PROJECT：Steiner Tree Problem

课程名称：**算法设计与分析** 任课教师：**张子臻**

年级	**17级**	**专业（方向）**	**软件工程**
学号	17343130	姓名	徐肯
电话	15057212086	Email	**979223119@qq.com**
开始日期	2019年6月5日	完成日期	2019年6月14日

Steiner Tree Problem简介

定义

Given an undirected distance graph G = (V, E, d) and a set S, where V is the set of vertices in G. E is the set of edges in G. d is the distance function which maps E into the set of nonnegative numbers and S∈V is a subset of the vertices of V.

The Steiner tree problem is to find a tree of G that spans S with minimal total distance on its edges.

其中，点集S称为正则点集，其余的点称为非正则点，被选入Steiner树中的非正则点称为Steiner点。

性质

Steiner最小树为NP-Complete问题，以下是关于图的Steiner最小树问题的性质，这些性质在后面的算法中会用到。

- **性质 1** 如果一个非正则点的度数为 1，那么该点一定不在 Steiner 最小树中。
- **性质 2** 如果一个正则点的度数为 1，那么与该点相邻的顶点一定在 Steiner 最小树中。
- **性质 3** 如果非正则点 V 的度数为 2，且与该点所关联的顶点为 X 和 Y，若边 (X,Y)存在，且距离（V,X）+（V,Y）> （X,Y），则非正则点 V 必定不在 Steiner 最小树中。
- **性质 4** 如果 Steiner 最小树存在，则 Steiner 点的数目最多不超过正则点的数目减 2。

思路

Steiner最小树问题要求找到一棵树，使得这棵树必须包含所有正则点，同时总距离最小。因为存在只包含正则点可能无法构成树，或者一些非正则点的加入可以使得总距离变小的情况，那么我们的任务就可以转化为对于非正则点的选取问题，即组合优化问题。

因此，最简单的思路应该是穷举了，计算选择非正则点的所有情况，但这样的组合问题的规模会呈爆炸性增长。因此，退而求其次，对于大规模的组合优化问题，可以引入近似算法来解决。近似算法虽然无法保证得到全局最优解，但可以在可接受的时间和空间范围内得到近似最优解。

算法介绍

在本次Project中，我使用了两种元启发式，一种是基于population的遗传算法，另一种是基于single-solution的禁忌算法，下面先对这两种算法做一个简单介绍。

遗传算法（Genetic Algorithm，GA）

遗传算法是一种模拟自然界中生物繁殖和进化过程中优胜劣汰、适者生存的全局搜索启发式算法。基本的算法流程如下

- 首先随机或者根据某些规则产生一组初始解（种群）
- 从初始种群（初代）开始，对于每一代中的个体根据其适应度值进行选择、交叉、变异等操作
- 重复上述步骤不断产生下一代，直到满足预先设置的终止条件

遗传算法通过上述步骤，实现从初始解向最优解逼近的自适应过程，很好的保证了搜索空间的多样性，而且通过变异也可以防止陷入局部最优解，因此有比较好的效果。在实际使用中，要解决以下问题：

- 如何对问题的解进行编码，即用如何个体的“染色体”表示一个解
- 如何初始化种群，初始种群的优劣一定程度上影响算法的性能
- 适应度函数的确定，即如何判断一个个体的优劣
- 选择、交叉、变异操作的实现

禁忌搜索（Tabu Search，TS）

与遗传算法中的群体概念不同，禁忌搜索从单个可能的解开始进行搜索。禁忌搜索由局部搜索算法（Local Search，LS）发展而来，但与局部搜索每次只选择更好的领域解不同的是，禁忌搜索通过引入禁忌表这一概念，标记近期已经访问过的局部最优解，从而在下一次迭代中避开这些解，甚至接收一个比当前差的邻域解来避免陷入局部最优。因此，通常会得到比局部搜索更好的效果。

算法的基本流程为：

- 产生一个初始可行解
- 对于每个当前的解，对其邻域进行搜索，找到一个合适的解（可能更好也可能变差），进入下一次循环
- 不断迭代，直到满足终止条件

在实际使用中，有以下概念或问题需要解决：

- 禁忌表，禁忌长度的确定
- 领域（候选集）的产生规则
- 特赦规则，有时为提高算法性能或者当前领域解都被禁止了，会允许某个解被解禁

算法实现

基于上述算法的原理，并结合Steiner最小树的一些性质，下面先简单介绍程序中两种算法的一些共性实现：

解的表示和初始解的产生：

对于解的表示，由于这是一个组合问题，我使用了二进制的编码方式，即每一位对应图中的一个点，为1则表示该点在生成树中，为0则不在，显然，所有正则点对应的位一定为1。

同时，根据之前提到的性质，我们通过检查图中结点的度数，就可以初步确定一些非正则点是否会在最后的最小生成树中，对于不能确定的点，随机使其为1或者0即可。遗传算法初始产生多个解，禁忌搜索产生一个解。

