2022-2023 秋季学期

《人工智能大作业》

作业成绩

评 分	

姓　　名： 姜文渊

学　　号： 20205988

学　　院： 计算机科学与工程学院

专　　业： 计算机科学与技术

班　　级： 计算机 2001 班

课程名称： 《人工智能》

任课教师： 张恩德

基于 Christofides 启发式的 TSP 问题近似解求解与 2OPT 优化的程序设计与实现

题目要求：设计一条旅行路线，所有城市均去过且只去过一次，要求所行走的路线距离之和越短越好（TSP 问题不需要考虑可达性，两两之间均可达)

摘　要

本文针对 TSP 问题，采用 Christofides 启发式算法，在多项式时间内求出代价小于等于最优解代价 1.5 倍之内的一个近似解。并在近似解的基础上采用了 2-OPT 优化方法和自行研究的优化算法。最终得到了较为优秀的结果，和精确解十分接近。文章主要包括 TSP 相关背景介绍、编程环境、技术路线、项目运行结果以及项目心得。

项目运行得到的结果与精确解十分近似，并且程序可以在多项式时间内得到确定的、准确的结果。在使用 C++编程语言实现克里斯托菲德算法的同时，可以促进图论相关知识的复习。最后采用 python 编程语言进行了 TSP 路径的绘制。项目综合可行性高、算法时间复杂度低。在 MacBook Pro 上运行项目，可以在 15ms 内得到代价为 155.153 的最终结果。（单位以经纬度下的欧氏距离之和计算）

【关键词】 Christofides **启发式算法**；TSP；Kruskal；**带权带花树**

目 录

一、题目相关背景介绍

1.1 TSP 问题定义

旅行商问题(Traveling salesman problem)，简称为 TSP 问题，是 1959 年提出的数学规划问题，TSP 属于典型的 NP 完全问题。其语言描述为：在一个具有 n 个城市的完全图中，旅行者希望进行一次巡回旅行，或经历一次哈密顿回路，可以恰好访问每一个城市一次，并且最终回到出发城市。而这次巡回旅行的总费用为访问各个城市费用的总和，故旅行者同时希望整个行程的费用是最低的，求这个路线的排列策略？

TSP 问题的组合解有 N!种组合，随着城市数量 N 的规模增加，组合数将呈指数级别递增，故使用穷举法将会面临组合爆炸问题，因此 TSP 属于 NP 完全问题。

1.2 电子文档打包文件名及文件列表

/Cpp_Codes，内附 city.csv,和 C++源代码；

/Python_Codes，内附 city.csv，和 python 源代码；

/报告，内附此份报告 word 及 pdf、以及最终运行结果绘制的图像文件。

1.3 编译执行环境与命令

本次项目的编程在 MacBook Pro 上完成，IDE 使用的是 VSCode。

/usr/bin/clang++ -fdiagnostics-color=always -std=c++17 -stdlib=libc++ -g /Users/johnjiang/Downloads/Cpp_Codes/AI_TSP/main.cpp -o /Users/johnjiang/Downloads/Cpp_Codes/AI_TSP/main

Python 绘制图像使用的 IDE 是 Pycharm CE。

/Users/johnjiang/PycharmProjects/draw_TSP_route/venv/bin/python /Users/johnjiang/PycharmProjects/draw_TSP_route/main.py

二、技术路线

对于 TSP 问题，我们可以考虑通过求解近似解+优化的方法，也可以考虑求解其精确解。但对于本题 N=34 的情况，显然分支限界等求解精确解的复杂度过大，我决定采用在多项式时间内采用启发式求解近似解+优化的技术路线。

在写代码的过程中，我先尝试采用了 MST 启发式求解近似解，但效果不是很好，结果约为 260（直接以经纬度为单位的欧氏距离之和）。MST 启发式从数学上已经证明，当费用函数满足三角不等式时，该算法找出的哈密顿回路产生的总费用，不会超过最优回路的 2 倍。但这个结果显然不能使我满意。于是我转向了更加复杂、更加难以实现的 Christofides 算法，可以证明，Christofides 算法所生成的解相对最优解有 3/2 的近似比，因此该算法可以满足在多项式时间内得到一个较优的解，便于后续优化。

