北京工业大学

2022 - 2023 学年 第1学期

信息学部 计算机学院

课程名称：	数据结构课程设计		
报告性质：	实验报告		
学号：	20090121	姓名：	陈立标
任课教师：	王众	课程性质：	学科基础必修课
学分：	2	学时：	60
班级：	200702	成绩：	
教师评语：			

需求分析(X)	根据题中需求，提供功能划分说明	
设计(L,S,流)	逻辑结构、存储结构设计、算法描述	
使用说明	界面是否友好	
总结(G)	是否感悟有收获	
摘要(Z)	考察文字抽象能力	
格式(m,p)	是否有目录、页号	

2022 年 11 月 18 日

摘要

本文所做题目为校园导游系统，要求设计实现一个校园导游系统，展示一个校园平面图，将用户选择的若干个场所都到达且仅到达一次，而且所走路程最短，将结果以图形方式展示给用户，并存入文件。

在语言使用方面，本文使用 Python+PyQt5 完成主体内容编写，同时结合 HTML 语言对文本进行更精致的展示。

在数据结构选择方面，本文选择**图结构**进行实现。将每一个目的地抽象图中的一个顶点，每两个目的地之间的路线抽象为图中的一条边，采用**邻接矩阵**进行存储、操作。

在算法选择方面，本文采用代表最优化算法的**回溯算法**、**动态规划**以及代表启发式算法的**遗传算法**进行实现。最优化算法诸如回溯算法、动态规划本质都为穷举+剪枝，优点在于能够取得全局最优解，缺点在于旅行商问题为 NP 难问题，穷举时间复杂度为 O(n!)，点数量达到一定值时难以在用户能够容忍时间内得到解；启发式算法通过已知知识寻找局部最优解，优点在于即使点数量很多，也能很快地完全运算，缺点在于可能会陷入局部最优，难以求得全局最优解。

在功能实现方面，本文(1)实现使用**爬虫**获取北工大 **47 个地点**真实经纬度数据，构建图网络模型；(2)使用 PyQt5 绘制 UI 界面并支持点击图片选中目的地、动态显示到达每个场所时间，提供预估总时间，展示详细路线文字描述；(3)实现一键路径优化，通过**多线程**智能地筛选出减少目的地点后的结果；(4)实现按选择点顺序给出最短路径；(5)实现设置功能，使用户能够**个性化设置**自己喜欢的界面展现方式。

除此之外，本文使用类继承、类方法重写等技巧，细分多个模块、多个文件进行编写，提高了代码复用与可读性。同时使用 Git 进行版本管理，同步上传 GitHub 以便查看个人进度、回滚之前版本。

目录

1. 需求分析

1.1. 功能分析

1.1.1. 旅行商问题

旅行商问题，给定若干城市和城市间距离，求解访问每一座城市一次并回到起始城市的最短回路。具体到课设情景，某同学选择校内多个地点，求解到达每个地点一次且仅一次并回到起始地点的最短回路。

从图论的角度来看，该问题实质是在一个带权完全无向图中，找一个权值最小的 Hamilton 回路。由于该问题的可行解是所有顶点的全排列，时间复杂度为 $O\backslash left(n!\backslash right)$,随着顶点数的增加，会产生组合爆炸，是一个 NP 完全问题。

对于旅行商问题的解决，现在主要分为两大类：最优化算法和启发式算法。

以深度优先搜索、回溯、动态规划为代表的最优化算法本质均为穷举所有可能选出全局最优解，其优化均以最大化剪枝效率为目标。最优化算法最大的问题，同时也是旅行商问题存在的意义便是其无法在多项式时间内求得解，意味着如果点数过多，该类算法的效率十分低，无法在用户能够容忍的时间范围内求得解。具体到本题，经过测试，剪枝程度最高的动态规划在规划 20 个目的地时便难以为继，而本文设计了 47 个点，显然经典算法无法胜任如此大规模的计算。但考虑到日常使用中用户目标点一般不会超过 10 个，因此最优化算法在时间上和启发式算法相差并不多，其精确求解的优点也得以凸显。

以遗传算法、粒子群算法、Hopfield 神经网络为代表的启发式算法是目前解决大规模旅行商问题的首选算法。启发式算法是基于直观或经验构造的算法，在可接受的花费（指计算时间和空间）下给出待解决组合优化问题每一个实例的一个可行解，其优点和缺点都显而易见：大规模下运行快、容易陷入局部最优而无法求得全局最优解。图 1 为本人参加校赛数学建模使用 Hopfield 神经网络求解旅行商问题的训练过程示例，此图可以代表绝大多数启发式算法的训练思路：随机初始化序列，通过某种已知信息(多为仿生信息)对序列进行迭代优化，在迭代足够多轮次后序列趋于平稳，此时的结果即为所求得的可行解。后文会着重对本文使用的遗传算法进行介绍。

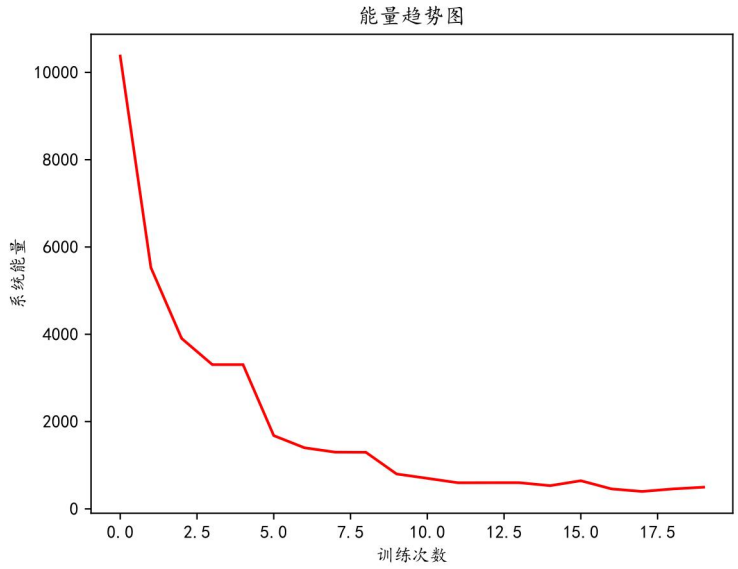

图 1 启发式算法效果展示

本文对上述两类算法中均有涉及，使用回溯、动态规划和遗传算法实现了对旅行商问题的求解，并能够智能选择使用哪种方式求解，同时开放设置允许用户自行选择使用何种求解方式。

1.1.2. 原始数据获取

题目要求使用真实数据展示出至少 10 个场所，本文选取校内 47 个地点进行展示。对于原始数据的获取本文基于 selenium 自动化工具编写爬虫程序，在百度地图坐标拾取系统[1]中获得所有地点的经纬度。获取示例如图 2 所示，首先在搜索框搜索想要获取的地点，在审查功能中可以看到能够通过 HTML 元素定位到该地点的坐标经纬度，流程图如图 3 所示。

