计算智能

专 业 : 人工智能

班 级 ： 1班

姓 名 ： 吴南群

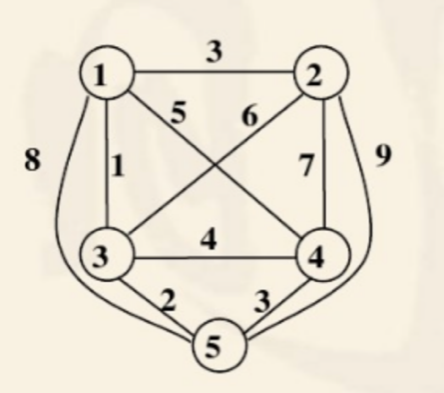
学 号 : 20214001002

1. **问题:**

作业1:

设计一个精准算法（能确保得到最优解）求解一个5个城市的对称TSP问题，要求分析：你的算法运行时间和算法复杂度分别是多少？如果城市数增加到16个（ulysses16数据集），你的算法能求解吗？如果可以，需要多少运行时间？有没有什么方法可以提升你的算法的时间效率并得到比较好的结果？

5个城市的距离信息如下：



|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ∞ | 3 | 1 | 5 | 8 |
| 3 | ∞ | 6 | 7 | 9 |
| 1 | 6 | ∞ | 4 | 2 |
| 5 | 7 | 4 | ∞ | 3 |
| 8 | 8 | 2 | 3 | ∞ |

求出最优路径和最短路径的值，给出算法的基本思路与步骤，给出源代码并做相应注释，给出程序运行截图。

ulysses16数据集(城市坐标点)：

NAME: ulysses16.tsp

TYPE: TSP

COMMENT: Odyssey of Ulysses (Groetschel/Padberg)

DIMENSION: 16

EDGE\_WEIGHT\_TYPE: GEO

DISPLAY\_DATA\_TYPE: COORD\_DISPLAY

NODE\_COORD\_SECTION

1 38.24 20.42

2 39.57 26.15

3 40.56 25.32

4 36.26 23.12

5 33.48 10.54

6 37.56 12.19

7 38.42 13.11

8 37.52 20.44

9 41.23 9.10

10 41.17 13.05

11 36.08 -5.21

12 38.47 15.13

13 38.15 15.35

14 37.51 15.17

15 35.49 14.32

16 39.36 19.56

EOF

最优路径为：

NAME : ulysses16.opt.tour

COMMENT : Optimal solution for ulysses16 (6859)

TYPE : TOUR

DIMENSION : 16

TOUR\_SECTION

1 14 13 12 7 6 15 5 11 9 10 16 3 2 4 8

-1

EOF

最短路径为72 ----73.9876

1. **求解**

电脑环境:

ubuntu 20.04 x86\_64

编译工具:

g++

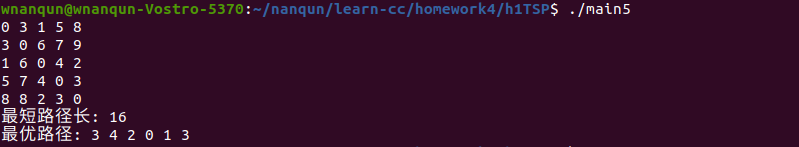
解法:

遗传算法-能够收敛-但不能保证每次得出最优答案

对于5个城市的解法:

代码:main\_5.cpp

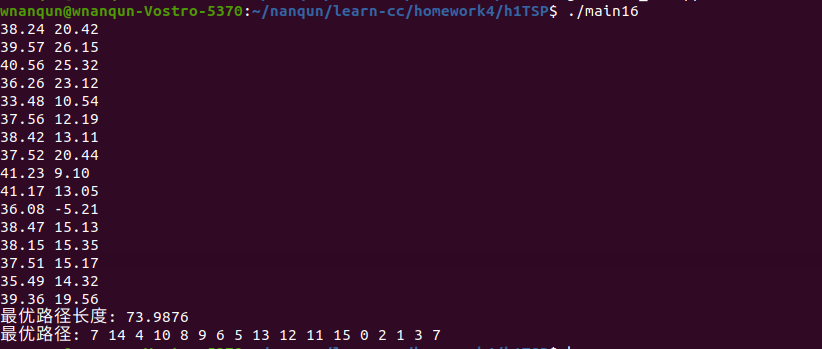
效果:



对于 16个城市的解法:

代码: main\_16.cpp

效果:



得到最优解

备注:由于不能提交zip文件，下面附代码:

**remead.md:**

数据集:

0 3 1 5 8

3 0 6 7 9

1 6 0 4 2

5 7 4 0 3

8 8 2 3 0

38.24 20.42

39.57 26.15

40.56 25.32

36.26 23.12

33.48 10.54

37.56 12.19

38.42 13.11

37.52 20.44

41.23 9.10

41.17 13.05

36.08 -5.21

38.47 15.13

38.15 15.35

37.51 15.17

35.49 14.32

39.36 19.56

**main\_5.cpp:**

#include<bits/stdc++.h>

using namespace std;

const int maxn=20;

const int maxm=1000;

int n=5; //城市数量

int M=100; //免疫个体数量

int T=50; //免疫代数

int Nnum=50; //克隆数量

int dist[maxn][maxn]; //记录城市间距离

struct cell

{

int path[maxn];

double fitness;

bool operator <(const cell x)const

{

return fitness<x.fitness;

