基于遗传算法的家具配送安装方案

古宜民　　郝中楷　　赵作竑

2018 年 1 月 2 日

摘要

在物流的快速发展的今天，如何合理安排车辆，在满足用户需求的同时尽量减少配送时间成为公司的重要工作。本文依据题目所以给数据研究货车配送方案，并建立模型进行求解。

问题一　本文首先对数据进行预处理，去除不合理的数据点，并用 Floyd 算法求出两目的地间最短距离，之后建立了虚拟目的地模型，解决同一目的地有多个客户的问题。最终采用遗传算法，将各辆货车的配送路线进行编码，针对问题一分别对编码、杂交、变异、选择等各个环节进行了优化，在若干代筛选后得到了问题一的近似优解。

问题二　由于需要考虑时间窗口的影响，而遗传算法具有一定的盲目性，为了提高算法效率，设计了构造算法。该构造算法能够高效地得出满足时间窗口的合理解，但是无法对解的质量做出保证。构造算法与遗传算法结合起来，分别避免两种算法的缺点，使用构造算法产生符合条件的初始群体，并通过遗传算法进行自交变异，达到优化的目的，最终得到了满足条件的近似优解。

问题三　对问题一与问题二中建立的模型进行稳定性分析，同时考虑货车在路上与安装的时间变化时，原来的解能否直接应用于新的条件。结果表明问题一与问题二的模型稳定性良好，在绝大多数情况下均能不做改动且符合条件。

关键字　遗传算法　　虚拟目的地　　分割　　多旅行商问题（MTSP）

目录

1 文献综述

问题一是以经典的多旅行商问题的变种。

旅行商问题（Travelling Salesman Problem, TSP）是典型的 NP 类问题。已有许多针对旅行商问题的解法，如最近邻算法、贪心算法、边交换算法、Lin-Kernighan-Helsgaun 算法等，还有许多启发式算法，如爬山算法、模拟退火算法、链式局部最优化、遗传算法、蚁群算法等。其中取得最显著成效的是丹麦人 Keld Helsgaun 和日本人永田裕一。

本题是以多旅行商问题（MTSP）为基础建立的。对于多旅行商问题，现在广泛应用的算法，如分支定界法、剥脱法、k-交叉法等，均能够对此类问题求解，然而这些算法普遍存在效率不高、适应性弱等缺点，在数据量较大时难以进行准确计算。

美国 Michigan 大学的 Holland 教授提出的遗传算法（Generic Algorithm，GA）是求解复杂的优化组合问题的有效方法，其思想来源于生物学的进化论。根据一般性的遗传算法，可以精心设计出威力强大的启发式算法，结合局部搜索方法使用则效果更佳。蚁群算法、模拟退火算法也是著名的启发式算法，但是遗传算法与它们相比，遗传算法的执行效率更佳，易于实现。

经过比较认为遗传算法是一种相对合适的解法，故本文选择遗传算法，并对其进行改进，以解决问题。

2 问题重述

2.1 问题背景

A 公司是一个家居产品物流配送企业。公司从商家获得客户的订单信息（包括产品品类、数量、体积等）。在配送之前，公司提前一天预约客户获得配送的时间窗口，并且在配送到达之后帮助客户安装家居产品。

为了在尽短的时间里完成配送任务，提高效率，公司决定统筹安排车辆的配送路线。公司需要多辆货车进行产品配送，各辆货车依次到达各个客户，并进行产品安装，直到完成配送任务。

2.2 相关信息

附件 1:《配送汇总表》包括了各个客户预约时间、安装需要的时间以及货物的体积。

附件 2:《目的地距离矩阵》中标明了各个目的地间的距离。

2.3 需要解决的问题

1. 假设不考虑时间窗口，近似认为每笔订单需要的安装时间确定，要求建立数学模型设计配送路线，使配送安装任务尽快完成，并利用数据给出数值模拟结果。

2. 考虑客户的配送时间窗口限制，假设每笔订单需要的产品安装时间确定，要求建立数学模型设计配送路线，使得配送安装任务尽快完成，并利用数据给出算例结果。

3. 考虑配送和产品安装时间的不确定，要求建立数学模型来设计配送路线，使得配送安装任务尽快完成，并给出算法设计。

3 问题分析

3.1 问题的整体分析

问题一并不考虑时间窗口以及不确定因素，属于理想化模型，而问题二、三分别加入了时间窗口的限制因素以及不确定性因素。

首先对原始数据进行预处理。之后建立模型解决问题一，再以问题一的遗传算法模型为基础，加入适当的修改，使问题一的模型能够适用于问题二。最后对问题一与问题二的模型进行稳定性分析，得到问题三的答案。

3.2 问题一的分析

问题一是多旅行商问题的变形。故决定采用遗传算法对问题一进行求解。

通过对选择算子与变异算子的设计以及对遗传算法中的参数的适当选取，在经过若干代的变异与筛选后，能够得到较优解。

3.3 问题二的分析

问题二在问题一的基础上增加了时间窗口的限制条件。对问题一的遗传算法模型进行适当的修改，并加入构造合理路线的算法，将两种算法结合起来，解决问题二。

3.4 问题三的分析

问题三在问题二的基础上增加了不确定性因素，故将堵车、工人休息以及安装时间的变化等影响时间的因素，对应为路长、安装时间等元素的不确定的变化，再对修改后的模型进行稳定性分析。