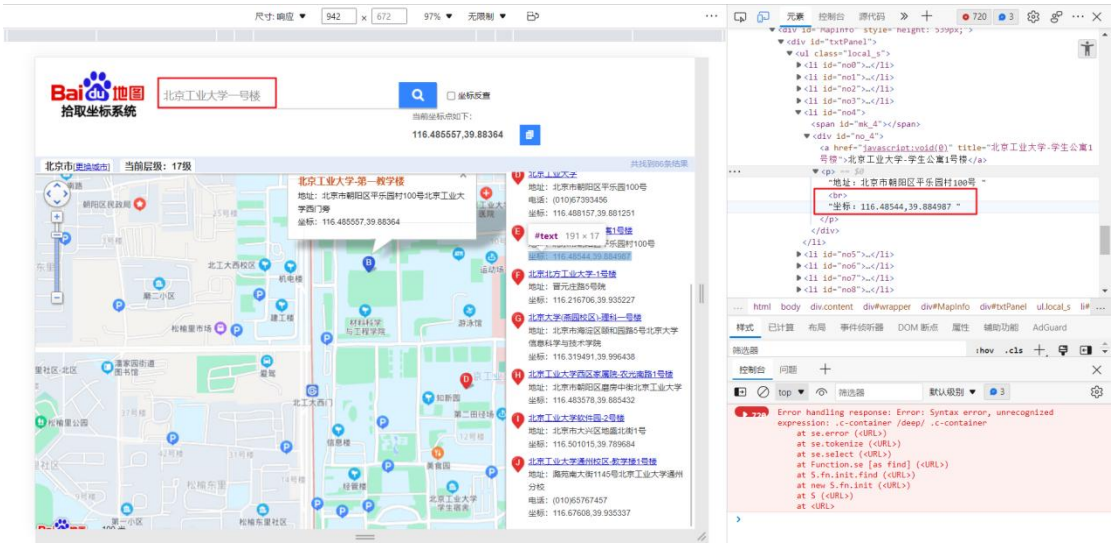

图 2 百度地图拾取坐标系统

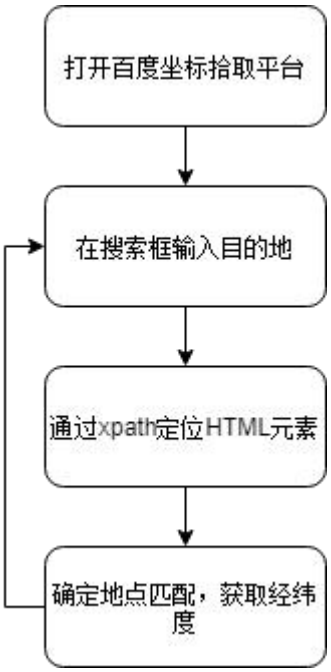

图 3 爬虫流程图

1.1.3. 路径优化

题目建议对总时长过长安排不下的结果，可以舍弃某些目的地以达到简化路径的目的。在无用户主观信息的情况下，本文对路径的简化以最大化减少路径总长度为目标。

假设用户此次选择 N 个目的地，在程序详细提示了总路径长度、所需时间后，若用户认

为所需时间过长，便可要求程序简化路径，程序简化思路为：遍历所有结点，计算删去该节点后其他所有结点的 TSP 结果，选择最优的结果进行展示。可以看出计算 N-1 个结果的流程完全一致且相互独立，因此本文采用 Python 多线程同时计算 N-1 个结果并在其中选择最优的结果进行展示。

1.2. 数据处理

wiki 百科中对于旅行商问题的描述如图 4 所示，可以看到旅行商问题被广泛界定为完全无向图中的问题，即每两个结点之间均有边存在。但具体到该题，在初始邻接矩阵设计中，显然在学校两边的建筑能够直达并不符合图的设计规范(应当通过多个中间建筑再到达)。

> TSP can be modelled as an undirected weighted graph, such that cities are the graph's vertices, paths are the graph's edges, and a path's distance is the edge's weight. It is a minimization problem starting and finishing at a specified vertex after having visited each other vertex exactly once. Often, the model is a complete graph (i.e., each pair of vertices is connected by an edge). If no path exists between two cities, adding a sufficiently long edge will complete the graph without affecting the optimal tour.

图 4　wiki 百科对 TSP 问题的描述

为了解决上述问题，本文设计的原始图模型如图 5 所示。这样的图显然比完全无向连接图更加贴近于实际。为了符合旅行商问题的规范，本文在获取当前邻接矩阵的前提下，通过 Floyd 算法获取每两个点之间的最短路径，同时递归存储所有前驱结点构成矩阵。

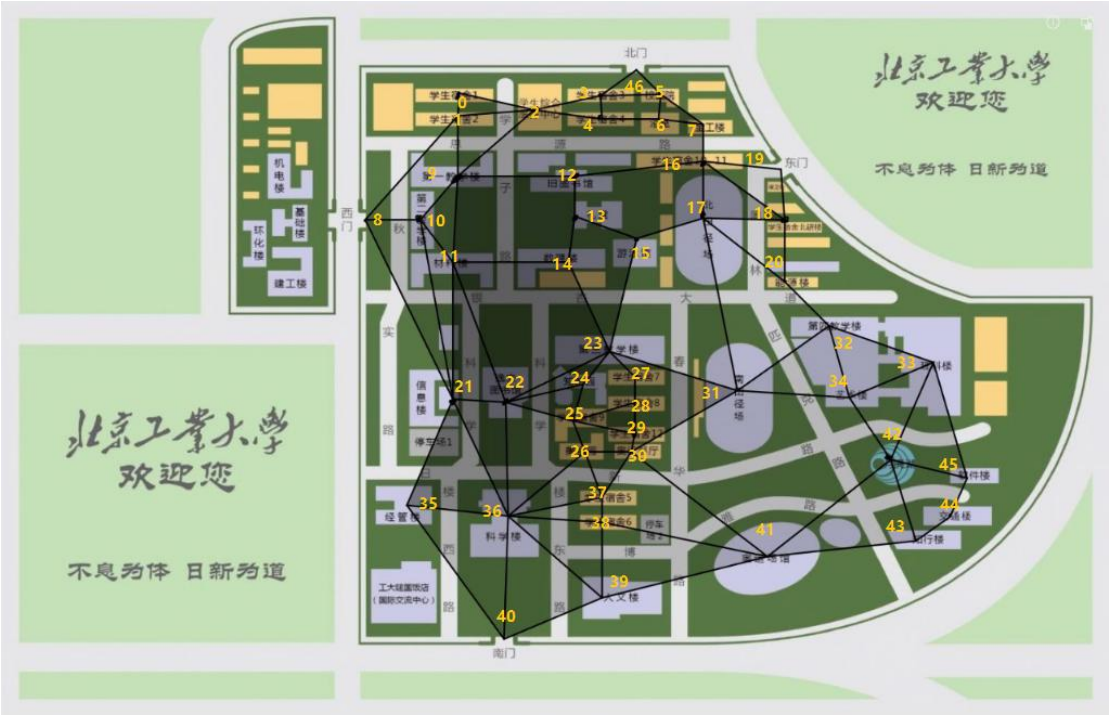

图 5 邻接图模型

在实际运算过程中，TSP 类以 Floyd 矩阵作为邻接矩阵进行计算，通过输入若干目标地点编号，先输出简单路线，即仅包含目标地点的路线，再通过 Floyd 的前驱结点矩阵找到每两个点之间的最短路径，这样不仅符合了 TSP 问题广泛认可的完全无向连接，还能够使得路线更加详细具体。

1.3. 用户界面设计

本文采用 PyQt5 开发图形界面，主界面如图 6 所示

图 6 程序主界面

1.3.1. 主界面设计

占程序大部分位置的校园地图来源于学校官网，是一个具有交互功能的区域，鼠标点击地图上某个地点即可选中，再次点击即可取消选中。同时右侧信息栏会以文字形式显示选中的信息。在选择完毕地点后，可以在“模式选择”中选择两种模式：TSP 模式和按顺序模式。前者即为题目要求旅行商算法的实现，后者为当用户希望严格按照所选路径顺序行动时，程序给出每两个点之间的具体路线。随后点击“生成路径”即可显示结果，演示图如图 6 所示。

