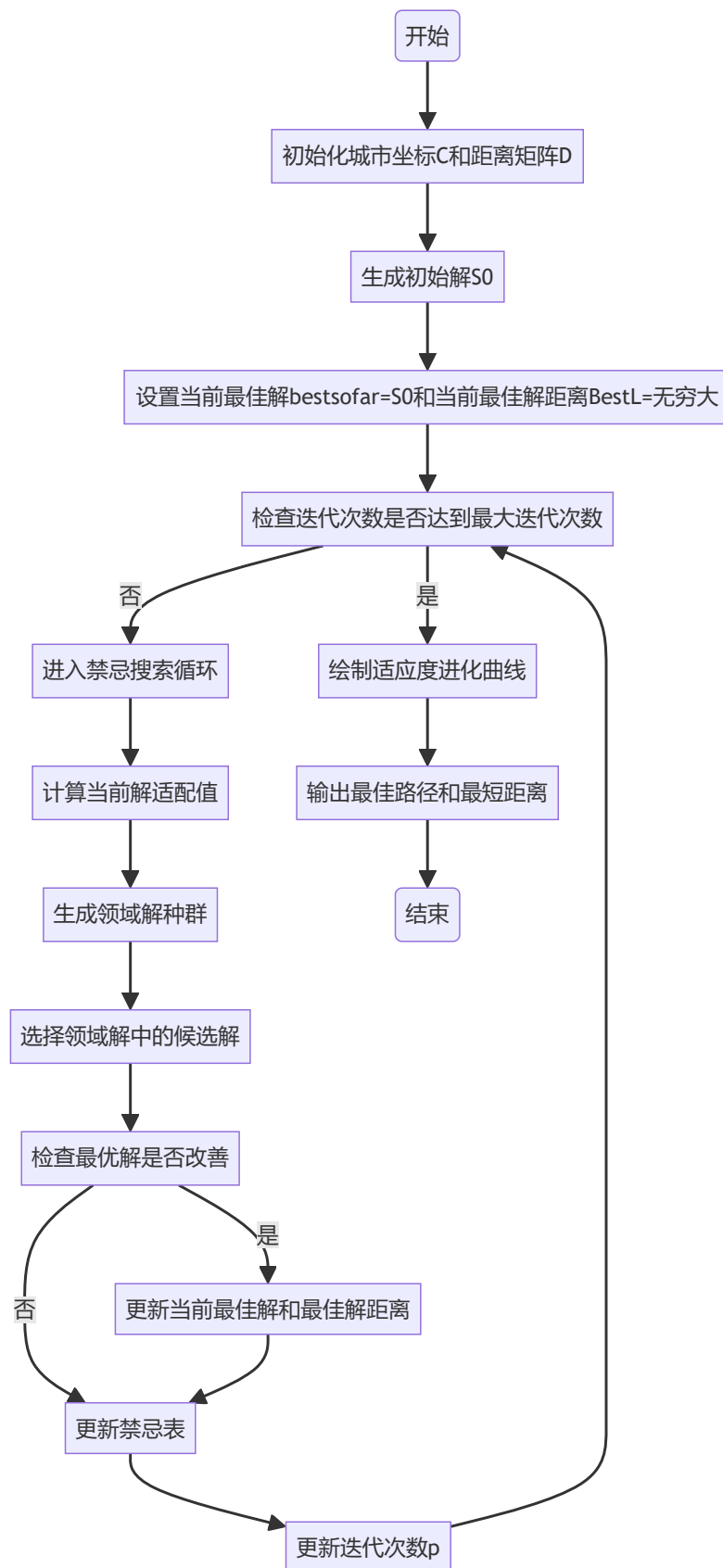


禁忌搜索解旅行商问题

1. 禁忌搜索算法图

禁忌搜索的特点是利用禁忌表来避免重复访问先前搜索过的解，从而有助于跳出局部最优解。这意味着禁忌搜索不仅仅考虑当前最优解，还会考虑过去搜索过的解，以避免陷入局部最优解而无法找到全局最优解。通过记录禁忌表，禁忌搜索可以在搜索过程中避免重复探索相似的解，从而更有可能在搜索空间中探索到更优的解决方案。



