

“模拟旅行查询系统” 实验报告

班级：2014211314

指导老师：徐塞虹

组长：叶文霆

小组成员：叶文霆 熊柏桥 董升华

摘要：

使用Python爬虫技术从网上获取汽车、飞机与火车三种交通工具的真实票价，基于其数值特征，我们建立了一个总计14个城市、342个交通方式的旅行交通网。

算法方面我们注意到问题与经典旅行商问题有诸多共同之处。我们对图先使用了时间复杂度为 $O(v^3)$ 的 Floyd 算法与 $O(e)$ 的SPFA算法做图的预处理，再使用业界流行的“模拟退火算法”解决时间最少与金钱最少策略的问题。对于带有时间限制的费用最少计划，我们使用我们基于前面的算法设计了启发式剪枝，其剪枝效果能达到朴素的最优化剪枝的二至三倍。

从Github桌面客户端界面获得设计灵感，我们基于Qt5.6.0实现了一个精美而简洁的图形化界面，可以动态显示每一个用户的状态与路线状态。

最后对算法的正确性与效率进行了严格的分析与验证。

关键词： Python爬虫 Floyd算法 SPFA算法 模拟退火算法 启发式剪枝
Qt图形界面 正确性分析 效率分析

I. 问题要求	2
II. 建立模型	3
III. 数据结构	5
IV. 算法设计	6
V. 算法测试	
1. 正确性分析	9
2. 健壮性测试	11
3. 时间性能测试	14
VI. Qt图形界面	
1. 开发环境	15
2. 使用说明	15
VII. 总结	18
VIII. 附录	19

一、问题要求

1. 问题简述

要求建立一张城市图表，城市之间有三种交通工具（汽车、火车和飞机）相连，某旅客于某一时刻向系统提出旅行要求，系统根据该旅客的要求为其设计一条旅行线路并输出；系统能查询当前时刻旅客所处的地点和状态，并要求做日志记录。

旅行计划包括起点、终点、必经城市与计旅行策略，策略共有以下三种：

- a) 花费最少策略
- b) 时间最少策略
- c) 带时间限制的花费最少策略

2. 任务分工

叶文霆：设计模型、设计与实现算法

熊柏桥：图形界面的UI设计与框架搭建

董升华：采集模型数据，协助实现图形界面的各个部件

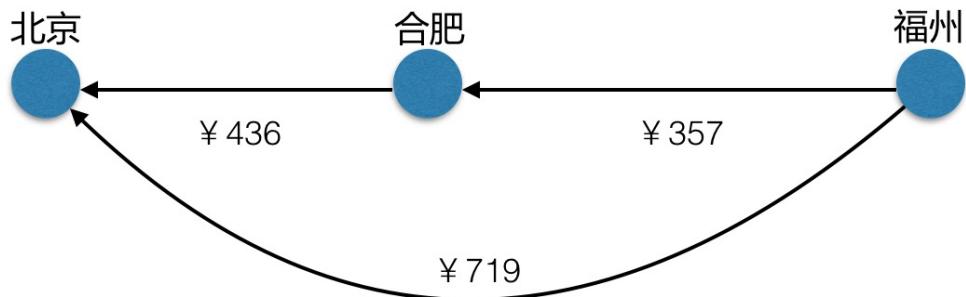
二、建立模型

1. 模型假设

有一个最朴素的方法来获得城市之间交通工具的费用与时长，即：从网络上搜索两个城市之间交通工具的价格，直接作为模型的边权即可。例如考虑福州到北京的火车费用，我们先从网络上查到G28是符合条件的高铁，并得知其始发时间9: 52、到达时间17: 35，费用是748元。这个模型看似很美好，但有个问题：

它忽略了火车始发站与到达站之间的途径城市。即考虑我们的模型中除北京福州之外还包含了G28的途径城市之——合肥。如果是之前的模型，我们设计出来的路线可能就是直接从福州到北京到直达线路，而不会经过南昌。其根本原因是忽略了陆地交通受限于地缘因素。为了解决这个问题，在不丧失普适性的前提下我们做出如下模型简化假设：

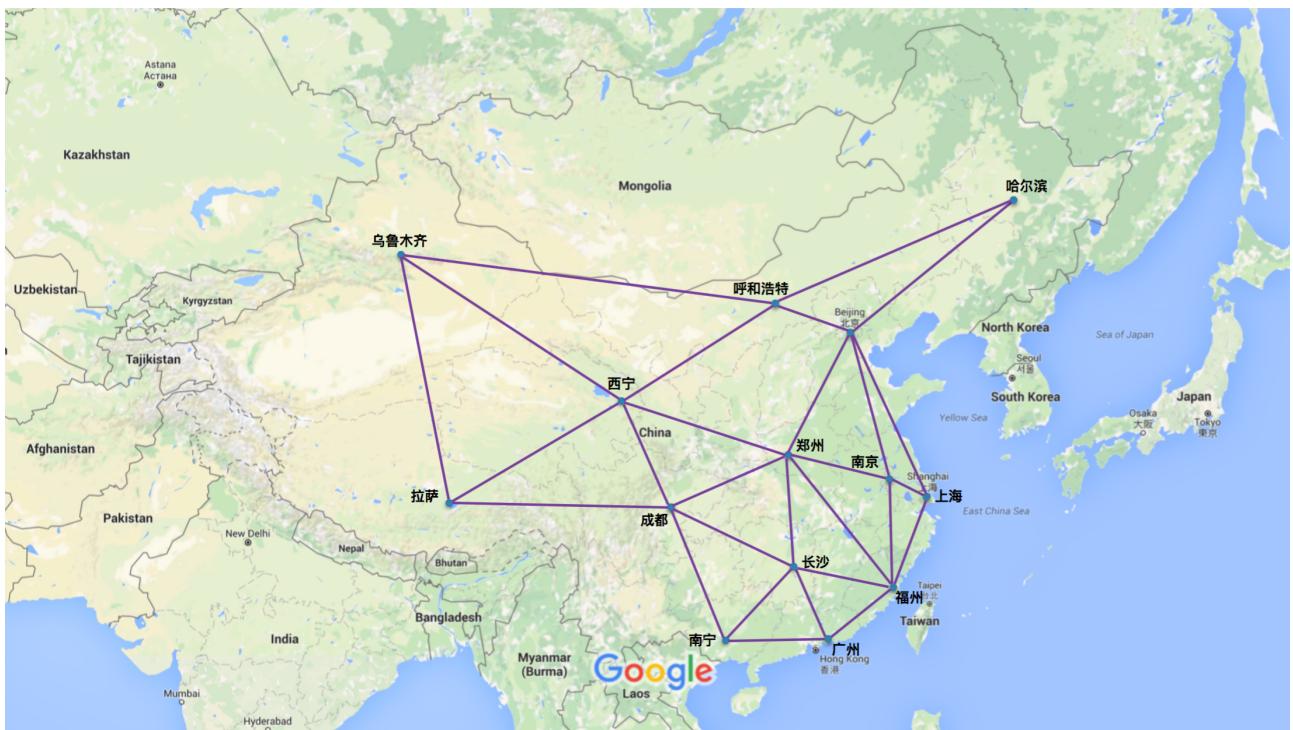
1. 两地之间有直达（不经过其他城市）的火车，当且仅当两地地缘上相邻。
2. 对于每一种交通工具，其每天的时刻表是固定的。