判断解的优劣

每一种二进制编码的解都表示了原图的一个子图，通过应用最小生成树算法得到的距离和来判断解的优劣，即总距离越小越好。程序中使用Kruskal算法来计算最小生成树。

以下是不同算法的特殊实现：

遗传算法

个体适应度

为简便，直接取解对应的最小生成树大小（即生成树越小，适应度越好）

精英保留

在交叉、变异过程中，由于不确定性，有时会失去种群中的最优个体，也就是说，在进化产生越来越多优良个体的同时，可能会失去最好的个体，这是算法所不希望的。精英保留策略确定了每代种群中最好的一小部分个体会被保留，不会被交叉产生的替换，也不会变异，直接进入下一代。

程序中简单实现了该功能，即对每代个体按照其适应度排序，在后面进化时保留前面最好的 ELITISMNUMBER 个。

```
#define ELITISMNUMBER 10

//筛选出精英个体（按fitness排序）
void GA::elitism() {
    sort(population, population + POPUSIZE - 1, [](Individual indiv1, Individual indiv2)
    {return indiv1.fitness < indiv2.fitness; });
}

//精英保留
for (int i = ELITISMNUMBER; i < POPUSIZE; i++) {
    population[i] = nextPopulation[i];
}
```

选择

采用锦标赛选择的策略，即从种群中随机挑选一定数量的个体，从中选取最优的作为繁殖后代的双亲之一。

```
//锦标赛选择
Individual GA::select() {
    int pos;
    int bestfit = INF + 1;
    for (int i = 0; i < TOURNAMENTSIZE; i++) {
        int r = randInt(0, POPUSIZE - 1);
        if (population[r].fitness < bestfit) {
            pos = r;
            bestfit = population[r].fitness;
        }
    }
    return population[pos];
}
```

交叉

采取较为简单的两点交叉方式，同时设置了交叉率这一参数。如果交叉后生成了无效解，则取消交叉。

```
//两点交叉
void GA::crossover(const Individual& p1, const Individual& p2, Individual& baby1, Individual&
baby2) {
    baby1 = p1;
    baby2 = p2;

    double p = randFloat();
    if (p > CROSSEROVER_RATE)
        return;

    int point1 = randInt(1, csLen - 2);
    int point2 = randInt(1, csLen - 2);
```

```
    if (point2 < point1) {
        swap(point1, point2);
    }

    for (int i = 0; i < csLen - 1; i++) {
        if (i < point2 && i > point1)
            continue;
        baby1.chromosome[i] = p2.chromosome[i];
        baby2.chromosome[i] = p1.chromosome[i];
    }
    //若交叉产生了无效解，换原
    if ((G.generateSubGraph(baby1.chromosome, csLen)).evaluate() == INF)
        baby1 = p1;
    if ((G.generateSubGraph(baby2.chromosome, csLen)).evaluate() == INF)
        baby2 = p2;
}
```

变异

对于非精英个体的染色体进行变异，根据变异率对每位二进制做和1的异或操作。

```
//变异
void GA::mutate() {
    double x;
    for (int i = ELITISMNUMBER; i < POPUSIZE; i++) {
        Individual temp = nextPopulation[i];

        for (int j = 0; j < csLen; j++) {
            if (G.terminals[j])
                continue;
            x = randFloat();
            if (x < MUTATION_RATE) {
                nextPopulation[i].chromosome[j] ^= 1;
            }
        }
        //如果变异后成了无效解，那么取消变异
        if ((G.generateSubGraph(nextPopulation[i].chromosome, csLen)).evaluate() == INF)
            nextPopulation[i] = temp;
    }
}

```

禁忌搜索

产生邻域解（候选解集合）

随机更改一位。

```
void TS::generateNeighbors()
{
    for (int i = 0; i < NEIGHBORSIZE; i++) {
        neighbors[i] = solution;
        int k = -1;