初始参数:

```

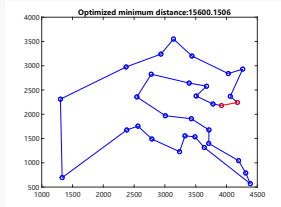
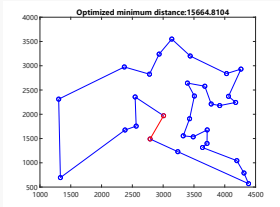
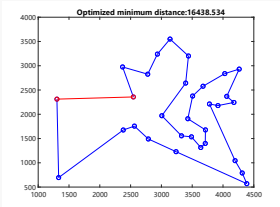
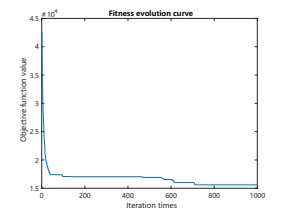
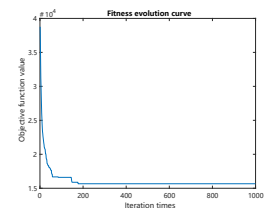
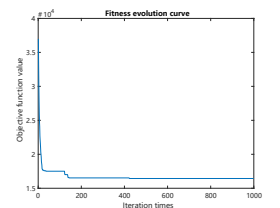
TabuL=round((N*(N-1)/2)^0.5); %禁忌长度，固定为与问题规模相关的一个量
Ca=200; %领域解个数
S0=randperm(N); %随机产生初始解
Gmax=1000;

```

2. 单独调节领域解

以下是领域解变化对算法性能的一些可能影响：

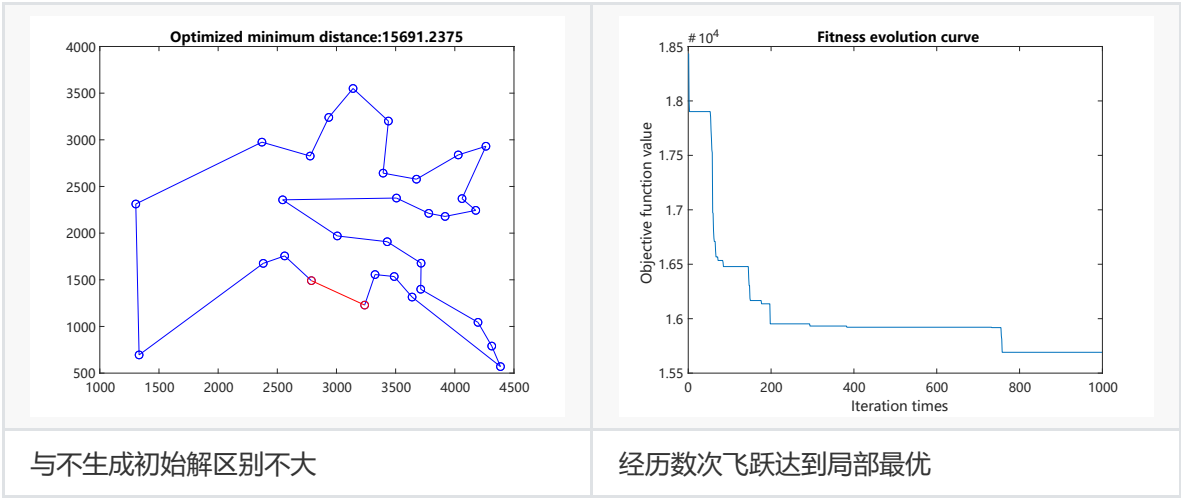
- 解空间覆盖性：**领域解的变化可以增加对解空间的覆盖程度，使得算法能够更全面地搜索潜在的解。更广泛的解空间覆盖通常意味着更大的搜索空间，这可能会增加找到最优解的机会。
- 局部搜索能力：**领域解的变化可以帮助算法摆脱局部最优解，因为它可以尝试不同的解决方案并找到更好的解。通过尝试多个领域解，算法可以更有可能找到全局最优解而不是陷入局部最优解。
- 搜索速度：**领域解的数量和质量会影响算法的搜索速度。如果领域解的数量过多，可能会增加计算成本，导致算法运行时间较长。另一方面，如果领域解的数量太少，可能会限制算法的搜索能力，导致可能的最优解被忽略。56-72-89
- 收敛性：**领域解的变化可以影响算法的收敛性，即算法是否能够在有限的迭代次数内收敛到一个较好的解。如果领域解的变化能够在每次迭代中提供足够的多样性和有效的搜索方向，那么算法更有可能在较短的时间内收敛到一个令人满意的解。
- 最终解的质量：**领域解的变化直接影响最终得到的解的质量。如果领域解的变化能够引导算法朝着更优的方向前进，并且能够充分探索解空间，那么最终得到的解可能会更接近最优解。