4 模型假设

1. 假设所有车辆只选择附件 2 中所给的路线，即车辆在任意两目的地间行驶的距离均可由附件 2 直接或间接得到。

2. 不考虑各车辆携带的工作人员的休息时间。

3. 忽略工作人员上车、下车，以及车辆发动、停车的时间。

4. 假设在一般情况下所有车辆平均速度均为 60 $\mathrm{km{\cdot}h^{-1}}$。

5 符号说明

符号	意义
j	货车的计数器
$\{a'\}$	所有虚拟目的地的集合
$\{a_j(i)\}$	货车经过的目的地的集合
n_j	货车 j 经过目的地的数目
$a(0)$	出发点
$m_j(i)$	货车经过的第 i 个目的地的目标运载量
$d_j(i,i+1)$	目的地 $a_j(i)$、$a_j(i+1)$ 之间的距离
$t_j(i,i+1)$	货车在 $a_j(i)$ 与 $a_j(i+1)$ 间移动所用时间
$\tau_j(i)$	货车在目的地 $a_j(i)$ 安装所用时间
T_j	第 j 辆货车完成配送任务所用的总时间
$T_j(i)$	第 j 辆货车到达虚拟目的地 $a_j(i)$ 时的时间
T	所有货车完成任务所用的最短时间
Δt	时间窗口的长度
$E(a_j(i))$	目的地 $a_j(i)$ 的预约时间
$\delta t_j(i,i+1)$	在路上时间的不确定度
$\delta\tau_j(i)$	在安装过程中时间的不确定度
$\lambda_j(i)$	货车 j 在路上所用时间的不确定
$\mu_j(i)$	货车 j 安装过程所用时间的不确定
$w_j(i)$	货车 j 在目的地 i 处等待的时间

6 模型的建立与求解

6.1 问题一

所有 n 个目的地可以表达为一个集合 $\{a\}$，问题要求所有货车到达各个目的地进行送货，总时间最短。货车 j 经过的路线用数列 $\{a_j(i)\}$ 表示。时间消耗的途径有两种：一是在路上，即从目的地 $a_j(i)$ 到目的地 $a_j(i+1)$ 之间移动消耗时间 $t(j, a_j(i), a_j(i+1))$；二是在目的地 $a_j(i)$ 安装家居产品消耗时间 $\tau_j(i)$。记货车 j 经过目的地的数目为 n_j 所以，货车 j 消耗时间之和 $T(j)$ 可以表示为：

$$T(j) = \sum_{i=0}^{n_j-1} t_j(i, i+1) + \sum_{i=1}^{n_j} \tau_j(i) \tag{6.1}$$

路线规划是要找到一种分割方法，将集合 $\{a_k\}$ 中的所有的元素无重复、无遗漏地分为 14 个数列。而最短时间其实取决于各辆货车的时间 $T(j)$ 的最大值，即

$$T = \max T_j$$

故所谓时间最短，便意味着该分割使 $\max T_j$ 取得最小值。求出该分割方法，问题一也得到了解决。

但是这里仍然存在一些问题。

首先，在任意两目的地间的距离与行走的方向无关。

$$\forall a(i), a(i+1) \in \{a\},\ d_j(i, i+1) = d_j(i+1, i) \tag{6.2}$$

然而经过观察发现，附件 2 中的数据并不完全符合式 6.2。所以首先对数据进行预处理。

货车 j 从目的地 $a_j(i)$ 到下一目的地 $a_j(i+1)$ 消耗的时间可以表述为式 6.3：

$$t_j(i, i+1) = \frac{d_j(i, i+1)}{\overline{v}} \tag{6.3}$$

当分割使 $\max T(j)$ 取得最小值时，时间 T 最短，所以在 $\tau_j(i)$ 不变的情况下，应尽可能地缩小 $\sum_{i=0}^{n_j-1} t_j(i, i+1)$。在第一问的简化条件下，可以认为 $\overline{v} = 60\ \mathrm{km{\cdot}h^{-1}}$，故应该在进行分割前确保 $d(i, i+1)$ 已达到最小，这样对后续步骤的处理起到简化作用。从附件 2 所给的数据看，无法确保在原始数据中 $d(i, i+1)$ 为可能路线中的最小值，所以建立最短路模型，求出任意两目的地间的最短距离 $\min d(i, i+1)$。此时，式 6.3 便可以表示为如下形式：

$$t_j(i, i+1) = \frac{\min d_j(i, i+1)}{\overline{v}} \tag{6.4}$$

另外，在数据中存在同一目的地有多名客户需要配送货物的情景。通过建立虚拟目的地模型可以方便地处理此种情景。求得的目的地的集合可以表示为 $\{a'\}$。实际上，分割应该是对新的集合 $\{a'\}$ 进行的。

在此之后，便可以开始考虑对集合 $\{a'\}$ 的分割了。采用遗传算法求解，在对分割方案进行编码后，选定参数，经过数代筛选，最终能够得到近似优解。

同时要注意，货车 j 经过的目的地数列 $\{a_j(i)\}$ 要求的运载量为 $\sum_{i=1}^{n_j} m_j(i)$，若记 M 为货车的最大运载量，则要注意

$$\sum_{i=1}^{n_j} m_j(i) < M,\ j = 1, 2, 3, \ldots, 14 \tag{6.5}$$

至此，可得到问题一的答案。

6.1.1 数据的预处理

在一般情况下，任意两目的地 $a(i)$，$a(i+1)$ 必定符合式 6.2，然而在附件 2 的数据中，可以明显观察到，部分数据点并不符合式 6.2 的规律，表 1 就清晰地显示了这一点。