```

```
        do {
            if(k != -1)
                neighbors[i].isChoose[k] ^= 1;
            do {
                k = randInt(0, nodeNum - 1);
            } while (determine[k] != -1);
            neighbors[i].isChoose[k] ^= 1;
        } while ((neighbors[i].value = (G.generateSubGraph(neighbors[i].isChoose,
nodeNum)).evaluate()) == INF);
    }
}
```

根据禁忌表找到合适的解

先对领域解排序，从好到坏选择，如果找到不在禁忌表中的，就返回该解并将其加入禁忌表。如果都在禁忌表，就进行特赦，程序简单的选择特赦最好的那个解。

```
Solution TS::findBestNeighbor()
{
    sort(neighbors, neighbors + NEIGHBORSIZE - 1, [](Solution a, Solution b)
    {return a.value < b.value; });

    int flag;
    for (int i = 0; i < NEIGHBORSIZE; i++) {
        flag = 1;
        for (auto it = tabuList.begin(); it != tabuList.end(); it++) {
            if (neighbors[i] == it->first) {
                flag = 0;
                break;
            }
        }
        if (flag) {
            tabuList.insert(make_pair(neighbors[i],TABULENGTH));
            return neighbors[i];
        }
    }

    //特赦
    //cout << "特赦" << endl;
    for (auto it = tabuList.begin(); it != tabuList.end(); it++) {
        if (neighbors[0] == it->first) {
            it->second = TABULENGTH;
            break;
        }
    }
    return neighbors[0];
}
```

更新禁忌表

每次迭代，将禁忌表中的解的禁忌长度-1，同时删除为0的解。

```
void TS::updateTabuList()
{
    for (auto it = tabuList.begin(); it != tabuList.end(); ) {
        it->second--;
        if (it->second == 0) {
            it = tabuList.erase(it);
        }
        else {
            it++;
        }
    }
}
```

具体代码见源文件。

实验结果

实际运行结果（以运行p401.stp中的数据为例）

> **说明**
>
> - 由于p401.stp中数据较大，可以使得算法的变化过程更明显。
> - 以下为Visual Studio 2017中的运行结果。
> - 由于近似算法的不确定性，不保证下一次运行结果和截图中的一致
> - 运行时最好将数据文件放在同一文件夹中
> - 在遗传算法中，由于我在初始化种群时限制了很多条件，并且种群规模的固定为100，初始化的时候可能比较慢

遗传算法

终止条件

- 进化1000代
- 种群中最优的适应度过了200代还没有变化（虽然后面可能会出现更优解）

C:\Users\97922\source\repos\Steiner Tree\Debug\Steiner Tree.exe

```
--------------------Minimum Steiner Tree Problem---------------

                      1、Genetic Algorithm
                      2、Tabu Search
                      3、Quit

----------------------------------------------------------------

Enter your choice: 1
Please enter the name of instance file: p401.stp
read file successfully! nodeNum = 100, edgeNum = 4950, terminalNum = 5
Please wait a minute,  we are prepare the initial population for GA...
Best individual in initial generation: 940
Best individual in generation 1: 873
Best individual in generation 2: 788
Best individual in generation 3: 679
Best individual in generation 4: 679
Best individual in generation 5: 635
Best individual in generation 6: 574
Best individual in generation 7: 555
Best individual in generation 8: 488
Best individual in generation 9: 488
Best individual in generation 10: 488
Best individual in generation 11: 488
Best individual in generation 12: 468
Best individual in generation 13: 468
Best individual in generation 14: 425
Best individual in generation 15: 414
Best individual in generation 16: 408
Best individual in generation 17: 396
```

C:\Users\97922\source\repos\Steiner Tree\Debug\Steiner Tree.exe

```
Best individual in generation 481: 158
Best individual in generation 482: 158
Best individual in generation 483: 158
Best individual in generation 484: 158
Best individual in generation 485: 158
Best individual in generation 486: 158
Best individual in generation 487: 158
Best individual in generation 488: 158
Best individual in generation 489: 158
Best individual in generation 490: 158
Best individual in generation 491: 158
Best individual in generation 492: 158
Best individual in generation 493: 158
Best individual in generation 494: 158
Best individual in generation 495: 158
Best individual in generation 496: 158
Best individual in generation 497: 158
Best individual in generation 498: 158
Best individual in generation 499: 158
Best individual in generation 500: 158
Best individual in generation 501: 158
Best individual in generation 502: 158
Best individual in generation 503: 158
We finished! the optimal minimum value is 158
The steiner point is: 8 31 34 36 45 52 55 65 87
Time spent from begin to the optimal value found: 81.896 seconds



Enter your choice:
```