Ca	100	200	300
profile time	56	72	89
Optimized minimum distance			
Fitness evolution curve			

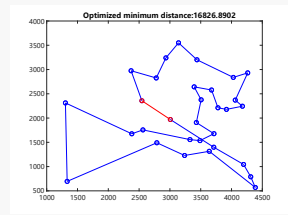
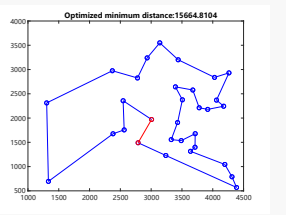
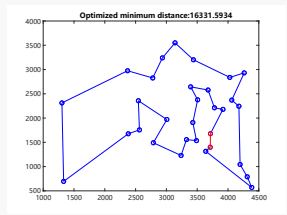
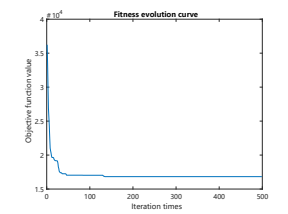
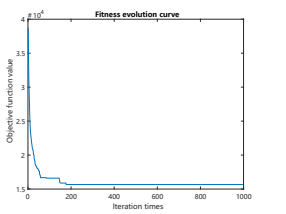
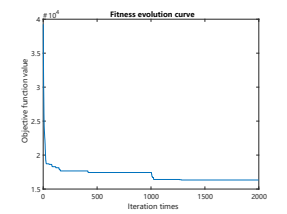
3. 单独调节初始解

尝试使用贪婪算法生成一个初始解，然后对其进行局部优化来改善初始解。

实际效果一般。



4. 单独调节最大迭代次数

Gmax	500	1000	2000
profile time	30	72	190
Optimized minimum distance			
Fitness evolution curve			

5. 单独调节禁忌表长

禁忌长度是禁忌搜索算法中的一个重要参数，它直接影响着算法的搜索能力和收敛速度。理论上禁忌长度的变化会对实验结果产生以下影响：

- 搜索空间覆盖度：**禁忌长度的增加可以增加算法对搜索空间的覆盖度，即在每次迭代中，算法能够考虑更多的解。这有助于提高算法找到全局最优解的概率。

2. **搜索速度**：一般来说，禁忌长度越长，算法的搜索速度就越慢。因为禁忌长度决定了算法记忆中保留的历史信息量，长禁忌长度会增加算法的计算复杂度和存储需求，导致搜索速度变慢。
3. **局部搜索能力**：适当增加禁忌长度可以增强算法的局部搜索能力，使得算法更容易跳出局部最优解，寻找更优的解决方案。
4. **收敛性能**：禁忌长度的变化会影响算法的收敛速度和稳定性。通常情况下，合适的禁忌长度可以加快算法的收敛速度，使得算法更快地找到较优解。但是如果禁忌长度设置不当，可能会导致算法陷入局部最优解或者在搜索空间中徘徊。