在实现了 Christofides 算法后，得到的结果为 160.465（直接以经纬度为单位的欧氏距离之和）已经很接近最优解了。因此我在得到局部最优解之后，采用了

2OPT 优化方法。得到了几种优化结果：158.92、158.463、157.643。我又采用 python 进行 TSP 路径的绘制，总结了绕远路的一些共性问题，并尝试采用了一种根据 TSP 问题自创的优化方法。最终经过优化，得到了目前最好的结果，155.153。

2.1 Christofides 启发式 TSP 问题近似解求解

2.1.1 克里斯托菲德算法是什么？

克里斯托菲德斯算法 (Christofides algorithm) 是旅行商问题在度量空间（即距离对称且满足三角不等式）上的一个近似算法。该算法可以保证相对最优哈密尔顿回路长度有 3/2 的近似比。尼科斯·克里斯托菲德斯 (Nicos Christofides) 于 1976 年首次发表了这个算法，故以他的名字命名之。截至 2017 年，这一算法仍然是一般性旅行商问题的算法中近似比最好的结果。

令 G= (V,w)是旅行商问题的一个实例。即, G 是一顶点集 V 上的一个完全图，函数 w 给 G 的每条边指定了一个非负实边权。依三角不等式，对每三个顶点 u,v 和 x 都有关系 w(uv) +w(vx) ≥w(ux).

克里斯托菲德斯算法可以被描述为如下伪代码。

1.构造图上的最小的生成树 T

2.令 O 为原图顶点集中在 T 上度数为奇数的顶点集。根据握手定理，O 中有偶数个顶点

3.构造点集 O 在原完全图上的最小完全匹配 M

4.将 M 和 T 的边集取并，构造重图 H ，满足其每个顶点都是偶数度的

5.H 可以形成一个欧拉回路

6.将前一步得到的图改造为一个哈密尔顿回路：只需要跳过前一步欧拉回路中重复的顶点即可（这个步骤又称作选取近道，"short-cutting"）。图示如下：

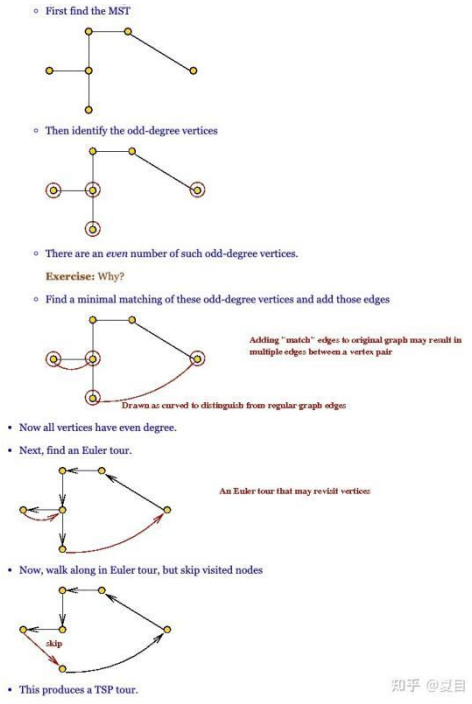


图 1 Christofides 算法的实现示意图

因此，我们的总技术路线就是先利用 Kruskal 算法求得 34 个城市的最小生成树 MST，并维护每个点的度数，以便于求得奇度顶点集合。再把以奇度顶点为顶点建立完全图，提供给带权带花树寻找一般图最小权值匹配。得到最小权值匹配后，从任意一个点作为起点，尝试寻找欧拉回路，这个过程采用的是 dfs 寻路。在得到一条欧拉回路后，我们只需要沿着这条路径走，删掉重复的点即可得到汉密尔顿回路，即最终的 TSP 路径。到这里，Christofides 算法已完成，后续我们根据 2-OPT 算法，每次随机选择任意的两个城市进行交换，比较交换前后的总代价，如果变小了，则说明交换是有效的。在此之后，我又尝试了自行优化，方法为：将一个点从路径中提取出来，并插入到后续的可能的空隙当中。这个方法能够打乱原来的阵型，达到跳出局部最优解的效果，从而有可能达到全局最优解。