图 6 运行结果演示

首先程序会在地图上绘制出完整路线，既包括所选的目的地(深色点)，也包括每两个目的地之间途径的地点（浅色点），对于每个目的地，弹出气泡提示目的地名称、预计到达时间，为了防止气泡挡住路线，初始化气泡透明度为 0.6，如图 6 下部气泡所示，当鼠标放置

在某个气泡上时，气泡透明度为 1，如图 6“北操”气泡所示。同时在右侧会展示总距离、预计总用时、简单路径、详细路径。

若用户认为当前所用时间过长，可以选择点击左侧“简化路线”按钮，程序会智能舍去一个点使得剩下的点总距离最小，效果如图 7 所示。路线中略去了“北操”目的地，同时右侧显示出了相关的信息。用户仍可以继续点击“简化路线”以进一步获得更精简的路线，可以预见的是下一步将舍去“理科楼”。

图 7 简化路线效果

若用户在查询路径时还未出发，可以在准备出发时点击“出发”按钮，此时页面中的气泡会刷新预计到达时间。所有操作结束后，用户可以点击“清空选择”以清空当次所有操作，重新开始导航地点的选择。本次图片结果和右侧信息栏输出结果均会存储于本地，具体路径可在后续设置界面个性化设置。

除此之外，界面上方的字体采用艺术手写体、图片和信息框边缘采用阴影效果等提高了界面美观度。

1.3.2. 设置界面设计

本程序为用户提供了个性化设置界面，如图 8 所示。

图 8 用户设置界面

第一行为背景图片设置，内置了三种背景，其中后两种为北京工业大学校园景色做模糊处理后的样子，其中一个的效果图如图 9 所示。第二行为默认模式选择，两种模式在主界面设计中已经介绍。第三行为默认算法设置，其中智能选择为大于等于 15 个点选用遗传算法、小于 15 个点选用动态规划算法。在其他设置中，显示文字框控制右侧信息栏是否显示白色背景，气泡透明度为生成路径后每个目的地上的气泡透明度，默认为 0.6，存储路径更改为用户可以自行指定图片路线、信息栏内容存储位置，默认为桌面。

在设置完成之后，点击确定更改，会弹出提示是否更改默认配置，若选择确定，则此次更改的结果会存入文件，当下次打开程序时便会按照更改后的默认设置展示。

图 9 背景图变更效果

1.4. 程序开发运行环境

本程序使用 Python3.6 开发，操作系统为 Windows11 21H2 版本，具体 Python 库版本如下图所示

```
pyqt5~=5.15.6
numpy~=1.19.5
geopy~=2.2.0
opencv-python~=4.5.5.64
requests~=2.27.1
```

2. 数据结构设计

2.1 数据结构

在逻辑结构方面，本文使用图结构进行描述，如图 10 所示。在校园导航情景下，将校园里各个地点抽象为图中的顶点，两个地点之间的道路抽象为图中的边，路径长度抽象为边的权值，构建无向图。

在存储结构方面，本文采用邻接表进行实现，邻接表是一个二维数组，其中 M[i][j]代表从 i 点到 j 点的距离。

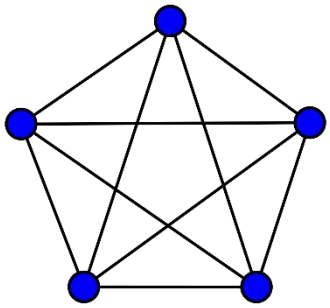

图 10 图结构示例

2.2 程序整体结构

程序整体流程图如图 11 所示，是能够持续运行的闭环结构。

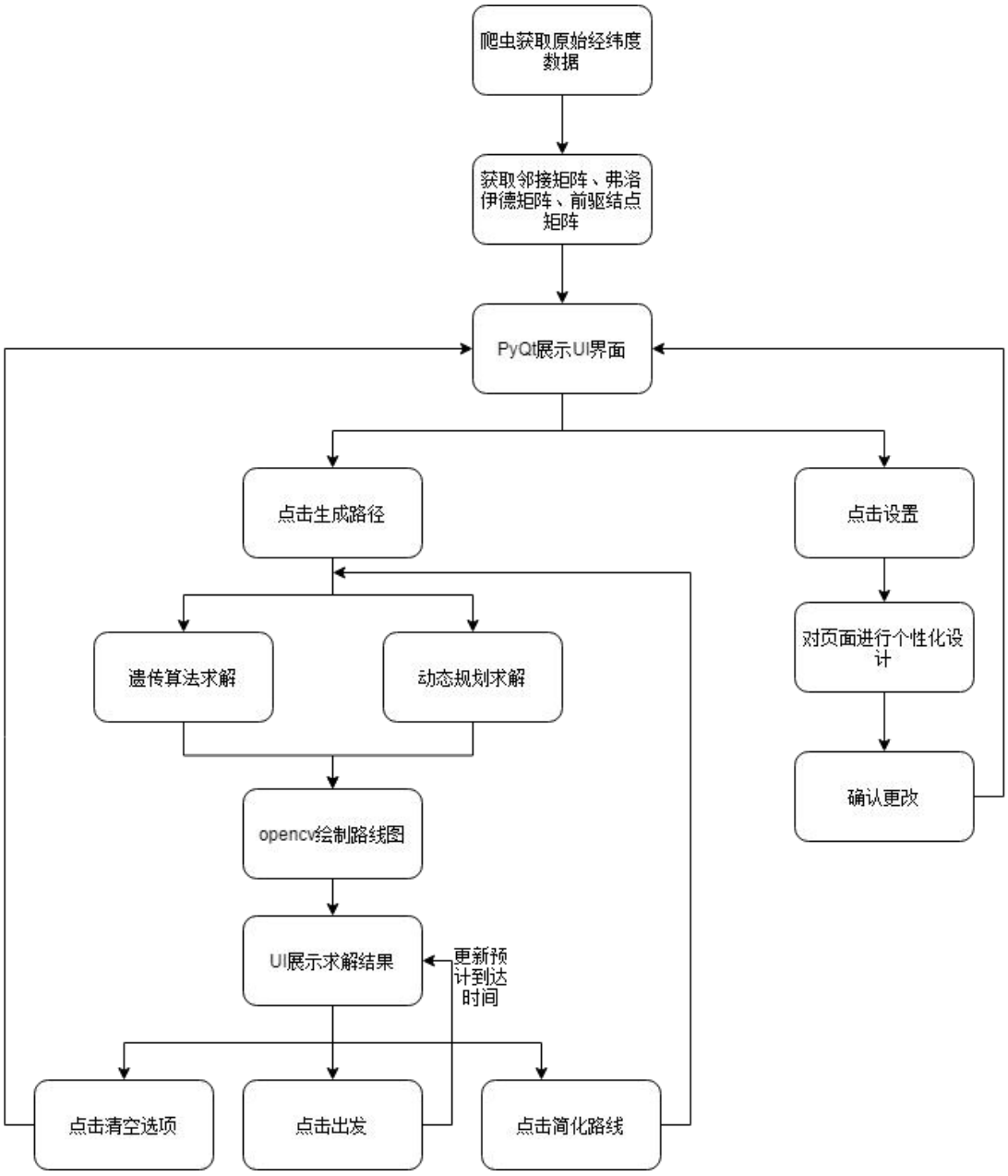

图 11 程序整体流程图

3. 详细设计

3.1 核心类定义

3.1.1 路线类

路线类是所有运算、展示的核心类，是遗传算法中的个体、主界面显示的对象。

路线类定义如下，类成员变量包括只含用户选择的目的地的路线与名称、包含中介地点的路线与名称、每个目的地距离起始点的距离、路线最短路径的长度。类成员函数包括清空路线信息、打印路线所有信息，具体对应关系见注释。

```
class Route:
    citys = []  # 所有地点和编号的对应关系
    simple_road = []  # 只包含指定目的地的编号
    entire_road = []  # 包含中间地点的编号
    min_distance = 1e9  # 最短路线长度
    signle_distance = []  # 每个目的地距离出发的距离
    simple_citys_name = []  # 目的地名称
    entire_citys_name = []  # 包含中间地点的名称

    def Clear(self)          # 清空路线信息
    def PrintInfo(self)      # 打印路线所有信息
```