因此，以右上半部分的数据为准，舍弃左下半部分的数据，使其保持一致，采用右上半部分的数据进行讨论、计算。修改后的数据示例如表 2 所示。

以预处理后的数据为基础，建立进一步的模型。

6.1.2 最短路模型

从式 6.4 可以看出，求出两点之间的最短长度 $\min d_j(i, i+1)$ 是十分必要的，因此使用 Floyd 算法[1]求出任意两点之间的最短长度，进而根据式 6.4 计算出货车在任意两点间移动的最短时间 $t_j(i, i+1)$。

从表 3 与表 4 的对比可以看出，附件 2 中所给的两目的地间的距离并不是两点之间的最短长度，使用 Floyd 算法十分高效地算出了两个目的地间的最短路径。

6.1.3 虚拟目的地模型

在求得任意两目的地间的最短路径长度，进而得到货车 j 在两目的地间移动所需最短时间 $t_j(i, i+1)$ 后，考虑以下问题：

按照已有的描述，目的地 a 只可能属于一辆货车 j。然而，从表 5 可以看出，存在多个客户在同一目的地的情况。已有的方案会把该目的地 a 的客户全部划分给一辆货车，这是不合理的，不同的客户应该有可能分属于不同的货车。然而客户的属性却与目的地本身联系紧密，因为客户之间的距离等于目的地之间的距离。为解决这一矛盾冲突，考虑引入“虚拟目的地”的概念。对于任意目的地 a，记该目的地等待配送的客户个数为 s_a。一般情况下 $s_a = 1$，但 $\exists s_a > 1$。为描述方便，记 $s_a > 1$ 的目的地中的一个为 $a(s)$，则有 $s_{a(s)} > 1$。

对 $a(s)$，将其分为 $s = s_{a(s)}$ 个虚拟目的地 $a(s)_1, a(s)_2, \ldots, a(s)_s$。

[1]代码见附录 C

	0	1	2	3	4	5	6	7	8	9	10
0	0	25.4	29	38.9	16.4	31	23.8	31	26.6	33.5	22.5
1	25.4	0	3.6	13.4	13	9.7	9.2	9.7	4.5	7.9	7.9
2	29	3.6	0	13.6	14.4	8.3	7.3	8.3	3.1	5	6.6
3	38.9	13.4	13.6	0	25.6	20.1	21.1	20.1	16.2	13.9	19.8
4	16.4	13	14.4	25.6	0	16	7.1	16	11.4	18.3	7
5	31	9.7	8.3	20.1	16	0	8.1	0.1	6.7	8.3	9.4
6	23.8	9.2	7.3	21.1	7.1	8.1	0	8.3	5.6	10.4	2.2
7	31	9.7	8.3	201	16	0.1	8.3	0	6.7	8.3	9.4
8	26.6	4.5	3.1	16.2	11.4	6.7	5.6	6.7	0	6.3	4.7
9	33.5	7.9	6.6	29.8	7	9.4	2.2	9.4	4.7	0	9.5
10	22.5	7.9	6.6	19.8	7	9.4	2.2	9.4	4.7	9.5	0

表 1: 经预处理之前，附件 2 中的部分数据。

可以看到，左下半部分的数据与右上半部分的数据并不一致。

	0	1	2	3	4	5	6	7	8	9	10
0	0	25.4	29	38.9	16.4	31	23.8	31	26.6	33.5	22.5
1	25.4	0	3.6	13.4	13	9.7	9.2	9.7	4.5	7.9	7.9
2	29	3.6	0	13.6	14.4	8.3	7.3	8.3	3.1	5	6.6
3	38.9	13.4	13.6	0	25.6	20.1	21.1	20.1	16.2	13.9	19.8
4	16.4	13	14.4	25.6	0	16	7.1	16	11.4	18.3	7
5	31	9.7	8.3	20.1	16	0	8.1	0.1	6.7	8.3	9.4
6	23.8	9.2	7.3	21.1	7.1	8.1	0	8.3	5.6	10.4	2.2
7	31	9.7	8.3	20.1	16	0.1	8.3	0	6.7	8.3	9.4
8	26.6	4.5	3.1	16.2	11.4	6.7	5.6	6.7	0	6.3	4.7
9	33.5	7.9	5	13.9	18.3	8.3	10.4	8.3	6.3	0	9.5
10	22.5	7.9	6.6	19.8	7	9.4	2.2	9.4	4.7	9.5	0

表 2: 在预处理之后，附件 2 中的部分数据。

以右上半部分的数据为准，覆盖了左下角的数据。此时数据满足式 6.2。

	0	1	2	3	4	5	6	7	8	9
0	0	25.4	29	38.9	16.4	31	23.8	31	26.6	33.5
1	25.4	0	3.6	13.4	13	9.7	9.2	9.7	4.5	7.9
2	29	3.6	0	13.6	14.4	8.3	7.3	8.3	3.1	5
3	38.9	13.4	13.6	0	25.6	20.1	21.1	20.1	16.2	13.9
4	16.4	13	14.4	25.6	0	16	7.1	16	11.4	18.3
5	31	9.7	8.3	20.1	16	0	8.1	0.1	6.7	8.3
6	23.8	9.2	7.3	21.1	7.1	8.1	0	8.3	5.6	10.4
7	31	9.7	8.3	201	16	0.1	8.3	0	6.7	8.3
8	26.6	4.5	3.1	16.2	11.4	6.7	5.6	6.7	0	6.3
9	33.5	7.9	6.6	29.8	7	9.4	2.2	9.4	4.7	0