图一 G28价格示意图

基于以上两个假设，我们考虑两非相邻城市某车次的票价时，我们分别计算其每一条边上的费用，累加起来就是总费用。如图一所示，假设我们要做 G28从福州到北京，我们用福州→合肥加上合肥→北京的票价之和793元来近似真实票价748元，误差为6.01%，说明在允许一定误差的情况下，假设合理。

2. 数据采集



建立模型之后，我们选取了中国14个一线城市，分别是北京、上海、哈尔滨、南京、郑州、长沙、福州、广州、南宁、西宁、成都、拉萨、乌鲁木齐和呼和浩特，在任意两个相邻城市之间有6条边，分别是火车、高铁、汽车来回两个方向各有一车次。除此之外任意两个城市之间还有来回飞机航班各一次，总计342条边。

具体做法是：

1. 对于陆地交通网的时刻表与价格信息，我们通过使用 Python 在[铁友网](#)上面通过重复提交表单，利用正则表达式匹配出我们需要的时刻信息与价格（取二等座的价格）。
2. 对于航班时刻表，我们不考虑中转航班（在实际中，国内航班里中转航班所占比例也不大），我们对于图中任意城市对都建立一条边，同样使用 Python，在[去哪儿网](#)上面查找对应的时刻表和价格信息。·

三、数据结构

```
// Travel-Simulation-System I
// Created by Yewenting. on 16/4/10.
// Copyright © 2016年 Apart I, 404. All rights reserved.

// global.h

#define MAXV          100           // 模型能存储的最大城市数
#define TIME_PER_SEC  10            // 多长时间更新一次状态

extern int current_time;           // 程序现在时间
extern int current_day;            // 程序现在天数

// 分别存储路径日志文件、事件日志文件
extern std::ofstream routeFile, eventFile;

// people.h

class TravelPlan
{
public:
    int source, destination, type;
    std::vector<int> station;        // 必经城市
    int timeLimit;                  // 策略三的时限
};

class People
{
private:
    std::string name;               // 客户现在所处的位置
    int location;                  // 旅行计划
    TravelPlan plan;               // 存储计划好的交通路线
    std::vector<Line> route;        // 用户到现在所花费的钱、时间
    int usedMoney, usedTime;
};

// trafficNet.h

class Line
{
public:
    std::string name;               // 线路名
    Line *NextLine;                // 下一条边
    int head, tail;                // 该边的起点与终点
    int leaveTime, duration;       // 出发时间与用时
    int cost;                      // 费用
};
```

```

class City
{
public:
    Line *FirstLine;           //第一条边
    std::string name;          //名字
};

class TrafficNet
{
private:
    City citys[MAXV];          // 存储城市信息
    std::vector <People> people;   // 存储用户的信息
    int num_of_city, num_of_people;
    std::map<std::string, int> cityNum; // 城市名与indice对应关系

    // 不考虑必经城市的价格最优路线与其费用
    std::vector <Line> cheapRoute[MAXV][MAXV];
    int cheapWay[MAXV][MAXV];

    // 不考虑必经城市的时间最优路线与其用时 第三维为出发时间
    std::vector <Line> quickRoute[MAXV][MAXV][MAXTIME];
    int quickWay[MAXV][MAXV][MAXTIME];

    // 用于策略三的深度优先搜索 分别存储当前最优解 最优路线与当前解
    int minMixCost;
    std::vector <Line> bestRoute, currentRoute;
};

```

四、算法设计

1. 最少费用策略

在最少费用策略中，我们不关心时间这一维对于路线选择的影响。为了省钱，我们可以花任意时间去等待下一辆最便宜的车。所以我们可以先用 Floyd 算法求出任意点对的最少的到达费用，剩下的就是考虑我们要怎样经过必经城市，能够使得我们费用最少。这个时候问题就转换为了标准的 TSP，即旅行商问题。对于旅行商问题有以下两种做法：