3.1.2 TSP 类

TSP 类是下述 TSP_DP、TSP_GA、TSP_BackTrack 的父类，包含了解决 TSP 问题不同算法的相同部分。

TSP 类定义如下所示，类成员变量包含 TSP 的 Route 类结果、邻接矩阵、弗洛伊德矩阵、弗洛伊德的前驱结点矩阵；类函数包括读取上述三个矩阵、清除路径信息、通过 road.simple_road 计算完整路径、最短距离、目的地距离出发点距离、城市名称。

```
class TSP:
    road = Route()      # 存储结果的路线类
    mat = np.array([]) # 邻接矩阵
    mat_floyd = np.array([])  # 弗洛伊德矩阵
    mat_pre = np.array([], dtype=int)         # 弗洛伊德前驱结点矩阵

    def Readtxt(self)       # 读取上述三个矩阵
    def ClearAll(self)      # 删除所有路线信息
    def GetMinDistance(self)        # 通过 simple_road 获取最短总距离
    def GetEntireRoad(self)         # 通过 simple_road 获取完整路线
    def GetTwoPointRoad(self, i, j)# 获取每两个点之间的中介地点
    def GetSignalDistance(self)     # 获取每个目的地距离出发点的距离
    def GetCitysName(self)          # 获取每个地点的名称
```

3.1.3 TSP_DP 类

TSP_DP 类继承于 TSP 类，是使用动态规划求解旅行商问题的具体实现。

TSP 类定义如下所示，类成员变量包含弗洛伊德矩阵、最终路线结果、起始节点、记忆

化搜索存储字典、目的地集合，类成员函数包括进行求解并返回 Route 类结果、具体递归求解过程。

```
class TSP_DP(TSP):
    X = self.mat_floyd  # 距离矩阵
    result = [] # 最终路线结果
    start_node = -1 # 起始结点
    memory = {} # 记忆化搜索存储
    nums = []   # 目的地集合

    def run(self)    # 进行求解并返回 Route 类结果
    def solve(self)  # 具体递归求解过程
```

3.1.4 TSP_BackTrack 类

TSP_BackTrack 类继承于 TSP 类，是使用回溯算法求解旅行商问题的具体实现。

TSP 类定义如下,类成员函数包含进行求解并返回 Route 类结果、具体递归求解过程。

```
class TSP_BackTrack(TSP):
    def run(self, nums)
    def backtrack(self, path, come, sum, used)
```

3.1.5 GA 类

GA 类是遗传算法求解 TSP 问题的原理实现，是 TSP_GA 的内置类

GA 类定义如下，具体含义将在后文介绍。

```
class GA(object):
    crossRate = aCrossRate  # 交叉概率
    mutationRate = aMutationRate  # 突变概率
    lifeCount = aLifeCount  # 种群数量
    geneLength = aGeneLength  # 地点数量
    matchFun = aMatchFun  # 适配函数
    lives = []  # 种群
    best = None  # 保存这一代中最好的个体
    generation = 1  # 一开始的是第一代
    crossCount = 0  # 一开始还没交叉过，所以交叉次数是 0
    mutationCount = 0  # 一开始还没变异过，所以变异次数是 0
    bounds = 0.0  # 适配值之和，用于选择时计算概率
    selected_pos = aSelected_pos      # 目的地集合

    def initPopulation(self)    # 初始化种群
    def judge(self)  # 评估，计算每一个个体的适配值
    def cross(self, parent1, parent2)    # 交叉
    def GetOne(self)     # 在种群中获取一个个体
    def mutation(self)   # 变异
    def newChild(self)   # 产生新的后代
    def next(self): # 产生下一代
```

3.1.6 TSP_GA 类

TSP_GA 类继承 TSP 类，是遗传算法求解 TSP 问题的具体实现。

TSP_GA 类定义如下，类成员变量包含目的地集合、城市的经纬度信息、最短距离、Ga 类，类成员函数包含初始化目的地、经纬度转距离计算、适应度函数计算、求解并返回 Routel 类结果。

```
class TSP_GA(TSP):
    citys = []
    selected_pos = selected_pos
    best_distance = -1 # 最短距离
    ga = GA()  # GA 类

    def initCitys(self) # 初始化目的地
    def distance(self, road)     # 距离计算
    def matchFun(self)  # 适应度函数定义
    def run(self, n=100)     # 训练，n 为训练轮次
```

3.2 具体实现

3.2.1 数据处理

数据处理的函数包含于 Crawler.py 和 get_distance.py 两个文件中，函数调用流程图如图 12 所示。数据处理作为预处理，其运算结果均存储于文件中，本身并不参与程序的运行且并不复杂，因此不具体介绍。

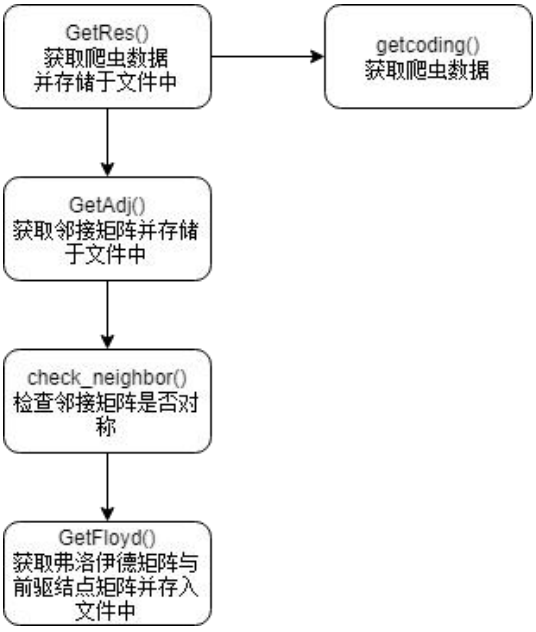

图 12 数据处理流程图

3.2.2 回溯算法

最“计算机”的实现方式，通过穷举所有可能情况获得全局最优解。对于有 N 个目的地的旅行商问题，共有 N!种路线，由于只是简单的遍历，程序编写并不复杂，函数流程图如图 13 所示。