2.1.2 克里斯托菲德算法的优势？

克里斯托菲德算法所生成的解相对最优解有 3/2 的近似比。下面给出简要证明：令 C 为图上的最优旅行商回路、令 O 为原图顶点集中在 T 上度数为奇数的顶点集。根据握手定理，O 中有偶数个顶点。注意到，删除 C 上的一条边将产生原图上的一棵生成树，其总权重必然不小于最小生成树，也就是 $w(T) \leq w(C)$。接下来，给 O 中的顶点编号，编号顺序依据回路 C 上的顶点顺序。再将 C 做划分，得到两个边集：其一包含所有起始点在生成树上有奇数度的边，另一个则包含所有起始点在生成树上有偶数度的边。这两个边集各对应于 O 上的一个完全匹配，且匹配的总边权不超过边集的总边权。

因为两个边集是 C 的划分，二者中的某一个至多有 C 总边权的一半，因而其对应匹配的总边权不超过 C 总边权的一半。因为没有比最小完全匹配更小的匹配，所以 $w(M) \leq w(C)/2$。T 和 M 的权重总和也就是欧拉回路的权重总和，也就是至多 $3w(C)/2$。由于做选取近道操作并不增加权重，所有最终输出的总边权也不超过 $3w(C)/2$.

2.1.3 克里斯托菲德算法的具体实现——Kruskal 求最小生成树

克里斯托菲德算法的第一步即为求出以所有城市为顶点的完全图对应的最小生成树。我选择使用 Kruskal 算法，其涉及到了并查集+贪心的思想。

克鲁斯卡尔算法是通过依次在所有边中挑选出最小权值的边来相加，直到添加的边等于结点数-1，因此我们要先将边集按照权值从小到大排序。对边的结构体进行运算符重载即可调用 sort()进行排序，如图二所示。

这里值得指出的是，我在编写 Kruskal 函数时犯了一个错误，如果两个点的代表元素不是一个元素，则应该向最小生成树的 vector 集合 MST 中添加一条{e[i].u,e[i].v,len}的边，而不是两个代表元素连一条边。

生成最小生成树 MST 的同时，我们可以维护所有顶点的度数，便于统计奇度顶点的个数，以及后续奇度顶点匹配。

```
struct edge{
    int u,v;
    double len;
    friend bool operator<(const edge & edge1,const edge & edge2){
        return edge1.len<edge2.len;
    }
}e[M+1];
```

图 2 边集定义以及运算符重载

2.1.4 克里斯托菲德算法的具体实现——奇度顶点一般图最小权值匹配

这个部分是整个克里斯托菲德算法最难的部分了。首先讲一下图的匹配问题。

首先需要指出一些图的定义，如下所示。

匹配：对于图 G=(V,E)的一个匹配 M 是边集 E 的子集，其中两两不共用顶点；最大匹配：边数最多的匹配；完美匹配：包含原图所有点的匹配。

最优匹配：又称为带权最大匹配，是指在带有权值边的二分图中，求一个匹配使得匹配边上的权值和最大。

交替路：从一个未匹配点出发，依次交错经过非匹配边、匹配边形成的路径。

增广路：一个末尾是未匹配点的交错路径。这样的路径可以通过翻转整条路上边的选择状态获得一个长度恰好增加 1 的匹配。

交错树：从未匹配点 开始寻找增广路时，交错路径组成的树。其中，我们称向根方向匹配的点为 S 点（黑点，偶点），背向根方向匹配的点为 T 点（白点，奇点）。

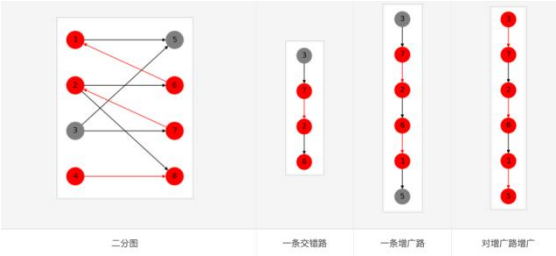


图 3 无向二分图某时刻对增广路进行增广的示意图

因此，我们需要利用增广路可以通过反转获得长度恰好增加 1 的匹配这条性质，不断进行增广，从而获得最大权完美匹配。

这个问题的一个经典算法是 KM 算法。其核心思想是通过松弛操作，让左边的点的顶标（vertex labeling）从其能够连到的最大边权不断降低，右边的点顶标从 0 不断增加，当两边的点顶标相加等于边权时，即为一个“等边”。与匈牙利算法类似，每次增广先在“等边”构成的一张二分子图上进行尝试匹配，如果匹配不到，则松弛（将标准放低一些）后继续尝试；直至所有边都成为等边仍未找到增广，算法结束。松弛的目的是使一些新边成为等边，加入二分子图，让左部点能尝试匹配的点更多；