表 3: 附件 2 中的部分原始数据示例，许多数据并不是最短路径。

	0	1	2	3	4	5	6	7	8	9
0	0.0	25.0	27.9	29.6	16.4	30.3	23.0	30.3	26.6	31.5
1	25.0	0.0	3.6	13.4	8.6	9.7	8.6	9.7	4.5	7.7
2	27.9	3.6	0.0	13.6	11.5	8.2	7.3	8.2	3.1	4.7
3	29.6	13.4	13.6	0.0	13.2	19.7	16.5	19.7	15.74	13.84
4	16.4	8.6	11.5	13.2	0.0	14.2	7.1	14.2	11.4	15.1
5	30.3	9.7	8.2	19.7	14.2	0.0	8.1	0.1	6.6	8.3
6	23.0	8.6	7.3	16.5	7.1	8.1	0.0	8.1	5.6	10.1
7	30.3	9.7	8.2	19.7	14.2	0.1	8.1	0.0	6.6	8.3
8	26.6	4.5	3.1	15.74	11.4	6.6	5.6	6.6	0.0	6.0
9	31.5	7.7	4.7	13.84	15.1	8.3	10.1	8.3	6.0	0.0

表 4: 经过 Floyd 算法处理的原始数据，两点之间路径的长度显著缩短了。

序号	配送日期	起运地	目的地	运输类别	现用车次	预约时间	安装时间	体积
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
2	7 月 1 日	0	2	送货	6-2	15:00	3	0.5
23	7 月 1 日	0	2	送货	6-3	16:00	3	0.8
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
4	7 月 1 日	0	4	送货	8-1	8:00	3	0.5
28	7 月 1 日	0	4	送货	9-5	15:10	2	0.9
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
22	7 月 1 日	0	22	送货	12-7	15:00	0	0.5
35	7 月 1 日	0	22	送货	12-5	11:40	0	1.2
46	7 月 1 日	0	22	送货	12-4	11:30	1	1.3
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮

表 5: 目的地有多个客户的情况相当普遍

对 $\forall\, a \neq a(s)$，有

$$d\big(a(s), a\big) = d\big(a(s)_k, a\big),\ k = 1, 2, \ldots, s \tag{6.6}$$

按照式 6.6 ，也可将所有 $s_{a(s)} = 1$ 的目的地视为对应一个虚拟目的地 a_1。那么所有虚拟目的地可以构成一个集合 $\{a'\}$。易证集合 $\{a'\}$ 与集合 $\{a\}$ 有如下关系：

$$d\big(a(s)_k, a(s)'_l\big) = d(s, l)$$
$$d\big(a(s)_k, a(s)\big) = 0$$

其中

$$k = 1, 2, \ldots, s_a$$
$$l = 1, 2, \ldots, s'_a$$

此时，虚拟目的地集合 $\{a'\}$ 的元素个数为 $n' = \sum_{i=1}^{n} s_a(i)$。

在建立了虚拟目的地模型后，问题一即可转化为：寻求一种分割方法，将集合 $\{a'\}$ 的 n' 个元素全部无重复、无遗漏地组成 14 个子列

$$\{a_1\}, \{a_2\}, \{a_3\}, \ldots, \{a_j\}, \ldots, \{a_{14}\},\ j = 1, 2, 3, \ldots, 14$$

每个子列 $\{a_j\}$ 表示货车 j 送货的路径。子列 $\{a_j\}$ 中每两相邻元素 $a_j(i)$，$a_j(i+1)$ 间的最短距离 $\min d(i, i+1)$ 已在最短路模型中求出。通过式 6.1 即可求出各辆货车的时间 T_j，从而得到总时间 $T = \max T_j$。

6.1.4 遗传算法求解模型

根据此题特点，采用遗传算法对集合 $\{a'\}$ 进行分割并求解问题一。我们分别对编码方案、遗传算子、选择算子以及杂交算子进行了设计，具体如下：

编码方案 我们需要对代表 14 辆货车移动路径的 14 个数列 $\{a_j\}$ 进行编码。

首先介绍我们采用的具体编码方案。一个具体、简短的例子如下：

货车 1 依次经过虚拟目的地 5，7，2，14，9，货车 2 依次经过 12，3，1，58 号虚拟目的地，货车 3 依次经过 31，65，22 号虚拟目的地。则该种送货路线一种可能的表示为

$$\begin{array}{cccccccccccc} 0 & 0 & 5 & 0 & 7 & 0 & 2 & 0 & 0 & 14 & 9 & 0 \\ 0 & 12 & 0 & 0 & 0 & 0 & 0 & 3 & 1 & 0 & 0 & 58 \\ 31 & 0 & 0 & 65 & 0 & 22 & 0 & 0 & 0 & 0 & 0 & 0 \end{array}$$