然而在实验中，可能由于调优范围过小，禁忌表长的改变并未造成搜索速度的明显提升：

TabuL	$\text{round}(0.5/(N-1)/2)^{0.5})$	$\text{round}(0.75/(N-1)/2)^{0.5})$	$\text{round}((N*(N-1)/2)^{0.5})$
profile time	约70	约70	约70

6. 参数综合调优

经过实验，以下参数已经较好地实现了优化：

```

TabuL=round((N*(N-1)/2)^0.5);           %禁忌长度，固定为与问题规模相关的一个量
Ca=170;                                  %领域解个数

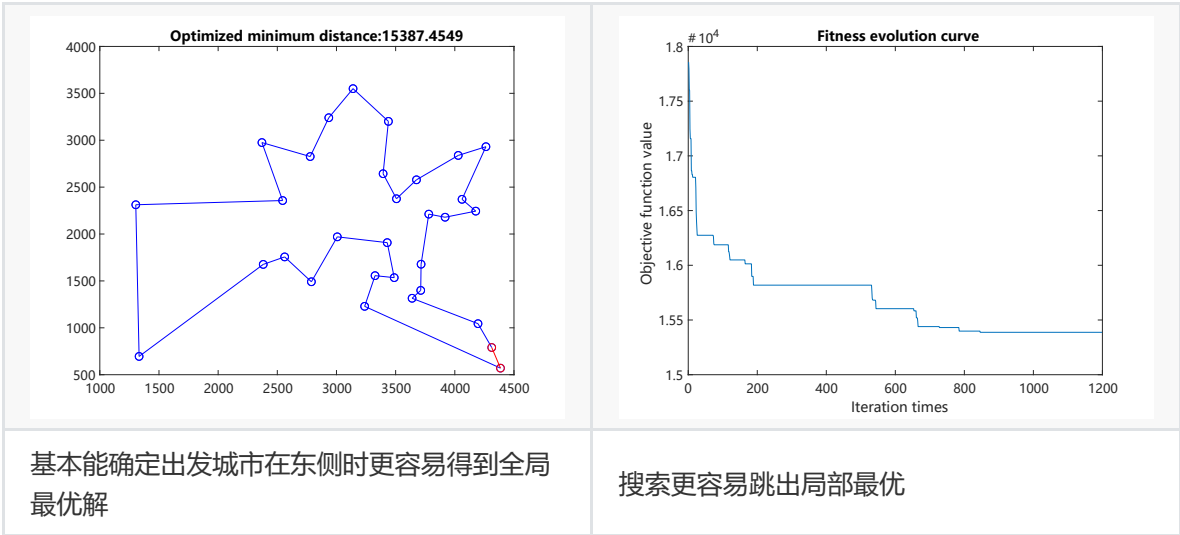
% 使用贪婪算法生成初始解
S0 = greedyAlgorithm(C);

% 对初始解进行局部优化
S0 = localOptimization(D, S0);

Gmax=1200;                               %最大迭代次数

```

Fence 6-1



7. 参数自整定

profile time = 199s

```

TabuL=round(0.5*(N*(N-1)/2)^0.5); % 根据经验
Ca=ceil(3.7*N); % 根据经验

% 使用贪婪算法生成初始解
S0 = greedyAlgorithm(C);

% 对初始解进行局部优化
S0 = localOptimization(D, S0);

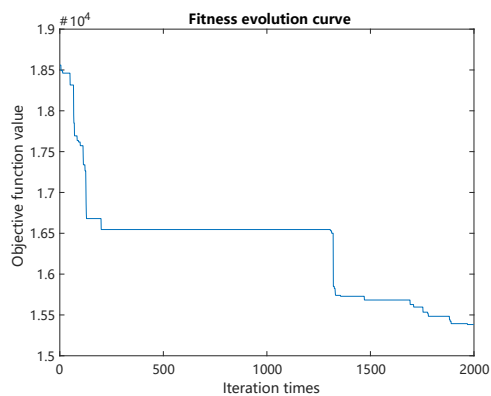
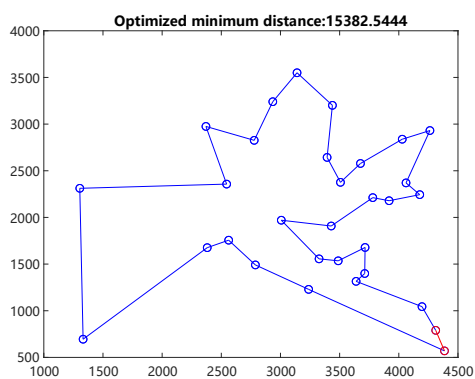
Gmax=3500; %最大迭代次数