- 朴素的状态压缩 + 记忆化搜索

第一种算法的思路是考虑这样一个状态表达： $f[i,s]$,表示的是现在在城市 i ，已经经过了 s 集合里的城市所花费的最少时间 (s 是一个station.size()位的二进制数，第 i 位为0表示未到达，1表示已到达），状态转移方程如下：

$$f(s, t, O) = \min \{ f(s, x, O - \{x\}) + dis_{x,t}, \forall x \in O \}$$

最后的答案显然是在 $f[destination, \{0, 1, 2, \dots, N-1\}]$ 中。这样状态空间的大小是 $n * 2^n$, 当必经城市的数目足够多 ($n \geq 30$) , 即用户要求设计大型旅游线路的时候, 对于一个要求及时响应的模拟系统, 这样的效率是不可接受的。

所以我们使用考虑使用效率更高的模拟退火算法

• 模拟退火算法

模拟退火的基本思想:

- a) 初始化: 初始温度 T (充分大), 初始解状态 S (是算法迭代的起点), 每个 T 值的迭代次数 L
- b) 对 $k=1, 2, \dots, L$ 做第3至第6步
- c) 产生新解 S'
- d) 计算增量 $\Delta t' = C(S') - C(S)$, 其中 $C(S)$ 为评价函数
- e) 若 $\Delta t' < 0$ 则接受 S' 作为新的当前解, 否则以概率 $\exp(-\Delta t'/T)$ 接受 S' 作为新的当前解.
- f) 如果满足终止条件则输出当前解作为最优解, 结束程序。终止条件通常取为连续若干个新解都没有被接受时终止算法。
- g) T 逐渐减少, 且 $T \rightarrow 0$, 然后转第2步。

在模拟旅行系统中, 我们作如下定义:

1. 评价函数: 即为当前解的总路径长度
2. 产生新解: 随机交换当前解中两个点的顺序
3. 接收函数: 接受函数。接受函数决定选择哪一个解决方案, 从而可以避免掉一些局部最优解。

首先我们检查如果相邻的解决方案是比我们目前的解决方案好, 如果是, 我们接受它。否则的话, 我们需要考虑一下影响因素:

1. 相邻的解决方案有多不好;
2. 当前的温度有多高。在高温系统下更有可能接受较糟糕的解决方案。

这里我们使用指类型函数进行拟合,

$$P = e^{\frac{\text{now Energy} - \text{new Energy}}{\text{temperature}}}$$

即上面的 $P(dE) = \exp(-dE/(kT))$ 。

2. 时间最优策略

与最小费用策略不同在于，时间这一维在交通方式选择中起到了影响。我们先考虑一个简化问题：不考虑必经城市，在 $time = k$ 时从城市 i 最早能在什么时间到城市 j （等待时间加转移时间）。我们使用了SPFA（Shortest Path Faster Algorithm）最短路算法。考虑到SPFA的时间复杂度为 $O(e)^1$ ，预处理的总时间为 $O(24 * v * e)$ 。具体求解过程的区别之处仅在于SPFA松弛的条件改为了：

```
int TrafficNet::Move(const int &time, const Line &edge) const
//表示在 time 时间走某边 将在什么时候到
{
    if (time % MAXTIME <= edge.leaveTime)
        return time + (edge.leaveTime - time % MAXTIME) + edge.duration;
    else
        return time + (edge.leaveTime + MAXTIME - time % MAXTIME)
+ edge.duration;
}

// 以下是SPFA松弛一个顶点的代码片段
while (k != NULL)
{
    if (quickWay[source][k->tail][time] > Move(quickWay[source][node]
[time], *k))
    {
        // 赋值quickWay、quickRoute 此处省略

        // 将松弛后的点添加入队列
        if (used[k->tail] == false)
        {
            used[k->tail] = true;
            q.push_back(k->tail);
        }
    }
    k = k->NextLine;
}
```

其余各处与标准SPFA类似，将预处理后的结果传给模拟退火函数即可。

¹ [O224] 段凡丁 关于最短路径的SPFA快速算法 《西南交通大学学报》1994年第2期 207-212,共6页

3. 带时限的费用最优策略

考虑到此时最优策略选择与题目要求所给的时间限制有很大的联系：当时间限制很宽松的时候，我们倾向于用更多的换乘等待时间与乘坐时间来换取费用的节省，而当时间限制很严苛的时候就恰恰相反。这一点说明了在带时限的费用最优策略中，经过必经城市的路线不一定时间最优或是费用最优，而是一种比较无规律的路线。这里的无规律指的是可能一会儿选择时间较优的，一会儿选择费用较优的。所以如果继续选择模拟退火算法作为核心算法，其答案将不会是最优解。

基于此类考虑，我们选择了能够保证正确性的深度优先搜索。在原本的最优化剪枝——当前费用大于当前答案就退出——与可行性剪枝——所用时间超过限制之外，我们还基于模拟退火算法设计了一个启发式剪枝：

假设在深搜的某一过程中我们还有 city1, city2...cityn 的必经城市还没有经过，我们用模拟退火算法分别计算出遍历这些城市至少需要的金钱 Money 与时间 Time。考虑到模拟退火计算出来的是最少的金钱/时间花费，如果加上这些花费我们仍不能够得到一个比当前解更优的答案，对这个搜索节点继续进行搜索显然是没有意义的，可以对其进行剪枝。在性能测试中，这一剪枝可以把耗时减少为原来的 30%~40% 左右，浮动大小取决于图的拓扑结构。

五、算法测试

I. 正确性测试

1. 费用最优策略

测试数据集：城市（点）数 $N = 20$, 交通工具（边）数 $E = 1140$

首先去掉 Release 版本算法三中的所有剪枝，即最优化剪枝、可行性剪枝与启发式搜索，我们就获得了一个简单朴素的搜索算法，考虑到此算法性能较差，我们选取了较小的数据集进行测试。