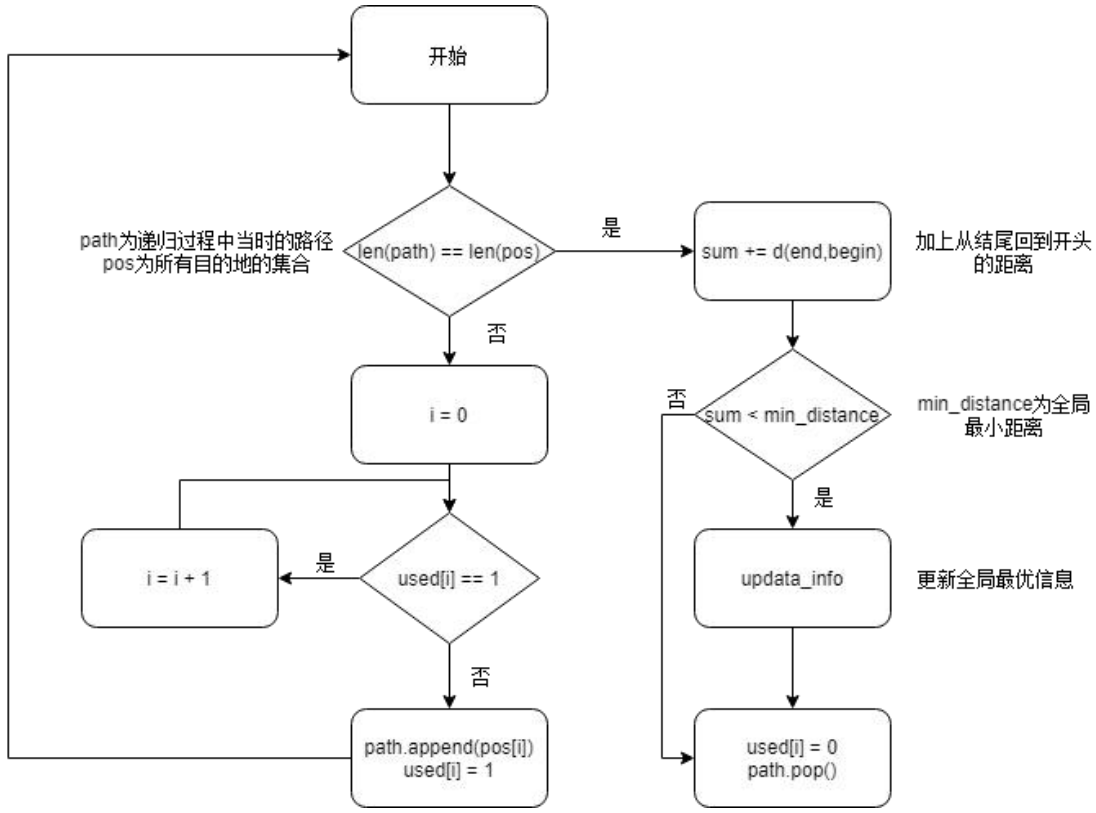

图 13 回溯算法流程图

简单的代价自然是时间复杂度高，O(n!)的时间复杂度使得回溯算法无法在多项式时间内求解，经测试在 13 个点一下的情况下，回溯算法能较快的计算出答案，但当点的个数进一步增加时，回溯便无法在用户能忍受的范围内计算出结果。但考虑到大多数用户不会选择多达 15 个点，因此回溯算法在大多数情况下可以胜任工作。

3.2.3 动态规划算法

动态规划的本质仍为穷举，但在此基础上进行了剪枝操作。

假设从点 p 出发，V 代表仍需要经过的点的集合，d(i,V)代表从点 i 出发经过 V 中所有结点，最后回到点 p 的最短路径长度。则可以写出以下状态转移方程：

$$d(i,V)=\begin{cases} mat[i][s] & i\neq s, V=\emptyset \\ min(mat[i][k]+d(k,V-\{k\})) & k\in V, V\neq\emptyset \end{cases}$$

当 V 为空集，带入到本题情景即为遍历了所有点之后，距离应为从当前点回到起始点的距离；当 V 不为空集，则遍历 V 中所有点，计算下一步经过某一点情况下的距离，取所有结果中最小的作为 i 之后要走的下一个结点。

动态规划给出的状态转移方程连接了两个子问题的状态，而在求解过程中，这些子问题彼此会出现重复求解的情况。假设目的地有{1，2，3，4，5}，当计算 d(2,{3,4,5})时其内部需要计算 d(2,{4,5})，但当计算 d(1，{3，4，5})时，内部仍需要计算 d(2,{4,5}),显然如果能避免这样的计算，特别在点数量较多时，求解效率会有很大的提升。

对于重叠子问题，由于动态规划遍历顺序无明显规律，因此本文使用递归+记忆化搜索的方式进行解决。声明字典 memory，当某次计算出 d(i，V)时，将 str(d(I,V))作为键，其求解结果作为值存储在字典 memory 中，每次需要计算时，先在 memory 中搜索是否已经计算过，如果计算过则无需再次计算，直接使用即可，若未计算则继续计算。

动态规划的流程图如图 14 所示。经测试，动态规划在 20 个地点以下均能够较为快速的求解，已经能够胜任绝大多数情况。

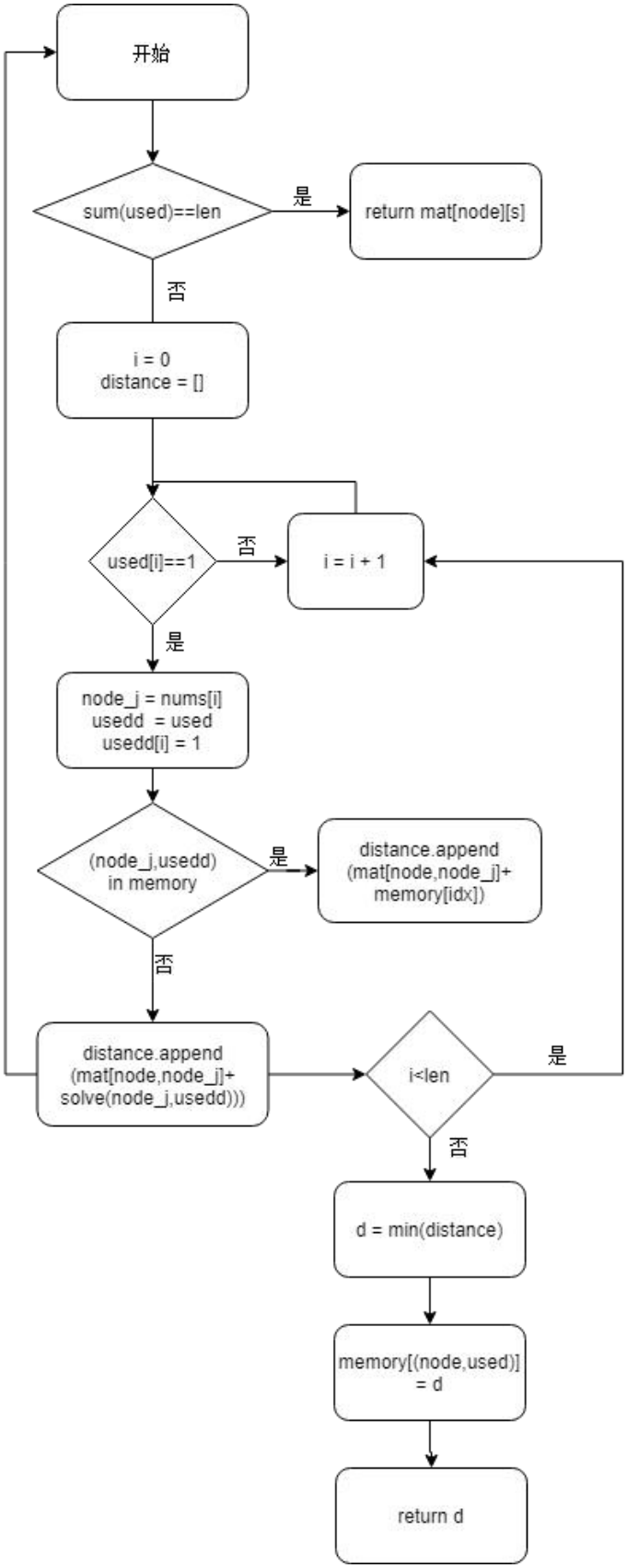
开始
sum(used)==len
是
return mat[node][s]
否
i = 0
distance = []
used[i]==1
否
i = i + 1
是
node_j = nums[i]
usedd = used
usedd[i] = 1
(node_j,usedd)
in memory
是
distance.append
(mat[node,node_j]+
memory[idx])
否
distance.append
(mat[node,node_j]+
solve(node_j,usedd)))
i<len
是
否
d = min(distance)
memory[(node,used)]
= d
return d

