# 二进制编码遗传算法解 0-1 背包问题

李馨　202011140104

## 1　问题描述

**0-1 背包问题**：给定 $n$ 个不同物品，每种物品都有自己的重量 $w_j$ 和价值 $c_j$，背包所能容纳的总重量为 $W$，我们如何选择，才能在背包中尽可能装入总价值最多的物品，且不超过背包的承重限制。

0 - 1背包问题的数学公式如下：

$$\max \quad f(x) = \sum_j c_j \times x_j$$

$$s.t. \quad g(x) = \sum_j w_j \times x_j \leq W$$

$$x_j = 0或1, \quad j = 1, \quad 2, \ldots, n$$

$$其中 x_j = \begin{cases} 1, & 若选择第j个物品 \\ 0, & 否则 \end{cases}$$

图 1: 背包问题数学描述

本问题中，物品数量 $n = 20$，物品的重量 $w_j$ 和价值 $c_j$ 见下表：

| $j$ | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 价值 $c_j$ | 91 | 72 | 90 | 46 | 55 | 8 | 35 | 75 | 61 | 15 | 77 | 40 | 63 | 75 | 29 | 75 | 17 | 78 | 40 | 44 |
| 重量 $w_j$ | 84 | 83 | 43 | 4 | 44 | 6 | 82 | 92 | 25 | 83 | 56 | 18 | 58 | 14 | 48 | 70 | 96 | 32 | 68 | 92 |

## 2　编程思路及方法

我们使用二进制编码遗传算法解决 0-1 背包问题，编程语言为 MATLAB。

### 2.1　参数选择

设置遗传算法使用的参数值（种群大小、应用遗传算子的概率等）。

本问题设定种群规模 $N_p = 20$，迭代次数 $T = 100$，交叉概率 $p_c = 0.8$，变异概率 $p_m = 0.3$。这些参数也可在函数文件中修改。

### 2.2　编码

对问题的潜在解进行基因的表示。

潜在解表示为一个 $1 \times 20$ 的行向量，第 $j$ 个元素为 1 则表示选择第 $j$ 个物品，为 0 则表示不选择。例如表示选择第 2 个、第 10 个和第 17 个物品的潜在解：

$$[0\ 1\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 1\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 1\ 0\ 0]$$

## 2.3 种群初始化

构建一组潜在的解决方案：

```
X = rand(Np, n) > 0.5;      % 初始种群
```

该行代码随机生成一个 $N_p \times n$ 的矩阵，其中 $N_p$ 是种群规模，$n$ 是物品数量。

## 2.4 个体评价

根据潜在解的适应性来评价解的好坏。本问题中适应度函数 $f(x)$ 和关于 $g(x)$ 的限制条件见图1，需要在 $g(x)$ 不超过容重 $W = 879$ 的条件下使得总价值 $f(x)$ 最大化。下方代码为对于给定种群矩阵 $X$，计算 $f$ 和 $g$ 的一个函数。其中对于超过容重的不可行解，使用解码方法进行处理，即按照价值重量比次序选择该解中的物品，直到背包不能再放入物品，这时的总价值作为该解的 $f$：

```
function [f, g, J] = calculatef(W, X, c, w)
% 输入限重，种群，价值和重量，计算适应度函数f，以及总重g，同时返回c/w降序排列索引
    Np = size(X, 1);
    ratio = c./w;          % c/w比率
    % 不可行解的处理方法：解码方法
    ratio1 = ratio;                % 复制一份ratio用于循环中修改、使用
    J = [];                        % 用于索引储存
    for i = 1 : size(X, 2)
        % 该循环用于确定c/w比率降序排列后各元素的原索引
        [¬, j] = max(ratio1);      % 下一个最大比率值及其索引
        ratio1(j) = 0;             % 已遍历过的值设为0
        J = [J, j];                % 按顺序储存索引
    end

    f = zeros(Np, 1); g = zeros(Np, 1);
    for k = 1 : Np
        % 该层循环用于处理第 k 个解，即第 k 行
        for j = J
            % 该层循环按顺序（J中已储存索引）选择物品，并得出g和f
            if (g(k) + X(k, j) * w(j) > W)  % 超重处理
                break                       % 退出循环
            end
            g(k) = g(k) + X(k, j) * w(j);   % 总重量
            f(k) = f(k) + X(k, j) * c(j);   % 总价值
        end
    end
end
```