这种编码方案最终得到二维数组 A，其行数为 14，列数为 n'。记第 j 行为 R_j，第 x 列为 C_x。则二维数组中任意数可用行、列表示为 $A_{x,j}$。将 14 辆货车的路线 $\{a_j\}$ 编码为 14 个横行 R_j，每行将数列 $\{a_j\}$ 表示为一部分 0 与一部分正整数，构成一个 n' 位非负整数串。在上例中，若去掉每行中的 0，则该行变为货车经过的虚拟目的地的编号的数列 $\{a_j\}$：

$$\begin{array}{ccccc} 5 & 7 & 2 & 14 & 9 \\ 12 & 3 & 1 & 58 & \\ 31 & 65 & 22 & & \end{array}$$

而对于 $A_{x,j}$ 中的每一列 C_x，该列中仅有一个非 0 数。

此种编码方案主要是考虑到方便编程实现而设计的。

遗传算子 设计专门的遗传算子如下：

选择算子 选择算子的好坏影响遗传算法的收敛性，选择压力过大容易导致未成熟收敛，选择压力过小又容易导致收敛速度过慢。为解决此问题，采用锦标赛选择（Tournament Selection），并做出适应此题的改进，保留历史上的最优的 25 个个体，取代当前群体中最差的 25 个个体，从而防止优良个体被破坏。改进后的锦标赛选择如下：

1. 从个体数目为 N 的群体中随机选取两个个体 G_1，G_2；
2. 比较两者适应值，得到适应值较大者 G_M；
3. 将 G_M 放入新的群体当中。
4. 重复步骤 1- 3 共 N 次，得到个体数目为 N 的新群体。
5. 将历史上最好的 25 个个体代替新群体中最差的 25 个个体；

6. 更新历史上最好的 25 个个体。

经过改进后的锦标赛选择可以在避免未成熟收敛的同时防止优良个体的基因被破坏。

杂交算子 我们采用由 Davis 等人提出的次序交叉算法（OX, Order Crossover）作为杂交算子。

次序交叉算法的具体步骤如下：

1. 将群体中代表每个个体的基因的二维数组 A 进行压缩，生成长度为 n' 的整数串；

   例如，二维数组

$$\begin{matrix} 0 & 0 & 5 & 0 & 7 & 0 & 2 & 0 & 0 & 14 & 9 & 0 \\ 0 & 12 & 0 & 0 & 0 & 0 & 0 & 3 & 1 & 0 & 0 & 58 \\ 31 & 0 & 0 & 65 & 0 & 22 & 0 & 0 & 0 & 0 & 0 & 0 \end{matrix}$$

   经过压缩后，可以得到下列整数串：

$$31\ 12\ 5\ 65\ 7\ 22\ 2\ 3\ 1\ 14\ 9\ 58$$

2. 将群体中所有个体进行配对，每两个个体分为一组；

3. 对于每一对，分别执行步骤 4-7 。

4. 按照杂交概率 p_c，随机判断该组是否发生杂交。若发生杂交，执行步骤 5-7 ，否则跳过该组；

5. 随机选择两杂交点 u，v，并要求 $1 \le u \le n'$，$1 \le v \le n'$；

6. 交换杂交段的起点 u 与终点 v 间的基因；

7. 对于杂交后的个体，分别进行式 6.5 的检验，若不符合，即为超载的情况。此时杀死该个体，重新随机生成一个符合式 6.5 的个体。

8. 将新群体中代表每个个体基因的整数列还原为二维数组。

其中步骤 6 相对较为复杂，可以通过下例说明：

个体 B_1 与个体 B_2 配对后被分配到了一组，个体 B_1、B_2 的基因经压缩后为

$$\begin{array}{c|ccccccccc} B_1 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\ B_2 & 9 & 8 & 7 & 6 & 5 & 4 & 3 & 2 & 1 \end{array}$$

随机生成的杂交段起点 $u = 3$，终点 $v = 5$。记杂交后两个体的压缩后的基因为 B_1' 与 B_2'。

首先，交换杂交段 u、v 间的基因，其余部分留空；

B_1'	_	_	7	6	5	_	_	_	_
B_2'	_	_	3	4	5	_	_	_	_

其次，将 B_1、B_2 位于第 $v+1$ 到第 n' 个整数以及第 1 到第 v 个整数，按顺序排成一列，生成两个整数列 D_1、D_2；

D_1	6	7	8	9	1	2	3	4	5
D_2	4	3	2	1	9	8	7	6	5

在 D_1 中的基因与 B_1' 中交换得到的新基因有所重复，在此例中，5、6、7 便是重复的；同样，D_2 中的基因与 B_2' 中交换得到的新基因同样也有重复，即 3、4、5。为避免重复，去掉 D_1 与 D_2 中重复的虚拟目的地编号，得到 D_1' 与 D_2'：

D_1'	8	9	1	2	3	4
D_2'	2	1	9	8	7	6

将 D_1' 填入 B_1' 中第 v 个整数之后，余下填到第 1 到第 $u-1$ 个数；将 D_2' 填入 B_2' 中第 v 个整数之后，余下也填到第 1 到第 $u-1$ 个数，就得到了完整的 B_1' 和 B_2'。