但是对于本题，我们需要进行的是一般图匹配。对于一般图，就没有二分图这么好的性质了。但唯一的不同是，一般图中可能有奇数长度的环，下简称奇环。在奇环上的每一个点连出去的非匹配边，都能够成为某个增广路上的边，这导致在增广的过程中难以判断如何翻转。

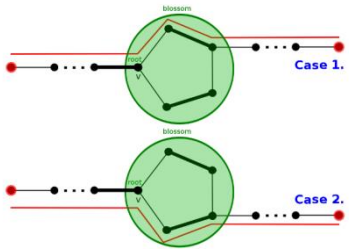


图 4 增广路经过奇环的两种形式

我们发现，奇环具有了所有黑点需要具备的性质：任意点连出去的非匹配边

都可以在从根开始的交错路上；奇环的根（花托）是向根方向匹配的（满足黑点条件）。于是，整个奇环可以缩为一个点，称为“花”。将花缩成一个点后，环上所有点变成黑点，并可以在交错树上向外延伸。这个性质就便于我们不断翻转增广路径了。于是一般图最小权值匹配即可用带权带花树去实现。

因此，通过类似 KM 算法的办法，不断去扩充增广路径。每次扩充路径时，如果不能满足条件，则进行松弛操作，降低期望值，以达到让更多的点能够加入匹配的目的。至于求最小权值匹配，只需要用很大的数字 INF 减去原权值即可。

```
pair<int,ll>calc(){
    int i,cnt=0;ll s=0;
    while(augment())++cnt;
    //不断尝试进行增广，记录增广次数
    for(i=1;i<=n;i++)if(mat[i]>i)s+=g[i][mat[i]].w/2;
    //对于每个点，检查是否成功匹配，且匹配的对方编号大于自己（那么对于另一个人肯定对方编号小于自己）匹配一次即可
    //sum每次加上的是匹配权值的一半（因为初始化的时候乘以二了）
    return mp(cnt,s);
}
```

图 5 不断进行增广操作直到求得最大权匹配（本题为最小权值匹配）

2.1.5 克里斯托菲德算法的具体实现——寻找欧拉回路与汉密尔顿回路

寻找欧拉回路采用的是 dfs 的思想。每次访问一个点，就将其加入欧拉回路的 vector 中，在与其邻接的所有点中，按顺序进行 dfs。需要指出的是，从不同的起始点出发进行 dfs，得到的欧拉回路有可能不同。因此为了后续能得到较为优秀的结果，遍历所有城市作为起始点的情况。而汉密尔顿回路只需要沿着欧拉回路走一遍，其中重复的点不再予以考虑即可。

三、程序运行结果与优化操作

3.1 程序运行结果

Christofides 算法运行得到的结果为 160.465，经过 2-OPT 优化后可以达到 157.976，经过 my-OPT 优化后可以达到 155.153。程序运行时间仅需 15ms 左右。

```
johnjiang — 99×50
Launching: '/Users/johnjiang/Downloads/Cpp_Codes/AI_TSP/main'
Working directory: '/Users/johnjiang/Downloads/Cpp_Codes/AI_TSP'
1 arguments:
argv[0] = '/Users/johnjiang/Downloads/Cpp_Codes/AI_TSP/main'
Christofides算法初步结果为:160.465
正在进行2-opt优化！
第1次
160.283
159.978
157.976
第2次
161.608
160.874
第3次
160.22
第4次
160.874
159.854
157.976
第5次
159.854
第6次
157.976
第7次
159.854
第8次
157.976
经过2opt优化后的最小代价为:157.976
正在进行my_opt优化！
第1次
156.431
155.153
经过自行研究的优化方法优化后，最小代价为: 155.153
cost of final_path = 155.153
最终路径为:
LANZHOU->YINCHUAN->XIAN->ZHENGZHOU->TAIYUAN->HUHEHAOTE->SHIJIAZHUANG->BEIJING->TIANJIN->HAERBIN->CH
ANGCHUN->SHENYANG->JINAN->HEFEI->NANJING->SHANGHAI->HANGZHOU->TAIBEI->FUZHOU->NANCHANG->WUHAN->CHAN
GSHA->GUANGZHOU->XIANGGANG->AOMEN->HAIKOU->NANNING->GUIYANG->CHONGQING->CHENGDU->KUNMING->LASA->WUL
UMUQI->XINING
最终TSP路径为:
12,7,14,13,9,4,8,5,6,0,1,3,10,16,15,17,20,28,26,23,19,24,29,31,32,33,30,25,22,18,27,21,2,11
此程序的运行时间为0.015694秒！
Process exited with status 0

Saving session...
...copying shared history...
...saving history...truncating history files...
...completed.

[进程已完成]
```