测试过程分为以下几步：

- a) 随机的选择起点，终点，途径城市；
- b) 将目标设为金钱最少，得到答案 ans1；
- c) 再将目标设为规定时间内花费最少，将时间限制设为 Maxlongint，得到答案 ans2；
- d) 比较 ans1 与 ans2，若相同则说明算法正确，若不相同则说明算法一或算法三至少其中一个有误；

e) 将上述四步循环10次。

最终结果，金钱最少计划的输出与搜索结果一致，证明了算法一的正确性。

2. 时间最优策略

测试数据集：城市（点）数N = 11，交通工具（边）数E = 280

首先对算法三做同样的处理，获得一个朴素版本的深搜算法。

测试过程分为以下几步：

- a) 随机的选择起点，终点，途径城市（小于等于五座）；
- b) 将目标设为时间最少，得到答案ans1；
- c) 再将目标设为规定时间内花费最少，将时间限制初值设为0；
- d) 判断在当前的时间限制内，算法三是否能得到一个可行解。若存在一个可行解，记当前时间限制为ans2，若不存在时间限制加一，重复该步；
- e) 比较ans1与ans2，若相同则说明算法正确，若不相同则说明算法二或算法三至少其中一个有误；
- f) 将上述步骤进行十次。

最终结果，时间最省计划的输出与爆搜结果一致，证明了算法一的正确性。

3. 带时间限制的金钱最省计划

考虑到深搜本身就搜索的所有可能的解空间，所以不存在真正能够证明算法三正确性的算法。但是对算法一算法二的验证结果，也佐证了我们算法三的正确性。

此外，我们还对算法三的答案进行了如下分析，也可以证明解的合理性：

测试数据集：城市（点）数N = 20，交通工具（边）数E = 1140

时间限制	最少花费	所用时间	转乘次数
30	107	28	7
35	95	33	7
40	30	40	5
45	30	40	4
50	28	50	6
70	14	61	5

CHART 1 时间限制与算法三最优解的关系

输入：从city1到city20，必经城市为city5, city9, city13。观察时间限制对最优解的影响。

从上表格中我们可以看出：

- a) **最优解都会尽量的耗尽时间限制。**这也符合交通工具所用时间越长，其价格越低的性质；
- b) **随着时间限制的越来越松，最少花费也越来越大，并有明显的分段。**这也符合当某非常便宜的路线符合时间限制以后，对于最优解会有很大的影响；
- c) **当时间限制很紧的时候，转乘次数比较大。**这是因为为了达到时间要求，乘客只好做尽量快的交通工具而不是等待一个直达的车次，从而增多了换乘次数；

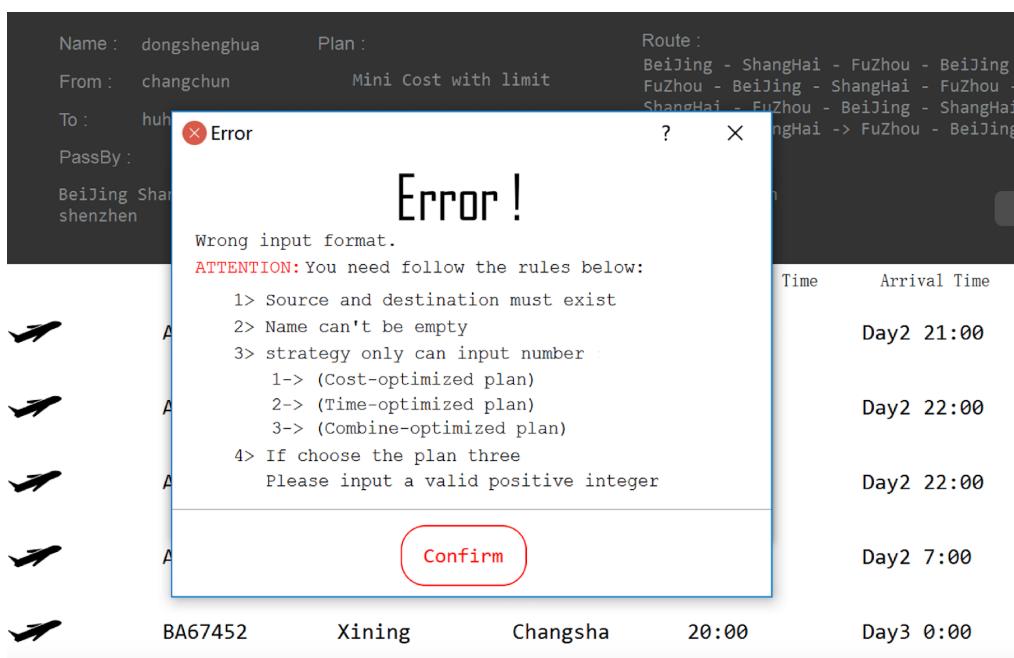
经过以上四点分析，在综合之前算法一、算法二的验证，可以说明算法三的正确性。

II. 健壮性测试

测试数据集：城市（点）数N = 20，交通工具（边）数E = 1140

1. 输入数据无效

会给出对应的错误信息框，详见第六节图形界面。



2. 必经城市为空

```
1 20 1
How many cities you wanna pass by, and what are they?
0
For client, Your route is:
At 0:00, take on the BUS17861 from 1 to 20, arrive at 21:00.
Totally, it takes you $9 and 45h.
1 20 2
How many cities you wanna pass by, and what are they?
0
For client, Your route is:
At 6:00, take on the DU5447 from 1 to 4, arrive at 9:00.
At 9:00, take on the YY24307 from 4 to 20, arrive at 15:00.
Totally, it takes you $54 and 15h.
1 20 3
How many cities you wanna pass by, and what are they?
0
And what's your time limit?
30
For client, Your route is:
At 8:00, take on the D10712 from 1 to 20, arrive at 2:00.
Totally, it takes you $23 and 26h.
```

