# 2 数学模型

该问题为典型的TSP问题，TSP问题为NP-hard问题，因此采用遗传算法这一启发式算法进行求解。遗传算法需要首先构建由若干解构成的种群，解的多少称为种群规模。对每一个解采用实数编码，数字1到6分别代表A,B,C,D,E,F，生成1到6的随机数组成的城市来初始化种群。种群生成后，在满足进化终止条件（本文设置为进化代数，达到进化代数则终止进化）下，进行选择，交叉，变异操作，使种群中的解发生变化，通过对比并保存最优解来不断提高最优解的质量，即不断下降城市序列的总距离。

# 3 算法设计

C++代码

1. **初始化距离矩阵6×6，代表A,B,C,D,E,F六个城市之间的相互距离。**

int d[6][6] = {{0,15,20,12,8,6},

{15,0,17,9,11,24},

{20,17,0,19,13,8},

{12,9,19,0,17,6},

{8,11,13,17,0,8},

{6,24,8,6,8,0}}; //城市距离矩阵

1. **包含相应头文件，定义命名空间，定义最短距离与最短距离对应的最优城市序列，初始化最短距离为999999。**

#include <vector>

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

#include <math.h>

using namespace std;//包含头文件

int vertex\_num=6; //共有六个城市

int pop\_size; //种群规模

vector<vector<int> > population;//由各个访问城市序列组成的种群

double best\_solution=99999999; //所有访问城市序列里的最短距离

vector<int> best\_individual; //最优种群个体代表的城市排列;

1. **定义变异概率为0.5，交叉概率为0.8.**

double transform\_probability = 0.8;//交叉概率

double tubian\_probability=0.5;//变异概率

1. **初始化种群：采用实数编码，数字1到6分别代表A,B,C,D,E,F。生成1到6的随机数组成的城市来初始化种群。**

void initialize\_population(){

srand((int)time(0) + rand());//随机数种子

vector <int> sort\_vector;

for (int i = 1; i <= 6; i++){

sort\_vector.push\_back(i);

}

for (int i = 0; i < pop\_size; i++){

vector <int> temp = sort\_vector;

int cnt = 0;

while (cnt < vertex\_num\*2){

int first = rand()% vertex\_num;

int second = rand()% vertex\_num;

while (first == second){

second = rand()% vertex\_num;

}

int exchange = temp[first];

temp[first] = temp[second];

temp[second] = exchange;

cnt ++;

}

population.push\_back(temp);

//利用随机生成的1到6的城市排列来初始化种群。

}

cout<<endl;

cout<<"种群初始化完成"<<endl;

}

1. **计算个体的适应度值，即个体的适应度值为从序列第一个城市出发访问城市序列中其他所有城市后，返回出发城市的总距离**

double caculate\_individual\_distance(vector <int> individual){

double distance=0.0;

for (unsigned int i = 1; i < individual.size(); i++){

double edge\_length = d[individual[i]-1][individual[i-1]-1];

distance += edge\_length;

}

//总距离等于访问完所有城市序列后的距离与从序列最后一个城市返回出发城市的距离和

distance=distance+d[individual[5]-1][individual[0]-1];

return distance \* 1.0;

}

1. **使用轮盘赌法进行选择，轮盘赌法通过对种群中所有个体的适应度值求和，再利用每个个体的适应度值与适应度值的总和作为选择该个体进入下一代的概率。轮盘赌法可以保证适应度值更高的个体（距离更短的城市序列）进入下一代种群，进而保证了种群朝着更好的方向进化。**

void choose\_individuals(){

vector <double> individuals\_fitness;

vector <double> accumulate\_probability;

double population\_fitness = 0;

for (int i = 0; i < pop\_size; i++){

double cur\_distance = caculate\_individual\_distance(population[i]);

//根据当前个体与当前代数下最优个体的适应度值的大小进行比较，对最优个体进行更新。

if (cur\_distance < best\_solution){

best\_solution = cur\_distance;

best\_individual = population[i];

}

double cur\_fitness = 1.0 / cur\_distance;

individuals\_fitness.push\_back(cur\_fitness);

population\_fitness += cur\_fitness;

//计算所有解个体的适应度值的总和，便于后面按照每个解个体的适应度值求取被选择的概率。

}

for (int i = 0; i < pop\_size; i++){

double probability = individuals\_fitness[i] / population\_fitness;

if (i != 0){

accumulate\_probability.push\_back(accumulate\_probability[i-1] + probability);

//轮盘赌法，计算累积概率，对[0,1]进行划分

}

else{

accumulate\_probability.push\_back(probability);

}

}

srand((int)time(0) + rand());

vector <vector<int> > newPopulation;

for (int i = 0; i < pop\_size; i++){

//生成0到1的随机数，落在哪个区间就选择哪个个体，重复pop\_size次。

double randomly = rand()%10000 \* 0.0001;

for (int j = 0; j < pop\_size; j++){

if (randomly < accumulate\_probability[j]){

newPopulation.push\_back(population[j]);

break;

}

}

}

population.clear();

population = newPopulation;