图 6 程序运行结果

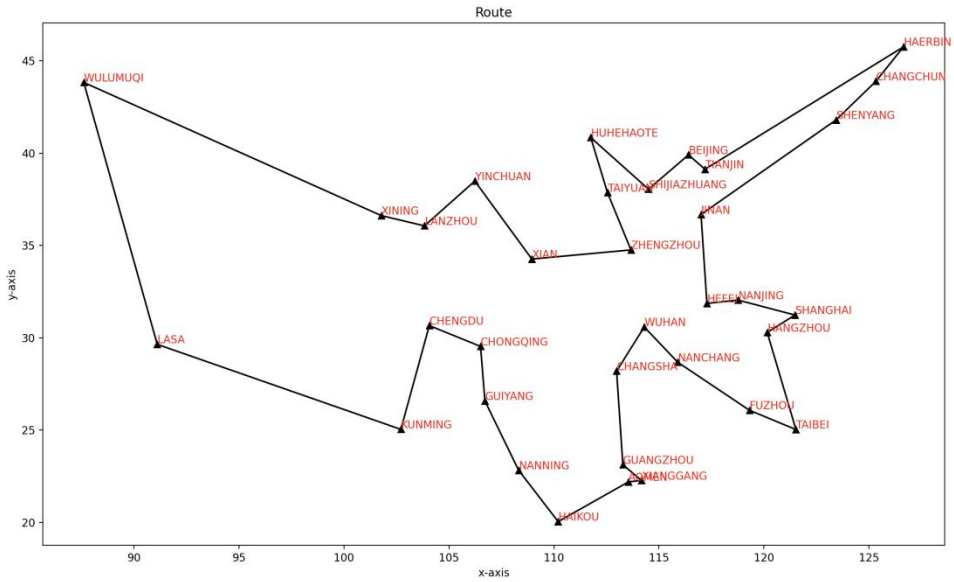
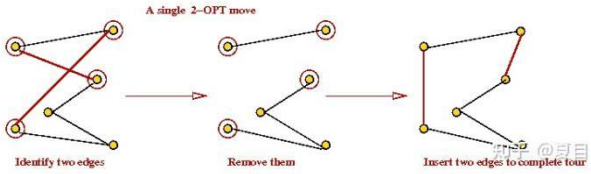


图 7 轨迹绘制结果

3.2 优化操作

在得到初步结果后，发现有些地方出现了绕远路的情况。于是采用了 2-OPT 的优化思路进行了第一次的优化。K-OPT 属于局部搜索启发式方法。思路为找到两条边和对应的端点并交换端点，若更优则保存。



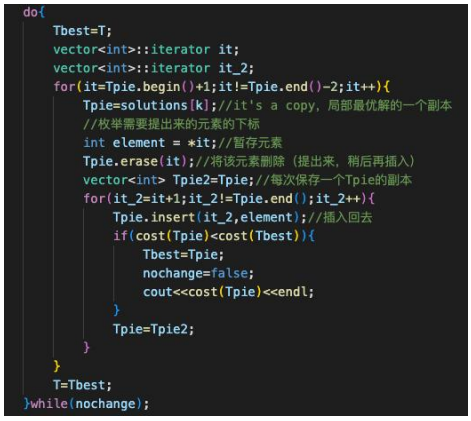
算法2(通过所有可能的交换找到最佳游览)

```
T = 一些tour(初始解)
noChange = true
repeat
   Tbest = T
   for all T中可能的边-对
      T' = 交换边-对中的端点获得的tour
      if T' < Tbest
          Tbest = T'
          noChange = false
      endif
   endfor
   T = Tbest
until noChange
return T
```