图3 必经城市为空时的解

从图3中我们可以看出来，当必经城市为空时算法能够给出合理解。

3. 起点终点为必经城市

```
1 20 1
How many cities you wanna pass by, and what are they?
3
1 10 20
For client, Your route is:
At 22:00, take on the BUS27350 from 1 to 10, arrive at 19:00.
At 23:00, take on the BUS9458 from 10 to 20, arrive at 4:00.
Totally, it takes you $10 and 76h.
1 20 2
How many cities you wanna pass by, and what are they?
3
1 10 20
For client, Your route is:
At 3:00, take on the FP3430 from 1 to 11, arrive at 4:00.
At 6:00, take on the MT9850 from 11 to 19, arrive at 14:00.
At 14:00, take on the YG20723 from 19 to 10, arrive at 15:00.
At 0:00, take on the D31203 from 10 to 20, arrive at 2:00.
Totally, it takes you $65 and 26h.
1 20 3
How many cities you wanna pass by, and what are they?
3
1 10 20
And what's your time limit?
50
For client, Your route is:
At 6:00, take on the D15724 from 1 to 2, arrive at 8:00.
At 8:00, take on the D15725 from 2 to 1, arrive at 10:00.
At 12:00, take on the D27938 from 1 to 14, arrive at 18:00.
At 18:00, take on the BUS20477 from 14 to 10, arrive at 0:00.
```

图4 起点终点为必经城市

从图4可以看出，当起点终点为必经城市的时候，不会出现多次经过起点终点的不合理情况。

4. 必经城市存在重复

```
1 20 1
How many cities you wanna pass by, and what are they?
3
10 11 10
For client, Your route is:
At 1:00, take on the BUS8909 from 1 to 11, arrive at 6:00.
At 12:00, take on the BUS1909 from 11 to 10, arrive at 9:00.
At 23:00, take on the BUS9458 from 10 to 20, arrive at 4:00.
Totally, it takes you $11 and 76h.

1 20 2
How many cities you wanna pass by, and what are they?
3
10 11 10
For client, Your route is:
At 3:00, take on the FP3430 from 1 to 11, arrive at 4:00.
At 6:00, take on the MT9850 from 11 to 19, arrive at 14:00.
At 14:00, take on the YG20723 from 19 to 10, arrive at 15:00.
At 0:00, take on the D31203 from 10 to 20, arrive at 2:00.
Totally, it takes you $65 and 26h.

1 20 3
How many cities you wanna pass by, and what are they?
3
10 11 10
And what's your time limit?
40
For client, Your route is:
At 3:00, take on the FP3430 from 1 to 11, arrive at 4:00.
At 4:00, take on the D31495 from 11 to 12, arrive at 6:00.
At 7:00, take on the D27607 from 12 to 10, arrive at 23:00.
At 0:00, take on the D31203 from 10 to 20, arrive at 2:00.
At 2:00, take on the D31982 from 20 to 9, arrive at 5:00.
At 8:00, take on the AF23335 from 9 to 10, arrive at 9:00.
At 11:00, take on the AF23336 from 10 to 9, arrive at 12:00.
At 15:00, take on the VG24604 from 9 to 20, arrive at 16:00.
Totally, it takes you $56 and 40h.
```

图5 必经城市存在重复

从上图中可以看出，必经城市存在重复并不会影响算法的正确性。

实际上算法对于以上三种情况并没有做特殊的判断，而是符合条件的最优解所自然具有的性质。

在这里我们并没有对模拟退火算法的正确性做理论上的分析，实际上这一块的学术领域仍十分活跃，与此类似的算法还有遗传算法等。一般认为模拟退火算法能够以极高的概率获得全局最优解，但是也有一定概率获得一个局部最优解。但是只有在图的规模很大的时候（百万级别），才会去考虑模拟退火算法的正确性，而我们的模型规模远不可能达到这种规模，所以模拟退火的正确性能得以保证。

III. 时间性能测试

1. 策略三的剪枝效果

测试数据集：城市数N = 20，交通工具数E = 280，必经城市数S = 4，时间限制T = 47h

使用的剪枝	所用时间	加速率
无剪枝	8.03	0
朴素剪枝	6.03	24.9%
启发式剪枝	2.6	67.6%

表2. 算法三剪枝效率分析

加速率为加了剪枝后比无剪枝所用的时间差与无剪枝总时间的比值。

从表2中可以看出，设计出的启发式剪枝能够有效加快算法的效率。

2. 算法的时间复杂度

在第三节算法设计中我们已经提到，算法一算法二的主要瓶颈在于预处理的耗时，而模拟退火过程只需要很短的常数时间。考虑到在实际交通网中图的连通度很高，而算法二所需要的预处理时间复杂度为 $O(24*v*e)$ 。所以我们的算法最大能够承受大概200个城市的的数据集。

而策略三的耗时取决于多个参数：必经城市数、图的大小、时限。在具体测试中我们发现，在时限不大、必经城市少的时候，策略三甚至也可以跑V = 100的数据集，但是当时限较大、必经城市较多的时候，策略三的求解就显得耗时极大，甚至无法跑出结果，而这是由搜索状态空间随着图的增长呈指数增长所必然决定的。

