 82: Best Fit = 0.11111
Iteration 83: Best Fit = 0.11111
Iteration 84: Best Fit = 0.11111
Iteration 85: Best Fit = 0.11111
Iteration 86: Best Fit = 0.11111
Iteration 87: Best Fit = 0.11111
Iteration 88: Best Fit = 0.11111
Iteration 89: Best Fit = 0.11111
```

```
Iteration 90: Best Fit = 0.11111
Iteration 91: Best Fit = 0.11111
Iteration 92: Best Fit = 0.11111
Iteration 93: Best Fit = 0.11111
Iteration 94: Best Fit = 0.11111
Iteration 95: Best Fit = 0.11111
Iteration 96: Best Fit = 0.11111
Iteration 97: Best Fit = 0.11111
Iteration 98: Best Fit = 0.11111
Iteration 99: Best Fit = 0.11111
Iteration 100: Best Fit = 0.11111
Iteration 101: Best Fit = 0.11111
Iteration 102: Best Fit = 0.11111
Iteration 103: Best Fit = 0.11111
Iteration 104: Best Fit = 0.11111
Iteration 105: Best Fit = 0.11111
Iteration 106: Best Fit = 0.11111
Iteration 107: Best Fit = 0.11111
Iteration 108: Best Fit = 0.11111
Iteration 109: Best Fit = 0.11111
Iteration 110: Best Fit = 0.11111
Iteration 111: Best Fit = 0.11111
Iteration 112: Best Fit = 0.11111
Iteration 113: Best Fit = 0.11111
Iteration 114: Best Fit = 0.11111
Iteration 115: Best Fit = 0.11111
Iteration 116: Best Fit = 0.11111
Iteration 117: Best Fit = 0.11111
Iteration 118: Best Fit = 0.11111
Iteration 119: Best Fit = 0.11111
Iteration 120: Best Fit = 0.11111
Iteration 121: Best Fit = 0.11111
Iteration 122: Best Fit = 0.11111
Iteration 123: Best Fit = 0.11111
Iteration 124: Best Fit = 0.11111
Iteration 125: Best Fit = 0.11111
Iteration 126: Best Fit = 0.11111
Iteration 127: Best Fit = 0.11111
Iteration 128: Best Fit = 0.11111
Iteration 129: Best Fit = 0.11111
```

```
Iteration 130: Best Fit = 0.11111
Iteration 131: Best Fit = 0.11111
Iteration 132: Best Fit = 0.11111
Iteration 133: Best Fit = 0.11111
Iteration 134: Best Fit = 0.11111
Iteration 135: Best Fit = 0.11111
Iteration 136: Best Fit = 0.11111
Iteration 137: Best Fit = 0.11111
Iteration 138: Best Fit = 0.11111
Iteration 139: Best Fit = 0.11111
Iteration 140: Best Fit = 0.11111
Iteration 141: Best Fit = 0.11111
Iteration 142: Best Fit = 0.11111
Iteration 143: Best Fit = 0.11111
Iteration 144: Best Fit = 0.11111
Iteration 145: Best Fit = 0.11111
Iteration 146: Best Fit = 0.11111
Iteration 147: Best Fit = 0.11111
Iteration 148: Best Fit = 0.11111
Iteration 149: Best Fit = 0.11111
Iteration 150: Best Fit = 0.11111
>>

>> BestSol.Position

ans =

  Columns 1 through 11

    0.3301    0.6989    0.6983    0.5683    0.5882
0.3889    0.2580    0.7274    0.0100    0.2013
0

  Columns 12 through 22

    0.0615    0.5673    0.1138         0    0.7783
0.0723    0.5408    0.5468    0.2897    0.3202
0.1751

  Columns 23 through 33
```

```
    0.3052    0.3670    0.1879    0.2427    0.5821
0    0.1912    0.3663    0.5579    0.4544    0.6931

  Columns 34 through 44

    0.4189    0.8773    0.4587    0.2099    0.2063
0.1602    0.6304    0.0168    0.4292    0.3338
0.7845

  Columns 45 through 55

    0.0147    0.8519    0.3966    0.6838    0.2403
0.4810    0.4233         0    0.6639    0.2346
0.3567

  Columns 56 through 59

    0.6799    0.4412    0.4449    0.4721

>>
```