```

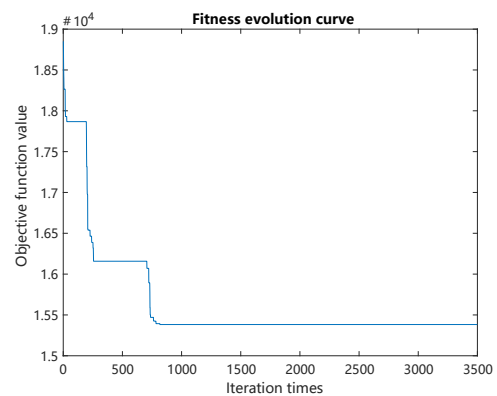
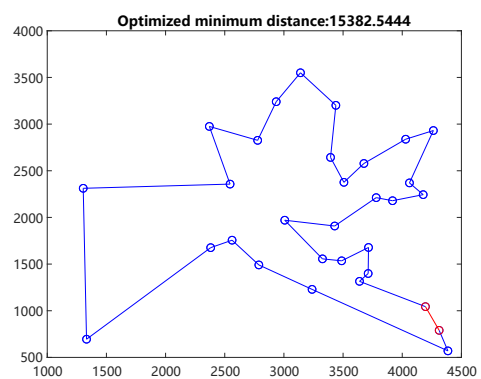
Fence 7-1