B_1'	3	4	7	6	5	8	9	1	2
B_2'	7	6	3	4	5	2	1	9	8

以上为杂交的过程。

变异算子 我们设计了两种变异算子：倒位与交换。

1. **倒位**：在变异个体中随机地选择两个倒位点，并将两倒位点之间的部分的非 0 值的顺序颠倒过来。

   示例如下：设原基因经压缩后为

   31 22 5 65 7 22 2 3 1 14 9 58

   设倒位点为 5 和 9，则经过倒位变异的基因为

   31 22 5 65 1 3 2 22 7 14 9 58

2. **交换**：交换要达到的直接目的是将 C_{x_1} 与 C_{x_2} 中的整数进行交换。

   首先在变异个体中随机地选择两列 C_{x_1} 与 C_{x_2}；

   其次将 C_{x_1} 中的一个 0 值 A_{x_1,j_1} 和 C_{x_2} 中的一个非 0 值 A_{x_2,j_2} 进行交换，即 $A_{x_2,j_2} \to A_{x_1,j_1}$，$0 \to A_{x_2,j_2}$；

   然后，在 C_{x_1} 中，必然 $\exists A_{x_1,j_1'} \neq 0$，$j_i' \in \{1,2,3,\ldots,14\} - \{j_1\}$，随机选择 $j_2' \in \{1,2,\ldots,n'\}$，将 $A_{x_1,j_1'}$ 的值与 $A_{x_2,j_2'}$ 的值进行交换，即 $A_{x_1,j_1'} \to A_{x_2,j_2'}$，$0 \to A_{x_1,j_1'}$。

   一个具体例子是，在二维数组

$$\begin{matrix} 0 & 0 & 5 & 0 & 7 & 0 & 2 & 0 & 0 & 14 & 9 & 0 \\ 0 & 12 & 0 & 0 & 0 & 0 & 0 & 3 & 1 & 0 & 0 & 58 \\ 31 & 0 & 0 & 65 & 0 & 22 & 0 & 0 & 0 & 0 & 0 & 0 \end{matrix}$$

中，首先随机得到 $x_1 = 6$，$x_2 = 8$，所以要交换的两列分别为 $C_{x_1} = C_6$，$C_{x_2} = C_8$；再在确保 $A_{6,j_1} = 0$ 的前提下随机得到 $j_1 = 1$；又因为 $A_{8,2} \neq 0$，所以选定 $j_2 = 2$。进行交换：$A_{x_2,j_2} \to A_{x_1,j_1}$，$0 \to A_{x_2,j_2}$，得到如下结果：

$$\begin{matrix} 0 & 0 & 5 & 0 & 7 & 3 & 2 & 0 & 0 & 14 & 9 & 0 \\ 0 & 12 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 58 \\ 31 & 0 & 0 & 65 & 0 & 22 & 0 & 0 & 0 & 0 & 0 & 0 \end{matrix}$$

接着，在 C_6 中，存在 $A_{6,3} \neq 0$，所以 $j_1' = 3$；在 C_8 中，所有的整数都为 0，故随机确定 $j_2' = 3$。

$$\begin{matrix} 0 & 0 & 5 & 0 & 7 & 3 & 2 & 0 & 0 & 14 & 9 & 0 \\ 0 & 12 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 58 \\ 31 & 0 & 0 & 65 & 0 & 22 & 0 & 0 & 0 & 0 & 0 & 0 \end{matrix}$$

进行第二次交换，$A_{x_1,j_1'} \to A_{x_2,j_2'}$，$0 \to A_{x_1,j_1'}$，得到最终的结果：

$$\begin{matrix} 0 & 0 & 5 & 0 & 7 & 3 & 2 & 0 & 0 & 14 & 9 & 0 \\ 0 & 12 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 58 \\ 31 & 0 & 0 & 65 & 0 & 0 & 0 & 22 & 0 & 0 & 0 & 0 \end{matrix}$$

这便是变异算子中交换方式变异的具体步骤的示例。

适应性 适应值函数采用双关键字对个体的适应度进行排名。

首先考察 T 指标，$T = \max T_j$，即为 14 辆货车中用时最长的一辆所用时间。该指标越小适应性越强。

若两个体 T 相同，则继续检查所有货车所用的时间之和 $\sum_{j=1}^{14} T_j$，该值越短，则个体的适应性越强。

遗传算法中的参数

群体规模 N 群体规模 N 影响到算法的性能与效率，N 过小会导致效果不佳，N 过大则计算量过大，收敛过慢。故取 $N = 100$。

杂交概率 $\mathbf{p_c}$ 与变异概率 $\mathbf{p_m}$ 杂交概率 p_c 控制杂交的频率，一般 p_c 选取较大值，本文选取 $p_c = 100\%$。

p_m 控制随机变异的幅度，一般相对 p_c 较小。选取 $p_m = 40\%$，倒位与交换各为 $0.5p_m = 20\%$。

停止准则 规定最大进化代数为 1000 代。

运算结果 此算法用 Python 实现，代码见附录 D 。

经过多次测试、运行，该程序表现良好，能够在若干代后收敛至较优的解。其中一次代表性的运算结果见于表 6 ，更多运算结果见附录 A 。

编号	运货路线
1	$9 \to 13 \to 33 \to 4 \to 29 \to 57 \to 37$
2	$61 \to 24 \to 19 \to 32 \to 49$
3	$2 \to 15 \to 50 \to 36$
4	$51 \to 17$
5	$16 \to 42 \to 53 \to 56 \to 55 \to 45 \to 23 \to 11 \to 30$
6	$20 \to 27 \to 2 \to 52 \to 28 \to 54$
7	$40 \to 59 \to 65 \to 30 \to 1 \to 5$
8	$41 \to 4 \to 7 \to 44$
9	$62 \to 3 \to 63 \to 22 \to 8 \to 35 \to 10 \to 60$
10	$46 \to 39 \to 34 \to 18 \to 47$
11	$22 \to 21 \to 26 \to 31 \to 58$
12	$22 \to 38 \to 64$
13	$12 \to 6$
14	$43 \to 14 \to 25 \to 48$
$T = \max T_j = 6.75$ h	