图 14 动态规划流程图

3.2.4 遗传算法求解

上述最优化算法已经能够解决本文绝大多数情况，但考虑到本文涉及到了最多 47 个点的规划，同时现在诸多领域在遇到类似 TSP 问题时规模都较大，为了适配大规模 TSP 运算，本文实现了启发式算法中具有代表性的遗传算法。

遗传算法是一种仿生算法，，通过个体的遗传、交叉、变异扩大搜索范围，以“优胜劣汰，适者生存”为迭代依据对种群进行更新。流程图如图 15 所示。

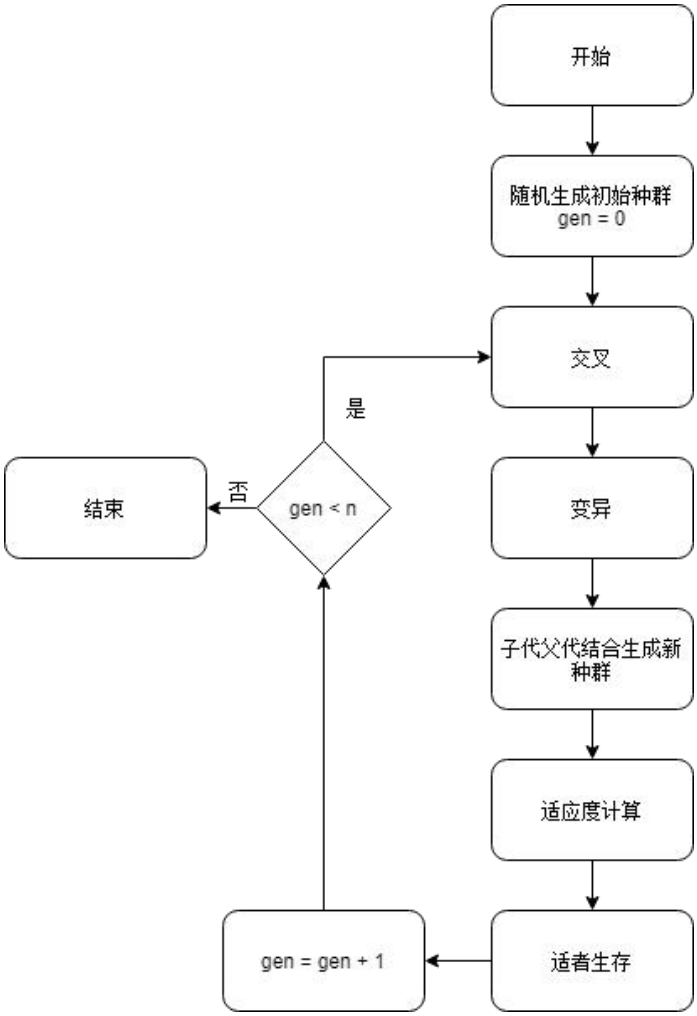

图 15 遗传算法流程图

首先，遗传算法随机初始化 30 个长度为目的地数的序列，每个序列即为一个个体，代表了实际路线的顺序。显然当点个数大于 5 时，30 个序列便无法涵盖所有情况，因此遗传算法通过仿 DNA 交叉变异来扩大搜索范围。

本文采用两点交叉的方式实现交叉操作。将种群两类分组共分为 15 组，对于每组的两个父代个体 f1=c11c12…c1n,f2=c21c22…c2n,随机选取两个交叉点，交换两个个体在所设定的交叉点之间的染色体。考虑如果直接进行交换，交换后的序列可能会出现重复编码，即会经过两次相同的地点，显然不符合 TSP 的要求，为此做如下设计:先取出父代 2 中位于两个交叉点之间的序列，对于父代 1 的所有元素 i，如果 i 不在上述序列中，便入栈新基因序列中，若 i 将要插入的索引为交叉点中较小的那个，则将上述序列全部入栈新基因序列中，随后继续插入父代 1 的元素直到完全生成新的序列。由于交叉点是随机产生，新基因序列便有可能会产生种群中从未有过的序列。

对于变异，考虑到自然界单基因变异概率本身便很小，本文取突变概率为 0.02。在程

序中对于每个序列，取分布于(0,1)之间的随机数 prop，若 prop<0.02,则在该序列中随机选择两个点交换位置，生成新的序列。

在交叉、变异均完成后，对当前所有的个体进行适应度评估，适应度计算函数如下。

$$L = \frac{1}{\sum_{i=0}^{n} d_{c_i c_{i+1}} + d_{c_n c_0}}$$

将所有序列按适应度大小排序，取前 30 个个体进入下一次循环，经过 20 次循环后，取历史适应度最高的序列作为可行解结果进行输出。

可以看到遗传算法每次只会处理固定 M 个可能序列（本文设定为 30），因此其时间复杂度为 O(EMN),其中 E 为训练轮次、N 为选择序列长度。可以看到时间复杂度和选择点的个数仅为一次关系,因此对于大规模的旅行商问题运算,遗传算法仍然能够较快地给出可行解。但随着问题规模变大，所有情况仍然有 N！种，遗传算法很难保证找到全局最优解。即使在点数量较少，最优化算法能够较快求得最优解的情况下，遗传算法依旧有一定概率无法求得最优解。因此本文设定当点个数大于 15 时选择遗传算法，点个数小于等于 15 时选择动态规划算法。

3.2.5 多线程优化路线

当总距离较长时,用户可能能够接受略去某些点来换取更短的路程,本文对此设计如下：遍历当前选择的所有点 p,对于 V-{p}的点集，重新进行 TSP 求解，选择所有解中能够减少距离最多的结果作为优化后的结果，能够看出这个过程多个 TSP 的运行完全相同且彼此独立，因此本文采用多线程进行优化，所有求解同时进行。