这种情况下进行多次尝试并且增加最大迭代次数利于得到全局最优：OMD出现过的最小值为15382.5444

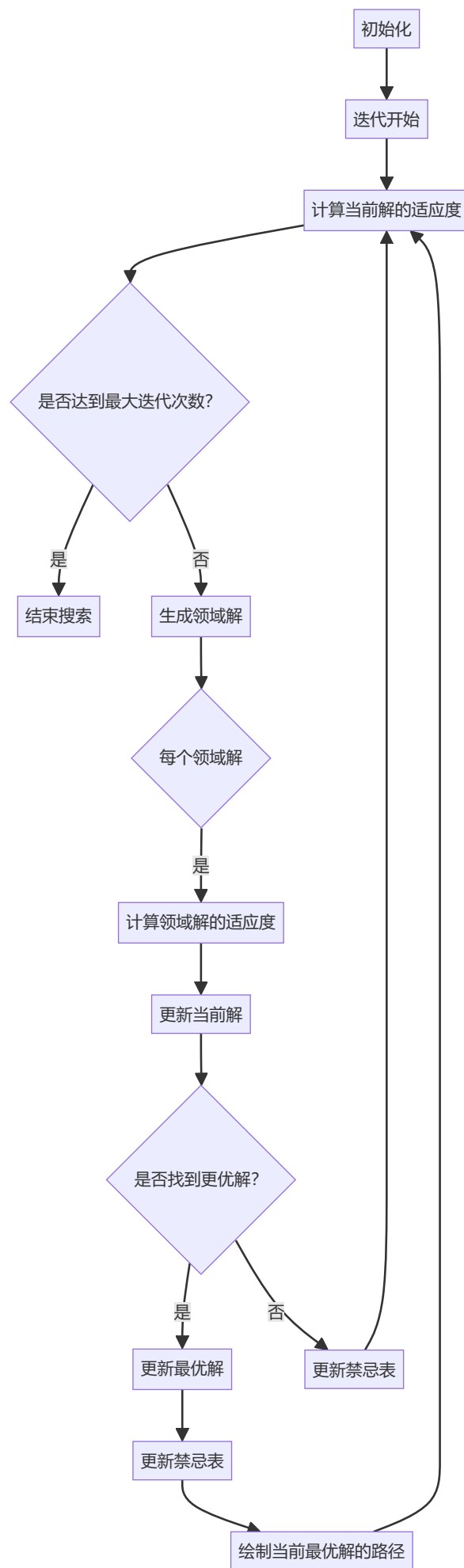
尝试26



尝试27



对于禁忌搜索，参数自适应要求记录当前解适应度并更新领域解：



8. 代码实现

```

clear all; % 清除所有变量
close all; % 关闭所有图形
clc; % 清除命令窗口

C = [1304 2312;3639 1315;4177 2244;3712 1399;3488 1535;3326 1556;...
      3238 1229;4196 1044;4312 790;4386 570;3007 1970;2562 1756;...
      2788 1491;2381 1676;1332 695;3715 1678;3918 2179;4061 2370;...
      3780 2212;3676 2578;4029 2838;4263 2931;3429 1908;3507 2376;...
      3394 2643;3439 3201;2935 3240;3140 3550;2545 2357;2778 2826;...
      2370 2975]; % 31个城市坐标

N=size(C,1); % TSP问题的城市数量，即31
D=zeros(N); % 距离矩阵初始化

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%计算城市间距离%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
for i=1:N
    for j=1:N
        D(i,j)=((C(i,1)-C(j,1))^2+...
                (C(i,2)-C(j,2))^2)^0.5;
    end
end

% 禁忌表初始化
Tabu = zeros(N); % 禁忌表
TabuL=round(0.5*(N*(N-1)/2)^0.5); % 禁忌长度为N*(N-1)/2的平方根的一半
Ca=ceil(3.7*N); % 邻域解的数量为3.7倍城市数量的上取整

% 产生初始解
S0 = greedyAlgorithm(C);

% 进行局部优化
S0 = localOptimization(D, S0);

bestsofar=S0; % 最佳解
BestL=Inf; % 最佳解距离初始化为无穷大
figure(1);
p=1;
Gmax=3500; % 最大迭代次数

% 开始迭代
while p<Gmax
    ALong(p)=func1(D,S0); % 计算当前解的目标函数值
    % 产生邻域解
    i=1;
    A=zeros(Ca,2); % 存储邻域解的索引及对应的操作
    % 随机生成邻域解
    while i<=Ca
        M=N*rand(1,2);
        M=ceil(M);
        if M(1)~=M(2)
            A(i,1)=max(M(1),M(2));
            A(i,2)=min(M(1),M(2));
            if i==1
                isa=0;
            else
                for j=1:i-1
                    if A(i,1)==A(j,1) && A(i,2)==A(j,2)

```



```

            isa=1;
            break;
        else
            isa=0;
        end
    end
end
end
if ~isa
    i=i+1;
end
end
end
end
% 评估邻域解
BestCaNum=floor(Ca/2); % 取邻域解的一半作为最优解数量
BestCa=Inf*ones(BestCaNum,4);
F=zeros(1,Ca);
for i=1:Ca
    CaNum(i,:)=S0;
    CaNum(i,[A(i,2),A(i,1)])=S0([A(i,1),A(i,2)]);
    F(i)=func1(D,CaNum(i,:));
    if i<=BestCaNum % 如果是前半部分的解
        BestCa(i,1)=i;
        BestCa(i,2)=F(i);
        BestCa(i,3)=S0(A(i,1));
        BestCa(i,4)=S0(A(i,2));
    else
        for j=1:BestCaNum % 如果是后半部分的解
            if F(i)<BestCa(j,2)
                BestCa(j,2)=F(i);
                BestCa(j,1)=i;
                BestCa(j,3)=S0(A(i,1));
                BestCa(j,4)=S0(A(i,2));
                break;
            end
        end
    end
end
end
% 对最优解进行排序
[JL,Index]=sort(BestCa(:,2));
SBest=BestCa(Index,:);
BestCa=SBest; % 更新最优解
% 更新当前解
if BestCa(1,2)<BestL
    % 如果找到更好的解
    BestL=BestCa(1,2);
    S0=CaNum(BestCa(1,1),:);
    bestsofar=S0;
    % 更新禁忌表
    for m=1:N
        for n=1:N
            if Tabu(m,n)~=0
                Tabu(m,n)=Tabu(m,n)-1;
            end
        end
    end
    Tabu(BestCa(1,3),BestCa(1,4))=TabuL;
else
    % 否则，尝试选择次优解