表 6: 问题一运算结果示例

6.2 问题二

与问题一相比，问题二加入了对时间窗口的考虑。记用户的预约时间为 $E(i)$，记货车到达客户所在的目的地的时间为 $T_j(i)$。

所谓时间窗口，就是要求货车到达客户所在的目的地的时间与用户预约的时间的差值

小于时间窗口 Δt 。即

$$0 \leq T_j(i) - E(i) \leq \Delta t \tag{6.7}$$
$$j = 1, 2, 3, \ldots, 14$$

车辆可以提前到达目的地，所以 $T_j(i) - E(i)$ 可以取负值。不过此时，需要货车进行等待。而当 $T_j(i) - E(i)$ 为正时，货车无需等待，可以直接安装。记货车 j 在每个目的地 $a_j(i)$ 等待的时间为 $w_j(i)$。

$$w_j(i) = \begin{cases} 0, & T_j(i) - E(i) \geq 0 \\ E(i) - T_j(i), & T_j(i) - E(i) < 0 \end{cases} \tag{6.8}$$

另外，需要指出的是，$w_j(i)$ 是完全由 $T_j(i)$ 决定的，因此，货车 j 到达目的地 $a_j(i)$ 的时间 $T_j(i)$ 可以在式 6.8 的基础上由递推公式 6.9 求出：

$$T_j(i+1) = T_j(i) + t_j(i, i+1) + \tau_j(i) + w_j(i) \tag{6.9}$$

由于 $i = 0$ 时对应的“目的地”实际上是出发点，所以规定

$$T_j(0) = 0 \tag{6.10}$$

只要路线已知，便可以根据根据式 6.8 、式 6.9 以及式 6.10 计算出到达目的地 $a_j(i+1)$ 的时间。

为解决问题二，一个常规的思路是在问题一的遗传算法的模型的基础上进行改进，然而对于时间窗口的限制条件而言，遗传操作较难做出保证杂交、变异得到的新个体满足时间窗口的算法设计。

另一种方法是在个体生成时就试图寻找合理的、满足时间窗口的解。然而在寻找这样的解的时候，难以保证解的质量较优。

所以我们将上述两种思路结合起来，首先找到一种能够有效构造合理解的方法；进而将这些合理解作为基础，使用有一定改变的遗传算法进行类似于“自交”的操作，从而得到时间更短的解。这种方法能够有效地避免遗传算法的杂交、变异等操作对可行解的破坏，较有效地解决了问题二。

6.2.1 合理解的构造

合理解的构造主要由以下步骤组成：

首先，将所有虚拟目的地 $\{a'\}$ 按照配送时间从早到晚排序，形成一个长度为 n' 的数列；

其次，对 14 辆货车，逐一检查能否在时间窗口中到达该目的地。货车到达目的地的时间由递推公式 6.9 求出；如果时间符合式 6.7 的要求，即能够按时到达时间窗口，该虚拟目的地便有可能加入到代表该货车的路线的数组 $\{a_j\}$。将该目的地随机分配给能够到达的货车中的一个，并重复这样的检查，直到所有的目的地都被分配完，于是得到了合理的解。

6.2.2 遗传算法与构造算法的结合

然而，难以通过上述算法得到高质量的解，因为在分配的时候，要考虑该目的地对整体的时间的影响，这是一件十分复杂的事情。所以，我们将该构造算法得到的解经过遗传算法进行优化，充分结合了两种算法的优点，同时避免了两种算法的缺点。

该遗传算法与问题一的遗传算法并无本质上的区别，主要对选择算子、杂交算子与变异算子进行了适当的改动。

基因编码 由于无须考虑不同个体间的杂交，对基因编码方案做出进一步的简化，直接使用 14 个整数串代表 14 辆货车经过的路线，而不再以 0 填充。[2]

选择算子 与第一题的选择算子类似，唯一的区别是在问题一的选择算子的基础上增加对时间窗口的检查。如果不符合时间窗口，则淘汰该个体，按照构造算法重新随机生成符合时间窗口条件的个体替换之。

实际上，由于变异产生的新个体也是按照构造算法生成的，所以其必然是符合时间窗口条件的，但这种检查增加了程序的鲁棒性。

自交变异算子 变异算子的设计结合了构造算法的部分内容。考虑到新群体中的每个个体都要满足时间窗口的条件，如果让不同个体进行杂交，会生成大量不符合时间窗口的个体，所以采取自交与变异的方法。主要内容如下：

1. 对于群体中每一个体，根据自交变异概率 p，随机确定该组是否变异；如果变异，则执行步骤 2- 4；
2. 随机选择两辆货车 j_1，j_2；
3. 将 j_1 与 j_2 到达的所有目的地 $\{A_{j_1}\}\cup\{A_{j_2}\}$（由编码方案，$\{A_{j_1}\}\cap\{A_{j_2}\}=\varnothing$）按照预约时间进行排序，形成整数串 D；
4. 按照构造算法，将该整数串重新分配给两辆货车，得到两个整数串 $\{A'_{j_1}\}$ 与 $\{A'_{j_2}\}$。