其中用到的 $J$ 是一个 $1 \times n$ 的行向量，储存物品价值重量比信息。若其第 $i$ 个元素为 $j$，表示物品 $j$ 是价值重量比从高到低排在第 $i$ 个的物品。

## 2.5 遗传算子

改变后代基因组成的遗传算子（选择、交叉、变异等）。

### 2.5.1 选择

本问题利用轮盘赌法选择父代解，适应度较高的解更容易被选到。下方为进行轮盘赌选择的函数：

```
function j = RWS(f)
% 轮盘赌选择，读取适应度函数列向量f，返回索引
    P = f / sum(f);              % 选择概率
    PP = [0; cumsum(P)];         % 概率的累积和
    r = rand(1);
    for j = 1 : size(f, 1)
        if (r ≥ PP(j)) && (r ≤ PP(j+1))
            break
        end
    end
end
```

关于 PP 和 P 的关系：如 P 是 $[0.1 \quad 0.2 \quad 0.3 \quad 0.4]^T$，则得到 PP= $[0 \quad 0.1 \quad 0.3 \quad 0.6 \quad 1]^T$。

### 2.5.2 交叉

本问题利用单点交叉方法，交叉概率 $p_c = 0.8$，生成随机数 $r$，若小于 $p_c$ 进行该组父代解的交叉操作。得到一组子代。该部分代码如下，其中 $i$ 表示第 $i$ 组父代解：

```
r = rand(1);                 % 决定是否进行交叉操作
if r < pc
    r = ceil(rand(1) * n); % 决定交叉点的随机数，ceil是向上取整
    % 交叉操作
    if r == 1
        offspring(2*i-1, :) = parent(2*i, :);
        offspring(2*i, :) = parent(2*i-1, :);
    else
        offspring(2*i-1, :) = [parent(2*i-1, 1:r-1), parent(2*i, r:end)];
        offspring(2*i, :) = [parent(2*i, 1:r-1), parent(2*i-1, r:end)];
    end
else
    offspring(2*i-1, :) = parent(2*i-1, :);
    offspring(2*i, :) = parent(2*i, :);
end
```

### 2.5.3 变异

对序列中的每一位生成一个随机数，与变异概率 $p_m$ 进行比较，若小于 $p_m$，进行翻转操作。该部分关键代码为：

```
mutation = rand(Np, n) < pm;     % Np*n的矩阵，由随机的0和1组成，为1的概率是pm
offspring = abs(offspring - mutation);
```

### 2.6 生成下一代组群

合并子代和父代，计算所有的 $f$，根据幸存策略，从中选取 $N_p$ 个最优序列作为下一代。

```
[f1, g1] = calculatef(W, X, c, w);
[f2, g2] = calculatef(W, offspring, c, w);
% 合并
XX = [X;offspring];
f = [f1; f2];
g = [g1; g2];
% 创造下一代种群
ff = f;
for i = 1 : Np
    % 储存最优的Np个解，作为下一代种群
    [¬, j] = max(ff);    % 下一个最优解f值及其索引
    ff(j) = 0;           % 已遍历过的解设为0
    X(i, :) = XX(j, :);
end
```

将新一代继续重复上述流程，共迭代 $T$（默认为 100）次。

## 3 运行结果

在种群规模 $N_p = 40$，$T = 100$ 的情况下，运行 50 次所需时间大概平均在 $3.5s$，正确率大概为 50%，测试代码及结果如下：

```
W = 879; % 限重
c = [91; 72; 90; 46; 55; 8; 35; 75; 61; 15; 77;...
    40; 63; 75; 29; 75; 17; 78; 40; 44];        % 价值
w = [84; 83; 43; 4; 44; 6; 82; 92; 25; 83; 56;...
    18; 58; 14; 48; 70; 96; 32; 68; 92];        % 重量

tic
bingo = 0;
for i = 1 : 50
    [x, f, g] = knapsackProblem(W, c, w);
    if (f == 1025) & (g == 871)
        bingo = bingo + 1;
    end
end
toc
bingo

tic
bingo = 0;
for i = 1 : 50
    [x, f, g] = knapsackProblem(W, c, w);
    if (f == 1025) & (g == 871)
        bingo = bingo + 1;
    end
end
toc
bingo
```

```
历时 3.544326 秒。
bingo = 28

历时 3.461819 秒。
bingo = 21
```

图 2: 运行结果

knapsackProblem.m 是该问题全部代码，test.mlx 是测试使用代码，即图2所截代码。