Python 的 Tread 库提供了多线程的实现方法，具体实现如图 16 所示。

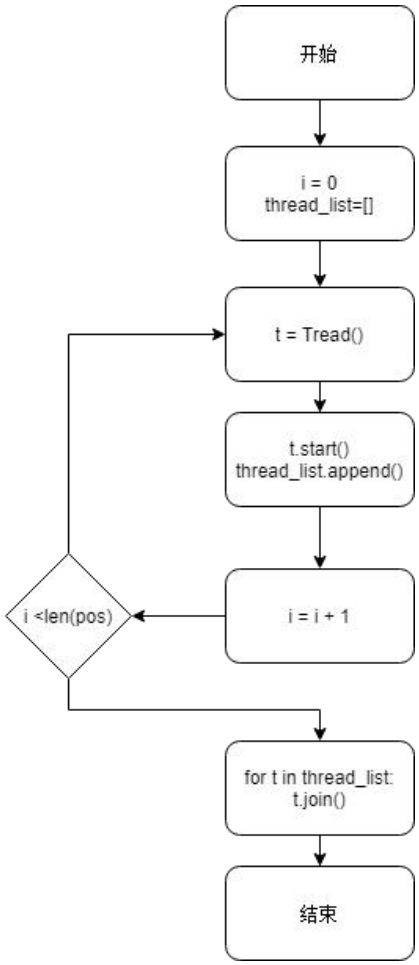

图 16 多线程实现路线优化

得益于 Route 类的封装，本过程不会产生冲突，值得注意的是，在创建完所有线程后，每个线程都应该调用 join 函数，以保证在所有线程都完成工作后再进行择优。

3.2.6 结果展示设计

本程序的结果展示主要体现在地图上的路线绘制、目的地预计到达信息气泡展示、右侧信息栏选中、撤销、求解结果展示。

对于地图上的路线绘制，本文使用 python 的 opencv 库实现。本文的路径分为两种：一种是只包含用户选择目的地的求解结果，另一种是基于前者，在弗洛伊德前驱结点矩阵中获取每两个目的地之间的最短路径，补全成为完整的路径。对于这两种路径，本文使用不同颜色、大小的点进行标志，依次两两连接完整路径中的点，处理结束后将路径存储于本地，同时替换主界面展示的图片。

对于气泡信息展示，其本质为多个独立窗口，定义于 Bubble.py 中的 Ui_Bubble 类，气泡默认的透明度为 0.6，当鼠标放置于某气泡上，通过重写进入窗口的事件函数，将气泡的透明度改为 1，既可实现既不遮挡路线、又不影响阅读。气泡内的预计到达时间为当前的系统时间加上到达每个目的地的时间，点击界面中“出发”按钮，该预计到达时间会重新计算。

3.3 算法复杂度分析

本文所使用的三种算法回溯、动态规划、遗传算法具体实现过程以及原理已经在上节展示，本节着重分析三种算法的时空复杂度

对于回溯算法，在时间上穷举所有可能，因此时间复杂度为 O(n!),空间上在求解过程中引入了标志已访问的 uesd 数组和存储当前路线的 path 数组，因此空间复杂度为 O(n)。

对于动态规划算法，其剪枝后的效果为定义的 d(i,V)状态仅计算一遍，当有 n 个目的地且确定出发点时，i 的取值有 n-1 个，V 可能为除去初始点外所有选择点的子集，共有 $\sum_{k=0}^{n-1}\binom{n-1}{k}$种，因此总的状态数为$(n-1)\sum_{k=0}^{n-1}\binom{n-1}{k}$，在获取每个状态时均只涉及从邻接矩阵中以索引方式取出以及简单的加和，因此时间复杂度为$O(n2^n)$。在空间上引入字典存储状态，因此时间复杂度即为状态数，为$O(n2^n)$

对于遗传算法，记种群中个体数量为 m，迭代次数为 e，选择的地区数为 n，交叉操作时间复杂度为 o(n),变异操作时间复杂度为 o(1),优胜劣汰排序时间复杂度为$O(n\log n)$,因此总的时间复杂度为$O(emn\log n)$，空间复杂度为存储每个个体的 O(n)

对于三种算法的总结如表 1 所示。

	时间复杂度	空间复杂度	是否精确解	求解速度
回溯算法	$O(n)$	$O(n)$	是	慢
动态规划	$O(n2^n)$	$O(n2^n)$	是	较快
遗传算法	$O(emn\log n)$	$O(n)$	否	对于少量点慢，对于大量点快

表 1 三种算法比较

4. 程序测试

4.1 正确运行程序用例

4.1.1 TSP 求解示例

在选点过程中随机选取 11 个点，结果如图 17 所示，可以看到路线十分舒展，符合设计

预期，同时总距离 2.258km、预计用时 34 分也符合认知。

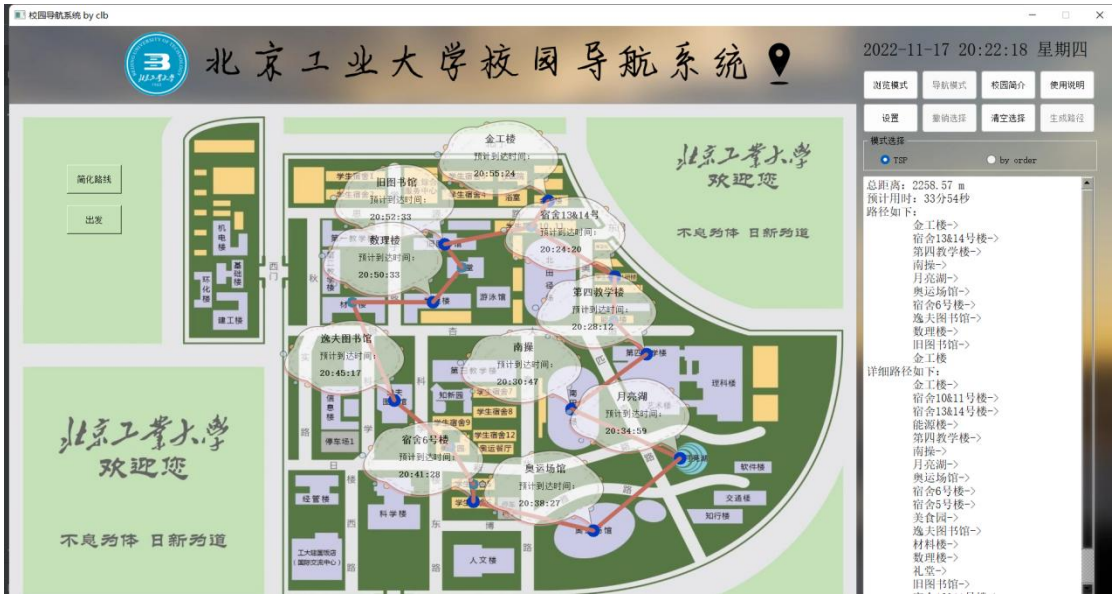

图 17 TSP 正常求解示例

4.1.2 多线程简化路线示例

为展示简化路线的效果，初始选点如图 18 所示，可以看到北操距离其余几个目的地的距离均很远，若按照上文所叙述的简化逻辑，其应当在本轮简化中被舍弃，简化后的路线图如如 19 所示，可以看到简化结果符合预期。