```

```

        for i=1:BestCaNum % 对于后半部分的解
            if Tabu(BestCa(i,3),BestCa(i,4))==0
                % 如果没有被禁忌
                S0=CaNum(BestCa(i,1),:);
                for m=1:N
                    for n=1:N
                        if Tabu(m,n)~=0
                            Tabu(m,n)=Tabu(m,n)-1;
                        end
                    end
                end
                Tabu(BestCa(i,3),BestCa(i,4))=TabuL;
                break;
            end
        end
    end
    ArrBestL(p)=BestL;
    p=p+1;
    % 绘制当前解路径
    for i = 1:N-1
        plot([C(bestsofar(i),1),C(bestsofar(i+1),1)],...
            [C(bestsofar(i),2),C(bestsofar(i+1),2)], 'bo-');
        hold on;
    end
    plot([C(bestsofar(N),1),C(bestsofar(1),1)],...
        [C(bestsofar(N),2),C(bestsofar(1),2)], 'ro-');
    title(['Optimized minimum distance:',num2str(BestL)]);
    hold off;
    pause(0.001);
end
BestShortcut=bestsofar;          % 最优路径
theMinDistance=BestL;           % 最小距离
figure(2);
plot(ArrBestL);
xlabel('Iteration times')
ylabel('Objective function value')
title('Fitness evolution curve')

function total_distance = func1(distance_matrix, solution)
    % 计算目标函数值，即路径总长度
    N = length(solution);
    total_distance = 0;
    for i = 1:N-1
        total_distance = total_distance + distance_matrix(solution(i),
solution(i+1));
    end
    % 添加从最后一个城市到第一个城市的距离，形成闭环
    total_distance = total_distance + distance_matrix(solution(N), solution(1));
end

function initialSolution = greedyAlgorithm(C)
    % 贪婪算法求初始解
    N = size(C, 1);
    initialSolution = zeros(1, N);
    visited = zeros(1, N);

    % 随机选择一个起始城市
    currentCity = randi(N);

```

```

initialSolution(1) = currentCity;
visited(currentCity) = 1;

% 选择最近的未访问城市进行连接
for i = 2:N
    minDistance = Inf;
    nextCity = -1;
    for j = 1:N
        if ~visited(j)
            distance = norm(C(currentCity, :) - C(j, :));
            if distance < minDistance
                minDistance = distance;
                nextCity = j;
            end
        end
    end
    initialSolution(i) = nextCity;
    visited(nextCity) = 1;
    currentCity = nextCity;
end
end

function improvedSolution = localOptimization(D, solution)
% 进行局部优化, 尝试交换城市顺序以减小路径长度
improvedSolution = solution;
N = length(solution);
improved = true;

while improved
    improved = false;
    for i = 1:N-1
        for j = i+1:N
            newSolution = swap(solution, i, j);
            if calculateDistance(D, newSolution) < calculateDistance(D,
improvedSolution)
                improvedSolution = newSolution;
                improved = true;
            end
        end
    end
end
end

function newSolution = swap(solution, i, j)
% 交换解中两个城市的顺序
newSolution = solution;
temp = newSolution(i);
newSolution(i) = newSolution(j);
newSolution(j) = temp;
end

function distance = calculateDistance(D, solution)
% 计算路径总长度
distance = 0;
N = length(solution);
for i = 1:N-1
    distance = distance + D(solution(i), solution(i+1));
end

```

```
% 添加从最后一个城市到第一个城市的距离，形成闭环  
distance = distance + D(solution(N), solution(1));  
end
```

Fence 8-1