}

1. **变异的概率为0.8，即在城市序列中调换两个城市的顺序，比如ABCDEF，变异操作可以使得A与C两个城市的位置调换，产生CBADEF。**

void individual\_tubian(){

srand((int)time(0) + rand());

for (unsigned int i = 0; i < pop\_size; i++){

srand((int)time(0) + rand());

int start\_index=rand()%6 ; //生成0到5的随机数上标

int end\_index=rand()%6;//生成0到5的随机数下标

while (start\_index>= end\_index){

start\_index=rand()%6;

end\_index=rand()%6;

}

int t=0;

double randomly = rand()%10000 \* 0.0001;

if(randomly<tubian\_probability){

//每个个体有突变概率的几率发生两个个体基因位的交换

t=population[i][start\_index];

population[i][start\_index]=population[i][end\_index];

population[i][start\_index]=t;

}

}

}

1. **交叉操作：使用顺序交叉法，设置变异概率，两个个体的对应区段进行交换，删除重复元素后依次补齐生成新的两个个体。交叉操作可以保证遗传算法的全局搜索性能。**

void individuals\_transform(){

srand((int)time(0) + rand());

//使用顺序交叉法

for (int i = 0; i < pop\_size; i++){

double start\_num = rand()%6;

double end\_num=rand()%6;

int start\_index=1;

int end\_index=5;

for (int j = 0; j < vertex\_num; j++){

if (population[i][j] == start\_num){

start\_index = j;

break;

}

}

//变异概率

double transform1 = rand()%10000 \* 0.0001;

if (transform1 < transform\_probability){

end\_num = rand()%6;

while (start\_num == end\_num){

end\_num = rand()%6;

}

for (int j = 0; j < vertex\_num; j++){

if (population[i][j] == end\_num){

end\_index = j;

break;

}

}

else{

int other = rand() % pop\_size;

while (other == i){

other = rand() % pop\_size;

}

for (int j = 0; j < vertex\_num; j++){

if (population[other][j] == start\_num){

if (j == vertex\_num - 1){

end\_num = population[other][j-1];

}

else{

end\_num = population[other][j+1];

}

break;

}

}

for (int j = 0; j < vertex\_num; j++){

if (population[i][j] == end\_num){

end\_index = j;

break;

}

}

}

if (start\_index > end\_index){

int temp = start\_index;

start\_index = end\_index;

end\_index = temp;

}

//将start\_index与end\_index区段内的部分进行逆序操作

vector <int> reverse;

for (int j = start\_index + 1; j <= end\_index; j++){

reverse.push\_back(population[i][j]);

}

int reverse\_index = reverse.size() - 1;

for (int j = start\_index + 1; j <= end\_index; j++){

population[i][j] = reverse[reverse\_index];

reverse\_index --;

}

}

}

1. **主函数部分：输入种群规模，进化代数参数，之后进行遗传算法操作。使用C++文件操作将运行结果输出至txt文档中，**

int max\_generation;

cout << "please input the size of population (e.g.: 300): ";

cin >> pop\_size; //输入种群规模

cout << endl;

cout << "please input the number of generation (e.g.: 10000): ";

cin >> max\_generation;//输入进化代数

cout << endl;

initialize\_population(); //种群初始化

ofstream output;

output.open("C:\\Users\\wangshengdong\\Desktop\\5.txt");

int generation = 0;

while (generation < max\_generation){//进化终止条件为进化代数

choose\_individuals(); //选择操作

individual\_tubian(); //变异操作

individuals\_transform();//交叉操作

cout <<"the best in current generation: "<<best\_solution<< endl;

output <<generation << " "<<best\_solution<< endl;//写入文件

generation ++;

}

system("pause");

}

# 4算法实现

1. **结果**

代数: 100

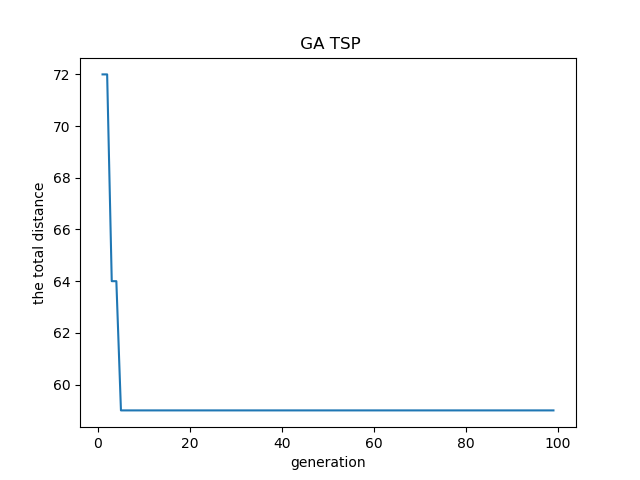
种群大小: 30

交叉概率: 0.90

变异概率: 0.50

最优解: 59.0

可以看到随着遗传算法的进化代数在10代之前，最短距离迅速收敛至59.0。这是由于该问题的规模较小，存在的可能解较少导致。绘图如下：



# 5结果分析

如图所示，在种群规模为30，进化代数为100代下，问题的最优解为4,1,6,5,3,2，用字母表示即为城市序列D,A,F,E,C,B此时总距离为59。说明遗传算法可以求得问题的一个较优解。