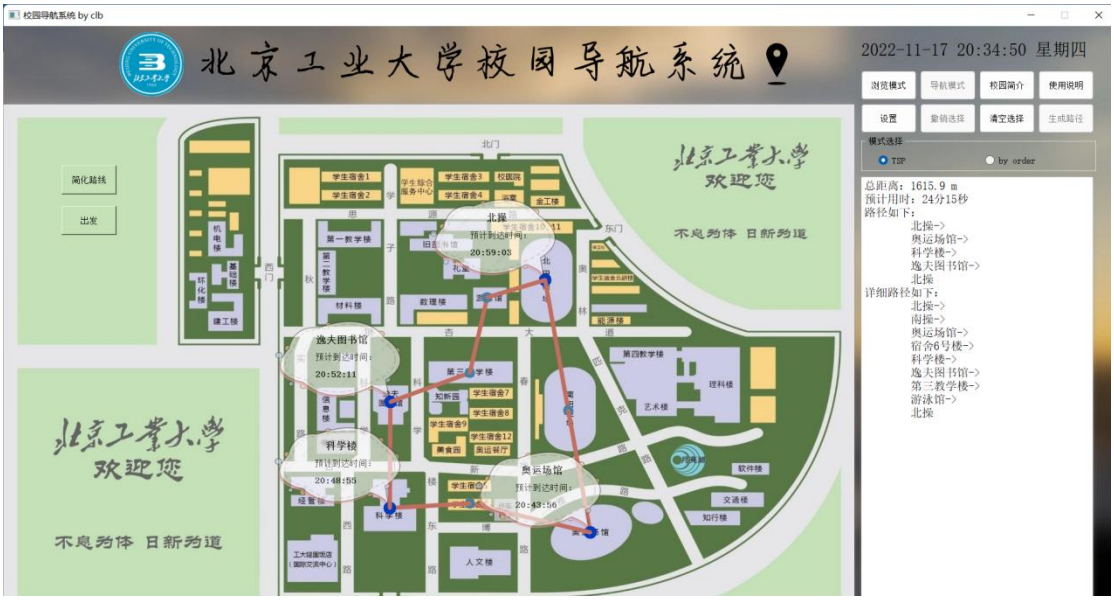

图 18 多线程简化路线示例

图 19 多线程简化路线结果展示

4.1.3 按顺序寻路示例

选点顺序为北操->奥运场馆->逸夫图书馆->理科楼，展示结果如图 20 所示，可以看到路线交叉、右侧信息框输出了正确的结果,符合设计预期。

图 20 按顺序寻路展示

4.2 导致程序错误的用例

暂未发现

4.3 临界情况示例

4.3.1 选择目的地过少

当用户只选择了一个点便开始导航时，程序会弹窗警告请至少选择两个目的地，如图 21 所示

图 21 选择目的地过少提示

4.3.2 简化路线过多

当用户多次点击简化路线时，程序可以做出反应，但当地点减少至两个时在点击简化路线，程序会提示“路径已经很短啦”，如图 22 所示。

图 22 简化路线过多提示

5. 用户使用说明

本程序为北京工业大学校园导航系统，内置 47 个学校地点，可以在选择多个地点后实现经过每个地点一次且回到原地点的路线规划。

首先，您可以点击程序地图上的任意地点(除建国饭店与西区)来选中您想要到达目的地，再次点击可以取消指定地点选中，点击右上角“撤销选择”可以撤销上一次选中的地点，点击“清空选择”可以清空之前选择的所有地点。所有选点信息会同步在右侧信息栏展示，您可以随时确定自己选点是否正确。

当选点完成后，右侧有两个模式供您选择，一个是 TSP,即上述功能的实现，一个是 ByOrder，即按照您选点的顺序为您规划最短到达路线。在选择完毕模式后，您可以点击“生成路线”来获取规划结果。右侧信息栏会详细展示您此次行程的总距离、预计时间以及路线规划;图片上会绘制出具体路线,同时您的目的地上会有气泡展示地点名称和预计到达时间，为了不遮挡您的视线，气泡默认为半透明，鼠标放置于气泡上方即可取消透明，您也可以在设置界面更改气泡透明度。若您当前不想出发，可以在出发时点击左侧“出发”按钮，气泡内的预计到达时间便会以您当前的系统时间进行更新。与此同时，您的路线信息也会存储在默认桌面的位置，包括路线地图以及右侧信息栏的显示信息，您可以在设置界面更改默认存储位置。

若您认为当前规划路径过长，想要简化路线，可以点击左侧“简化路线”按钮，系统会智能为您略去一个花费时间最长的目的地，具体省略信息也会在右侧信息栏中展示。

点击右上角“设置”按钮，可以进入设置界面，您可以选择程序背景图、规划模式、规划算法、气泡透明度、是否展示右侧文本框背景、结果存储路径等，当您选择完毕后点击“确认更改”，在弹出的提示框中选择确认更改默认配置即可将您此次的更改永久保留，当下次进入程序时仍然可以保留上次的设置。

右上角的校园简介可以在子窗口中自动打开学校官网,您可以在该地方获取学校最新消息。

6. 总结与提高

本次课设是本人首次使用 Python 开发完整项目、首次开发电脑桌面应用，对于本人知识面的扩展起到了非常大的作用。

在代码结构的设计过程中，本人尝试细分多个模块、多个类进行组织，尽量减少模块之间的耦合，同时在使用时只使用每个模块对外提供的函数接口，这样组织的好处在于在扩增功能或修复 BUG 等情况时，只需要对指定模块进行更新，无需更改其他模块，同时程序的可读性也大幅提高。

在界面设计过程中，本人愈发发现 UI 设计的重要性。受限于 Qt 的样式接口功能较少、社区内容不够丰富，难以使用 Qt 开发出 Web 端清新流畅的样式，但本人仍花费一两周的时间学习界面设计，参考商业网站设计思路[1][4][5]，专注于界面设计、用户体验，包括在模块周围设置阴影[3]、在背景设置模糊处理的校园景色、设计设置界面允许用户个性化自己的使用体验、设置气泡[2]并跟随主窗口移动等。随着 js 语言的出现，现代 web 的发展，web 开发已经有超过桌面端的趋势，作为计算机专业的学生，也应该紧跟趋势，不断学习前沿语言、框架。

开发的过程注定是苦乐交织的，在过程中不可避免的会被很多问题所困扰而停滞不前，比如不熟悉 Python 某些语法、为了解决多线程之间的冲突、为了让气泡跟随主窗口、为了解决程序体量太大难以定位问题所在等。作为程序员，我认为在日常学习中遇到问题是极其常见的，应当放平心态，通过靠谱的搜索引擎如 Google、Bing 等、靠谱的交流社区如 CSDN、Stackoverflow、github 等主动独立寻找问题答案,对于繁多的技术并不一定需要深刻掌握，现用现查是更为高效的方式。

我也曾遇到过在做别的项目时因为没有及时存档导致项目的丢失，在了解并使用过 git 技术后，我认为这是一个非常有用且必要的工具，特别是其背后的 Github 开源社区，能够了解到 Linux 当时开源的里程碑意义、体会到现代程序世界的开源精神。我的程序也已经上传到 Github 上[6],待该课程结束会设置为公开状态。

我十分庆幸自己在开发过程中能够保持热情、保持对于新事物的探索，面对繁多的开发技术，我也总是想都了解一二，掌握基础的用法，我相信王老师鼓励我们爬虫、模仿商业网

站的出发点也在于此，我们不应当像高中一样仅局限于课内知识的获取，而更应该拥抱广大的互联网世界，探索前沿技术的魅力。

最后，感谢所有在开发过程中帮助过我的朋友，感谢王老师每次不辞辛苦的答疑、检查！

7. 参考资料

[1]. https://api.map.baidu.com/lbsapi/getpoint/index.html

[2]. https://blog.csdn.net/qq_41484105/article/details/103146355

[3]. https://blog.csdn.net/marwenx/article/details/108036435

[4]. https://www.bjut.edu.cn/xxgk/xxjj1.htm

[5]. https://edu.hicomputing.huawei.com/zh/learningresources

[6]. https://github.com/GreaterChen/BJUT-DataStructureProject