六、Qt 图形界面

1. 开发环境

Qt5.6 + Qt Creator + Qt Designer / Qt5.6 + visual studio2015

2. 使用说明

主机配置：2736*1824分辨率 文本放大比率200%下的效果

A. 主视图功能与效果

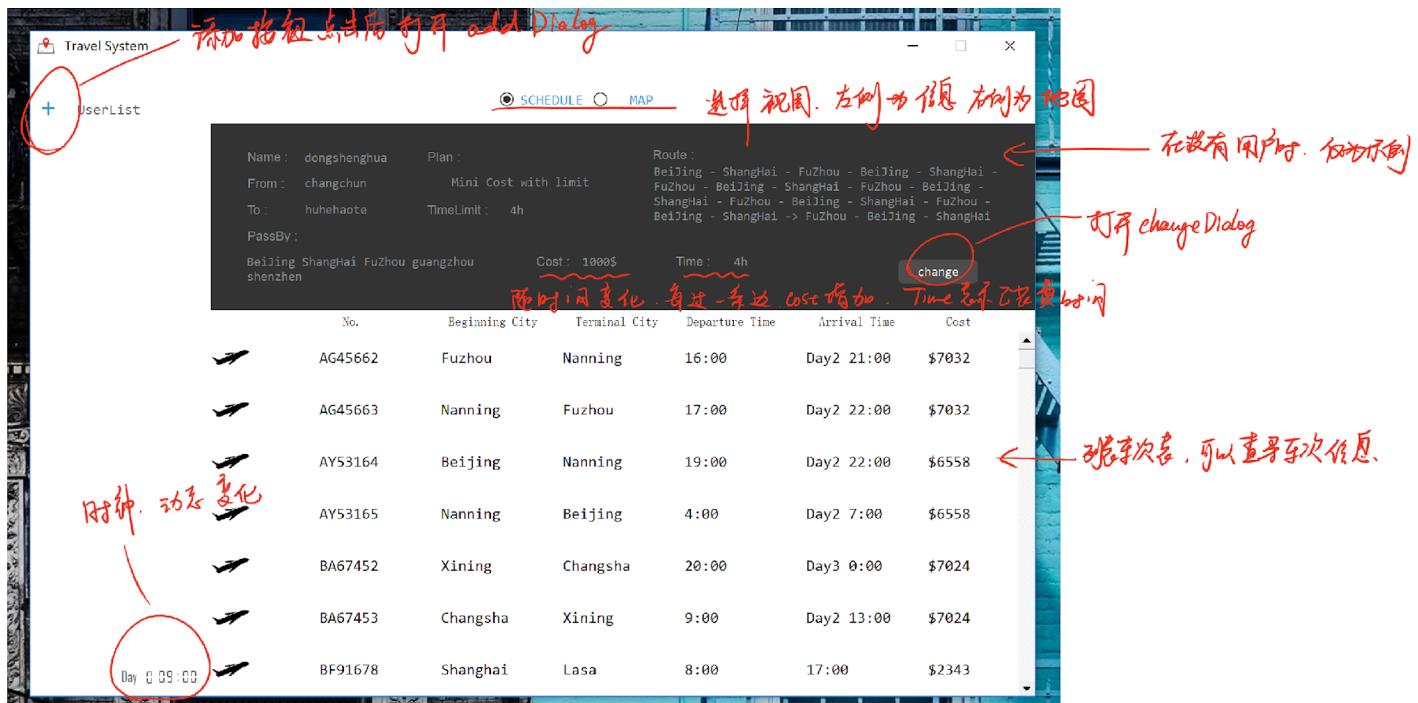


图6 主视图界面介绍

B. 点击左上角“+”，显示添加新用户的要求对话框。需要按照输入规则输入路线规划要求。假如输入为假则弹出Error信息，并重新提示输入规则。

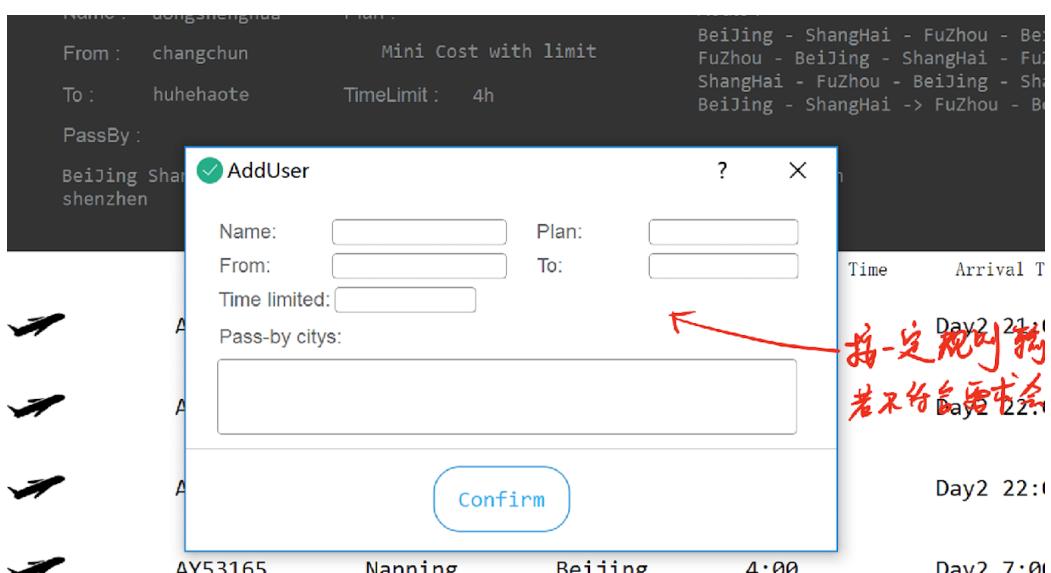


图7 创建用户对话框

C. Error信息对话框。提示输入规则。

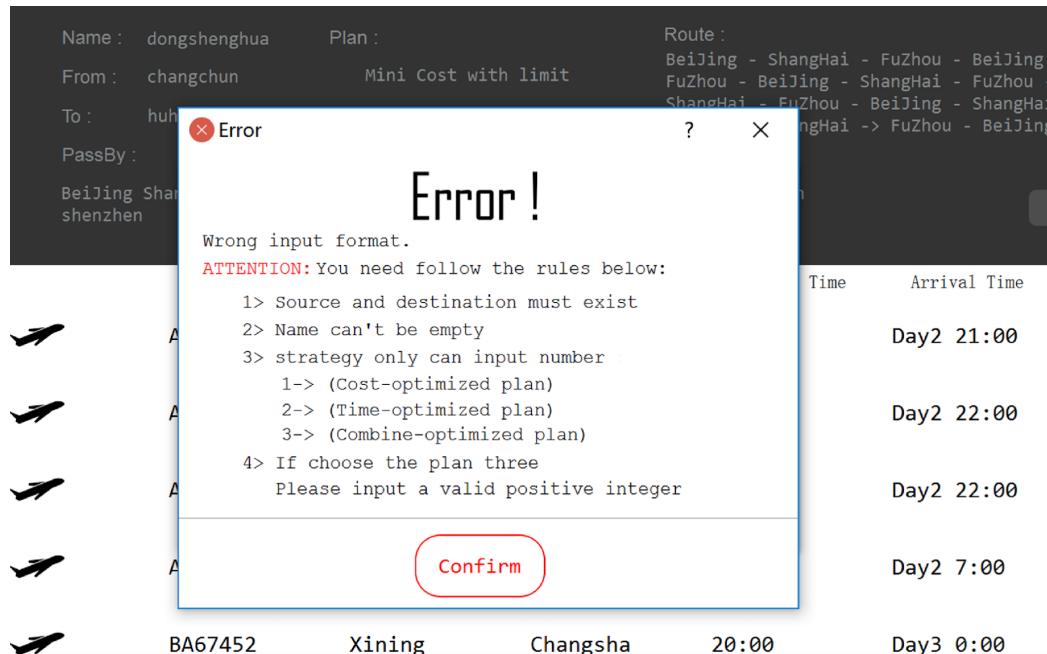


图8 错误信息对话框

D. 输入完毕后进入等待界面。防止由于计算路径花费时间过长而导致的程序伪卡死，提示用户等待并停止输入。最长时间设定为2s（可以设置时长）。

Waiting for caculating route

E. map信息栏以及动态变化的map。其中当前所在的路线用会闪烁的蓝色光晕效果指明，当前点使用大小变化的蓝色光点，到达目的地以后使用红色光点标识。地图上方依次显示用户名、出发地、目的地、当前位置与下一座城市。

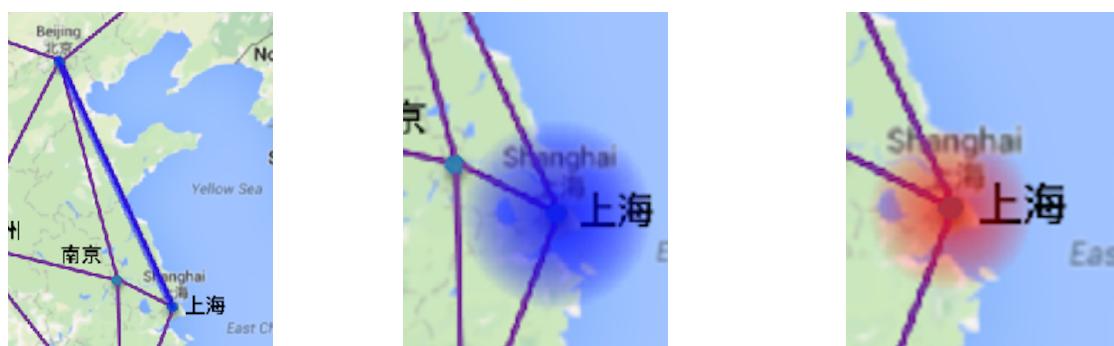