禁忌搜索

禁忌搜索我直接让它迭代了1000次

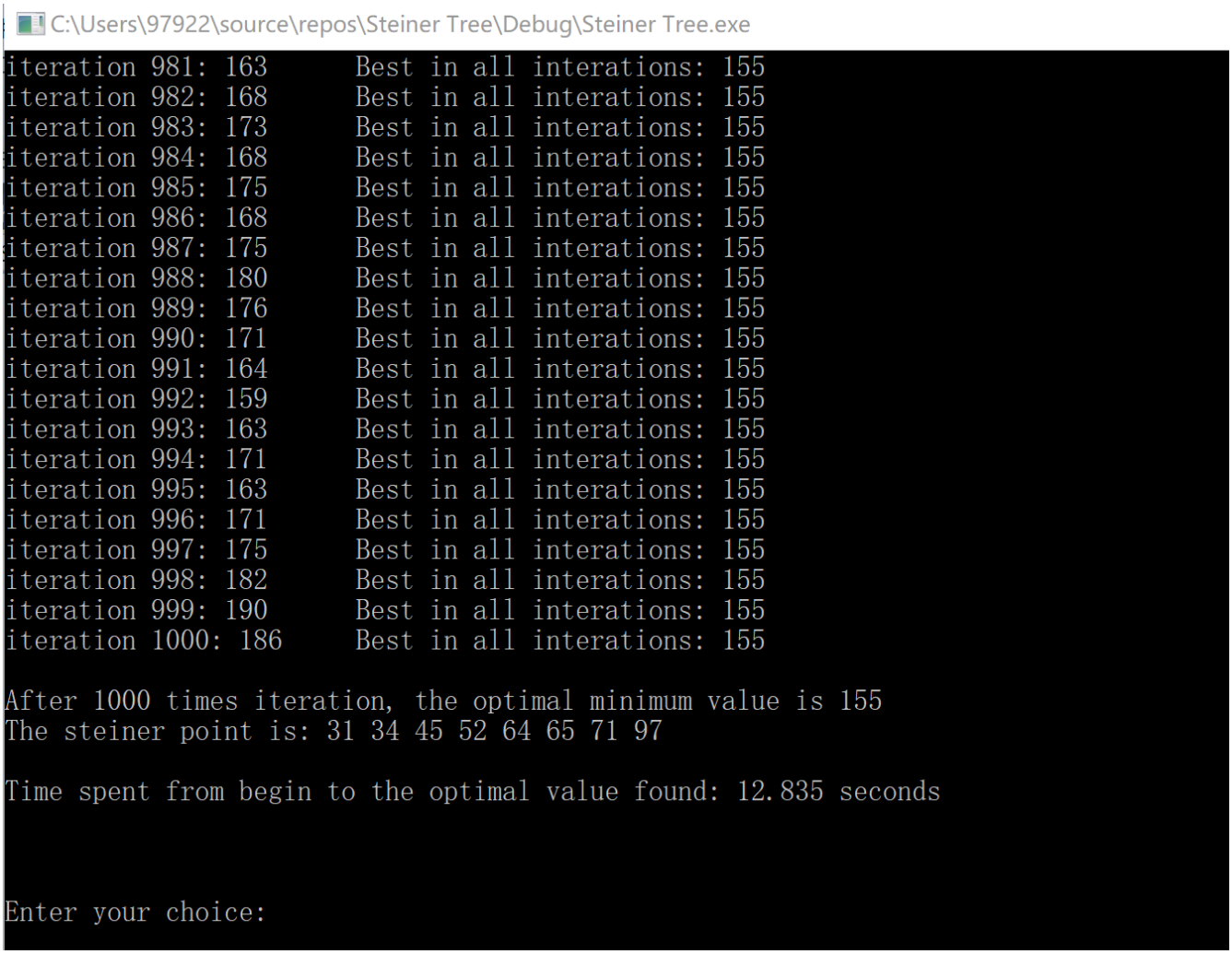
C:\Users\97922\source\repos\Steiner Tree\Debug\Steiner Tree.exe

```
--------------------Minimum Steiner Tree Problem---------------

                    1、Genetic Algorithm
                    2、Tabu Search
                    3、Quit

------------------------------------------------------------


Enter your choice: 2
Please enter the name of instance file: p401.stp
read file successfully! nodeNum = 100, edgeNum = 4950, terminalNum = 5
Iteration 0: 1515       Best in all interations: 1515
iteration 1: 1433       Best in all interations: 1433
iteration 2: 1344       Best in all interations: 1344
iteration 3: 1278       Best in all interations: 1278
iteration 4: 1219       Best in all interations: 1219
iteration 5: 1169       Best in all interations: 1169
iteration 6: 1094       Best in all interations: 1094
iteration 7: 1052       Best in all interations: 1052
iteration 8: 1008       Best in all interations: 1008
iteration 9: 980        Best in all interations: 980
iteration 10: 934       Best in all interations: 934
iteration 11: 897       Best in all interations: 897
iteration 12: 850       Best in all interations: 850
iteration 13: 814       Best in all interations: 814
iteration 14: 783       Best in all interations: 783
iteration 15: 754       Best in all interations: 754
iteration 16: 729       Best in all interations: 729
iteration 17: 695       Best in all interations: 695
iteration 18: 670       Best in all interations: 670
```