图 8 遍历所有可能的 2-OPT 算法伪代码

经过程序的调试和总结，我发现 2-OPT 的优化仍然保持了整体路径的稳定性。因此它优化到 157 左右就找不到更优秀的局部最优解了。

此时我把 2-OPT 交换两个点带来的两条边代价比较，扩展为“增加三条边、删除三条边，并进行代价比较”的更普遍的情况。具体实现方法是：将原队列中的一个城市点保存并从队列中删除，并尝试插入到后续的所有可能的空隙中。这种大胆的操作可以破坏原有序列的稳定性，从而产生“突变”而取最小值可以将“变异”后的优秀特征进行保留和“遗传”。从而保留最优值。



```
do{
    Tbest=T;
    vector<int>::iterator it;
    vector<int>::iterator it_2;
    for(it=Tpie.begin()+1;it!=Tpie.end()-2;it++){
        Tpie=solutions[k];//it's a copy, 局部最优解的一个副本
        //枚举需要提出来的元素的下标
        int element = *it;//暂存元素
        Tpie.erase(it);//将该元素删除（提出来，稍后再插入）
        vector<int> Tpie2=Tpie;//每次保存一个Tpie的副本
        for(it_2=it+1;it_2!=Tpie.end();it_2++){
            Tpie.insert(it_2,element);//插入回去
            if(cost(Tpie)<cost(Tbest)){
                Tbest=Tpie;
                nochange=false;
                cout<<cost(Tpie)<<endl;
            }
            Tpie=Tpie2;
        }
    }
    T=Tbest;
}while(nochange);
```

图 9 提取元素并插入到后续位置的 my-OPT 优化算法

至此，技术路线已经圆满完成。

四、心得体会

完成这个项目遇到了许多的问题。首先在最初选择解决方案的时候，综合考虑了最近邻算法、MST 启发式、Christofides 启发式、禁忌搜索、模拟退火、遗传算法、群智能算法。但由于刚打完 ICPC，最后决心趁着热乎劲继续用 C++去实现这个项目。最后我发现通过启发式算法进行近似解求解是可以在多项式时间内完成的。这就把令人头大的 NPC 问题转化为了一个易于理解，一步步可以实现的子问题。并且，多项式时间内可以处理的问题，其求解方法一旦确定，对于给定的输入一定会产生确定的输出，这一点让我坚定了选择克里斯托菲德算法。而当我做完项目，其他同学需要十几秒乃至几分钟运行完项目，而我的项目仅需 15ms 时，让我觉得这条技术路线是正确的。

在实现过程中，我首先尝试了 MST 算法，发现虽然实现起来十分简单，但结果只能保证最优解 2 倍之内，最终结果也确实有 260，非常大。因此在询问同学们得到的结果后，我决定向更难的启发式算法进军。克里斯托菲德算法在整体实现上，Kruskal 求 MST、Euler 回路、Hamilton 回路这些都是十分易于实现的。而求最小匹配的部分，由于求出有 14 个奇度顶点（一开始还求错了，求出 20 个顶点，导致结果一直出错），暴力枚举匹配复杂度会很大，因此我去论坛上寻找了各种资料，并按顺序学习了匈牙利算法、KM 算法、带花树算法、LCA（最近公共祖先）、带权带花树算法。这个过程让我收获了很多。在 ACM 训练期间，由于二分图匹配这种题型很少涉及，因此一直没有进行查缺补漏。这个过程遇到的问题就是图论相关知识需要进行理解。

最后是优化部分遇到的问题。在一开始得到结果的时候，遇到了路径打叉、绕远路的情况。于是考虑了 2-OPT 优化。但发现优化后到 157.976 之后就无法跳出局部最优解了。其实包括欧拉回路的求解也出现了以不同的起点多次运行局部搜索，可以跳出局部最优解。下面的图片是我根据当时的一些中间结果，总结出来的几种需要优化的情况。包括打叉、绕远路、无法跳出局部最优解。最后经过尝试，发现需要打破原有的序列顺序，进行带有猜测性质的选择操作、交换操作。