}

}a[maxm],b[maxm];

vector<int>Path; //用于随机生成路径

void get\_path(int \*path);

void init();

double get\_fitness(int \*path); //计算适应度-长度

void copy(int \*a,int \*b,int num=n);

int main()

{

init();

int t=0;

while(t<T)

{

for(int i=0;i<M;i++)

{

a[i].fitness=get\_fitness(a[i].path);

}

sort(a,a+M); //排序

for(int i=0;i<M/2;i++)

{

for(int j=0;j<Nnum;j++)

{

copy(a[i].path,b[j].path);

if(j!=0)//变异

{

int p1=rand()%n,p2=rand()%n;

while(p1==p2)p2=rand()%n;

swap(b[j].path[p1],b[j].path[p2]);

}

b[j].fitness=get\_fitness(b[j].path);

}

sort(b,b+Nnum);

copy(b[0].path,a[i].path);

a[i].fitness=b[0].fitness;

}

for(int i=M/2;i<M;i++)

{

get\_path(a[i].path);

a[i].fitness=get\_fitness(a[i].path);

}

t++;

}

sort(a,a+M);

cout<<"最短路径长: "<<a[0].fitness<<endl;

cout<<"最优路径: ";

for(int i=0;i<n;i++)

cout<<a->path[i]<<" ";

cout<<a->path[0]<<endl;

}

//随机路径

void get\_path(int \*path)

{

random\_shuffle(Path.begin(),Path.end());

for(int i=0;i<n;i++)

{

path[i]=Path[i];

}

}

//初始化输入

void init()

{

//计算距离

for(int i=0;i<n;i++)

{

for(int j=0;j<n;j++)

{

cin>>dist[i][j];

}

}

srand(1314);//随机种子

for(int j=0;j<n;j++)

{

Path.push\_back(j);

}

for(int i=0;i<M;i++)

{

get\_path(a[i].path);

}

}

//获取路径长

double get\_fitness(int \*path)

{

double res=0;

for(int i=1;i<n;i++)

{

res+=dist[path[i-1]][path[i]];

}

res+=dist[path[0]][path[n-1]];

return res;

}

void copy(int \*a,int \*b,int num)

{

for(int i=0;i<num;i++)

{

b[i]=a[i];

}

}

**main\_16.cpp:**

#include<bits/stdc++.h>

using namespace std;

#define sq(x) (x)\*(x)

const int maxn=20;

const int maxm=1000;

int n=16; //城市数量

int m=100; //免疫个体数量

int T=50; //免疫代数

int Ncl=50; //克隆数量

double dist[maxn][maxn]; //记录城市间距离

struct cell

{

int path[maxn];

double fitness;

bool operator <(const cell x)const

{

return fitness<x.fitness;

}

}a[maxm],b[maxm];

vector<int>Path; //用于随机生成路径

void get\_path(int \*path);

void init();

double get\_fitness(int \*path); //计算适应度-长度

void copy(int \*a,int \*b,int num=n);

int main()

{

init();

int t=0;

while(t<T)

{

for(int i=0;i<m;i++)

{

a[i].fitness=get\_fitness(a[i].path);

}

sort(a,a+m);

for(int i=0;i<m/2;i++)

{

for(int j=0;j<Ncl;j++)

{

copy(a[i].path,b[j].path);

if(j!=0)//变异

{

int p1=rand()%n,p2=rand()%n;

while(p1==p2)p2=rand()%n;

swap(b[j].path[p1],b[j].path[p2]);

}

b[j].fitness=get\_fitness(b[j].path);

}

sort(b,b+Ncl);

copy(b[0].path,a[i].path);

a[i].fitness=b[0].fitness;

}

for(int i=m/2;i<m;i++)

{

get\_path(a[i].path);

a[i].fitness=get\_fitness(a[i].path);

}

t++;

}

sort(a,a+m);

cout<<"最优路径长度: "<<a[0].fitness<<endl;

cout<<"最优路径: ";

for(int i=0;i<n;i++)

cout<<a->path[i]<<" ";

cout<<a->path[0]<<endl;

}

//随机路径

void get\_path(int \*path)

{

random\_shuffle(Path.begin(),Path.end());

for(int i=0;i<n;i++)

{

path[i]=Path[i];

}

}

//初始化输入

void init()

{

double x[maxn],y[maxn];

for(int i=0;i<n;i++)

{

cin>>x[i]>>y[i];

}

//计算距离

for(int i=0;i<n;i++)

{

for(int j=0;j<n;j++)

{

dist[i][j]=sqrt(sq(x[i]-x[j])+sq(y[i]-y[j]));

}

}

srand(1314);//随机种子

for(int j=0;j<n;j++)

{

Path.push\_back(j);

}

for(int i=0;i<m;i++)

{

get\_path(a[i].path);

}

}

//获取路径长

double get\_fitness(int \*path)

{

double res=0;

for(int i=1;i<n;i++)

{

res+=dist[path[i-1]][path[i]];

}

res+=dist[path[0]][path[n-1]];

return res;

}

void copy(int \*a,int \*b,int num)

{

for(int i=0;i<num;i++)

{

b[i]=a[i];

}

}