可以看到，每次迭代的解都不同，155为历史最优解。

观察对于这组数据的两个算法结果，发现禁忌算法快很多，因为遗传算法每一代都要进行大量计算。

其余文件的结果

> **说明**
>
> - 以下结果综合了两种算法的结果
> - 由于近似算法的不确定性，每次运行的结果均会有所不同，以下取的都是我在多次运行后得到的最好结果
> - 时间的计算为单纯的算法时间。并且当最终的最优解第一次出现就停止计时，即后续的迭代不算入时间，因为没有产生更优解。
> - 遗传算法有时会比较慢，由于产生初始解时做了限制，且要求产生可行解，同时种群较大。

name	|V|	Result	Running Time
cc3-4p	64	2338	5.6s
cc3-4u	64	23	0.298s
cc6-2p	64	3279	7.8s

name	|V|	Result	Running Time
cc6-2u	64	32	0.852s
hc6p	64	4003	2.413s
hc6u	64	39	0.384s
b01	50	82	0.722s
b02	50	83	0.511s
b03	50	138	0.32s
b04	50	59	5.392s
b05	50	61	0.202s
b06	50	122	0.353s
b07	75	111	0.414s
b08	75	104	1.995s
b09	75	220	0.536s
b10	75	86	9.099s
b11	75	88	7.784s
b12	75	174	1.798s
b13	100	165	40.395s
b14	100	238	90.884s
b15	100	318	3.263s
b16	100	130	10.084s
b17	100	131	13.062s
b18	100	218	1.974s
antiwheel5	10	7	0.03s
design432	8	9	0.019s
oddcylcle3	6	4	0.019s
oddwheel3	7	5	0.02s
se03	13	12	0.028s
P401	100	155	12.835s

分析与总结

分析

对于上述的实验结果来说，两种算法的实现应该还算比较成功的，但是我在算法中对于一些参数的设置，例如遗传算法的种群大小以及交叉变异的概率，禁忌搜索的禁忌长度以及邻域空间的产生等方面还是比较粗糙的，如果通过精心设计，并根据数据量的大小自适应的调整，可能会有更大的可能得到最优解。

对比我实现的两种算法，对于同一数据，由于算法的不确定性，每次运行两个算法都有好有坏，有时可以快速得到某个很好的解，但有时需要很久甚至得不到。遗传算法有时产生初始种群需要很长时间，但经过多次进化，结果还是不错的；禁忌算法收敛速度较快，基于单体使得其运行速度也很快，但可能会失灵，容易陷入局部最优解，可能是我设计的不够好。

让我感受最深的是，在遗传算法中，一开始我是完全随机的生成一些初始解，但这样解收敛的很慢，而且效果也不好。后来，我根据Steiner最小树的性质，在生成初始解的时候就确定下来是否选择一些非正则点。通过这样的修改，得到的效果有了明显提高。

总结

经过本次的Project，我实现了上述两种原理上相似，但实现方法不同的近似算法，相比于之前都是得到固定的精确解，这次的编程给了我不一样的体验。虽然过程也没有一帆风顺，但也没有很难，只要在开始写代码前搞清楚它们的实现原理即可。

本次实验增强了我对于近似算法，特别是元启发式算法的理解，这些近似算法能够在可接受的时间内得到近似最优解，解决一些NP难度的组合最优化问题，在实际应用中很有价值。