图9 旅行状态表示示意图

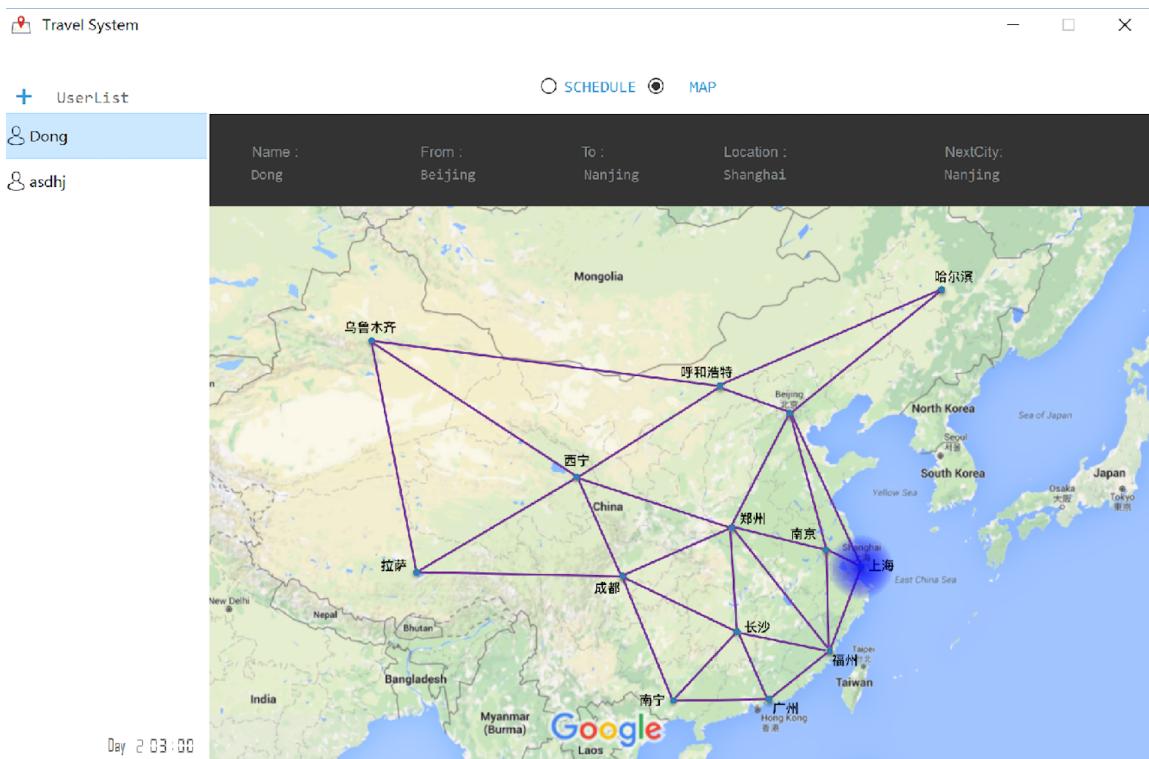


图10 地图选项卡下整体效果图

F. 如果需要更改路线，在计划选项卡下点击“Change”输入对应的信息，单击“Confirm”即可。

Time	Arrival Time	Cost
Day2 21:00		\$7032
Day2 22:00		\$7032
Day2 22:00		\$6558
Day2 7:00		\$6558
Day3 0:00		\$7024
Day2 13:00		\$7024

图10 修改路线对话框

3. 程序结构

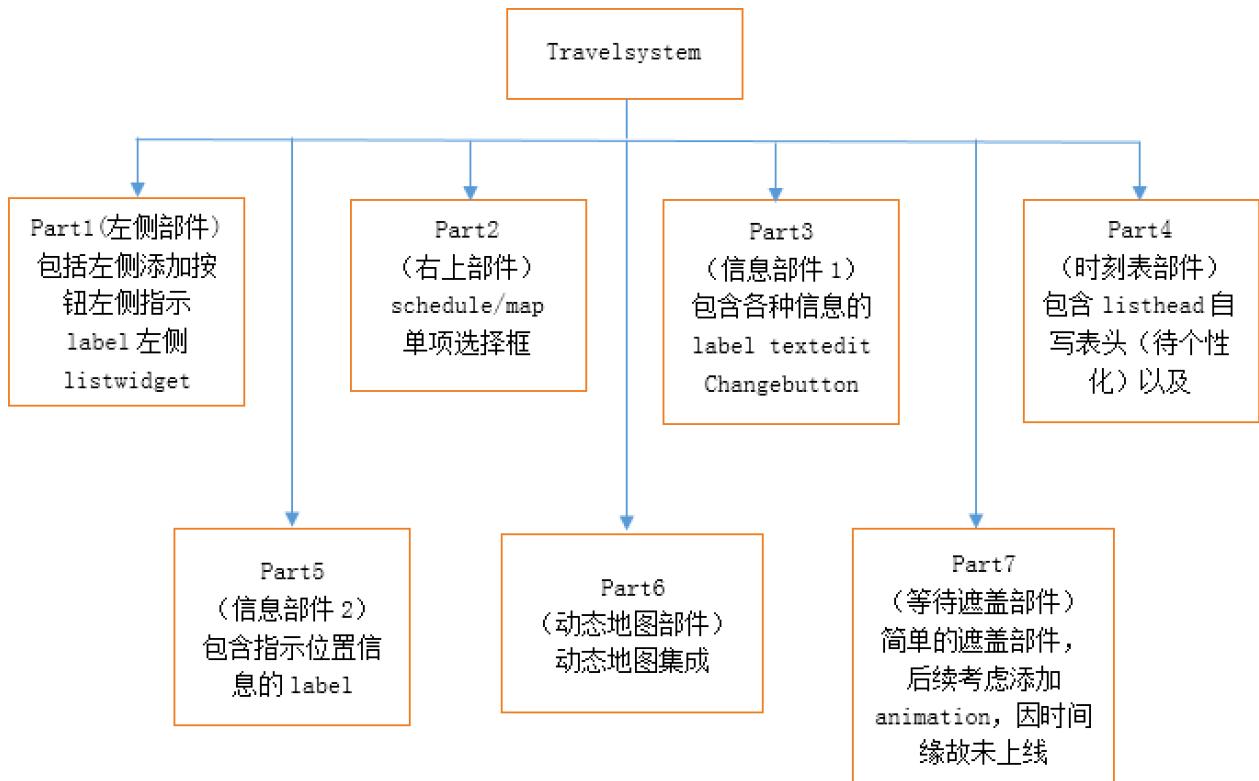


图11 图形界面框架

七、总结

综上所述，我们设计的“模拟旅行查询系统”有以下优点：

- a) **快速。**通过使用高效率的模拟退火算法，使得我们的程序能够以非常快的速度求解出正确的答案。而这也保证了为实现美观的图形化预留了资源，保证了用户体验。
- b) **美观。**我们使用了最新的 Qt5.6.0 实现我们的图形化界面。使用信号槽、结合业界中已有的资源，我们设计并实现了一个美观且实用的图形化界面。

但是，目前我们的软件还有一些不足，在于：

- a) **分辨率不能够自适应。**考虑到开发的方便，我们将分辨率设为了固定的值，而这可能会造成部分用户的体验不畅。
- b) **没有密码，缺少安全性。**即用户可以随意查看他人的信息而不需要获得许可，而这可能会造成隐私泄露。

附录

1. 算法二正确性代码片段

```
//client是指向乘客的指针
if (client->plan.type == 2)
{
    client->plan.type = 3;
    for (int i = 1;; i++)
    {
        //清空当前路径 赋值时间限制
        client->route.clear();
        client->plan.timeLimit = i;

        //判断在此时间限制内可否得到合法解
        Design_Route(*client);
        if (!client->route.empty())
        {
            cout << "the right answer of plan 2 should be "
            << i << "h" << endl;
            Print_Route(client->plan.source, client-
                        >route);
            break;
        }
    }

    //复原client
    client->plan.type = 2;
    client->plan.timeLimit = 0;
}
```

其余代码均可见于压缩包中:)