总的来说，本次项目对我的编程能力有了很大的提升。TSP 这个 NPC 问题被转化为若干个较易实现的图论操作，便于具体编程实现。在实现 Christofides 算法后，我没有止步，而是学习如何在此基础上进行优化，对发现的问题进行了总结归纳，并尝试了创新性的优化算法。

本次项目也有些遗憾，虽然这类近似解求解方法运行速度和运行效果都很好，但是没有尝试去实现如群智能算法那一类的人工智能启发式算法来解决 TSP 问题。并且 C++语言实现起来，代码量相较于其他同学还是会多一些。不过，本次项目让我广泛搜集资料并进行学习，在不断摸索中一点点进步，对日后的人工智能方向的学习可以算作一个很好的开始。感谢老师看到这里。

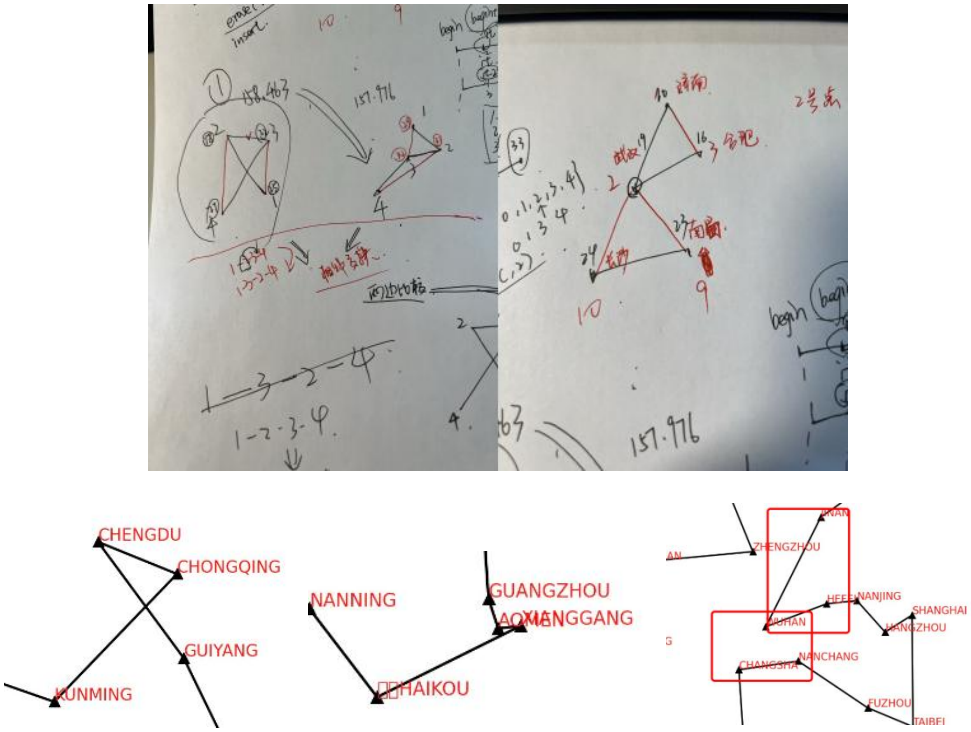


图 10 项目过程中发现的需要进行优化的几种问题

五、参考文献

[1]C++读写 CSV 文件
https://blog.csdn.net/weixin_43531632/article/details/122281033
[2]最小生成树 Kruscal 算法、并查集去环（详细图解，C++）
https://blog.csdn.net/Kyrie_irving_kun/article/details/113820315
[3]匈牙利算法详解
https://blog.csdn.net/lemonxiaoxiao/article/details/108672039
[4]Kurt Mehlhorn et al., Implementation of O(nmlogn) weighted matchings in general graphs: the power of data structures
https://dl.acm.org/doi/pdf/10.1145/944618.944622
[5]OI-wiki，一般图最大匹配
https://oi-wiki.org/graph/graph-matching/general-match/
[6]wenruo，KM 算法详解+模板
https://www.cnblogs.com/wenruo/p/5264235.html
[7]从匈牙利算法到带权带花树——详解对偶问题在图匹配上的应用
https://potassiumwings.github.io/2021/09/10/from_hungary_to_blossom_tree/