6.2.3 运算结果

遗传算法与构造算法的结合得到了很好的结果，其中一个典型的例子见表 7 ，更多例子见附录 B 。代码见附录 E 。

[2]为保证与问题一的统一以及代码的简洁性与通用性，在代码中还是保留了 0 的填充，即依旧采取问题一的编码方案。

编号	运货路线
1	$54 \to 44 \to 63 \to 9 \to 58$
2	$53 \to 38 \to 49 \to 3 \to 5 \to 12$
3	$36 \to 30 \to 13 \to 23 \to 11$
4	$30 \to 40 \to 22 \to 1 \to 2 \to 50$
5	$35 \to 34 \to 51 \to 8 \to 10$
6	$37 \to 33 \to 14 \to 62$
7	$45 \to 64 \to 41 \to 52 \to 19$
8	$56 \to 25 \to 57 \to 22$
9	$26 \to 31 \to 15 \to 4$
10	$24 \to 39 \to 32 \to 42 \to 16$
11	$4 \to 61 \to 17 \to 2$
12	$60 \to 21 \to 22 \to 7 \to 46 \to 27$
13	$65 \to 28 \to 43 \to 55 \to 20$
14	$47 \to 48 \to 59 \to 6 \to 29 \to 18$
$T = \max T_j = 10.82$ h	

表 7: 问题二运算结果示例

6.3 问题三

问题三在问题二的基础上加入了不确定的因素。所有不确定的因素最终会影响到配送的时间，甚至导致不符合时间窗口。而配送的时间的影响可以归为两个方面：对送货时间 $t_j(i)$ 的影响以及对安装时间 $\tau_j(i)$ 的影响。记送货时间的不确定为 $\delta t_j(a_j(i), a_j(i+1))$，记安装时间的不确定为 $\delta\tau_j(a_j(i), a_j(i+1))$。故求时间 $T_j(i)$ 的递推公式 6.9 可以表达为

$$\begin{aligned} T_j(i+1) = T_j(i) & \\ & + t_j\big(a_j(i), a_j(i+1)\big) + \delta t_j\big(a_j(i), a_j(i+1)\big) \\ & + \tau_i\big(a_j(i)\big) + \delta\tau_i\big(a_j(i)\big) \\ & + w_j(i) \end{aligned} \tag{6.11}$$

若分别记

$$\lambda_j(i) = \frac{\delta t_j\big(a_j(i), a_j(i+1)\big)}{t_j\big(a_j(i), a_j(i+1)\big)}$$

$$\mu_j(i) = \frac{\delta\tau_i\big(a_j(i)\big)}{\tau_i\big(a_j(i)\big)}$$

则式 6.11 可以表示为

$$\begin{aligned} T_j(i+1) = T_j(i) \\ &+ (1+\lambda_j(i)) \cdot t_j\big(a_j(i), a_j(i+1)\big) \\ &+ (1+\mu_j(i)) \cdot \tau_i\big(a_j(i)\big) \\ &+ w_j(i) \end{aligned} \tag{6.12}$$

$w(j)$ 依旧由式 6.8 求出。

以此为新的递归公式，随机指定不确定 $\lambda_j(i)$ 与 $\mu_j(i)$，便可以得到货车到达目的地的时间 T_j。

对于路上时间的不确定度 $\lambda_j(i)$，影响模型稳定性的主要因素会导致时间变长，一个等效的做法是，将目的地间的路程加长。即：根据原路程的数据安排路线，并随机增大一些路的长度[3]，模拟堵车的情况。而对于安装时间的不确定 $\mu_j(i)$，可以通过增加安装的时间来进行分析。将以上一部分进行合并，同时对路上时间与安装时间的不确定度进行分析：

1. 根据原始数据规划合理路线；
2. 加长部分目的地间的长度；
3. 加长部分安装时间的长度；
4. 计算各辆货车在新的长度的路上运行的情况，判断能否在时间窗口内到达目的地。

稳定性分析结果表明，我们建立的模型对问题一、问题二均表现良好，在绝大部分情况下都能够顺利完成配送目标。

7 模型的评价

本文根据遗传算法建立的模型成功地解决了三个问题，并且具有很大的灵活性，可以根据实际问题方便地做出修改，能解决很多类似问题，如快递员送货、路线规划、人事安排等。

问题一 根据优化的遗传算法，取得了很好的、满足条件的解。唯一的不足是均衡性一般，即不同货车配送的目的地数差别较大，但一般配送目的地少的货车配送的目的地距离起点较远，时间上仍较为均衡，故完全可以使用。

问题二 我们设计的构造可行解算法表现优良，能高效地随机构造满足时间窗口的可行解。但虽然理论上能够取到解空间内所有可行解，但因变异算法不够优秀，存在过早收敛的状况，而导致无法跳出局部最优解，实际情况表现为 100 代之后的结果与 1000 代类似，过早收敛较为明显，表明该变异算法还有一定提升空间。

[3] 代码见附录 F

问题三 经检验，我们的模型稳定性较好，个别路径及安装时间出现变化时，不会对最终结果造成太大影响。但堵车等情况发生时，仍然存在较小的可能性，使货车无法在时间窗口内到达目的地，所以还存在改进的空间。

A 问题一的运算结果示例

编号	运货路线
1	$19 \to 45 \to 25 \to 16 \to 35 \to 33 \to 4 \to 52$
2	$2 \to 22 \to 61 \to 2$
3	$58 \to 62 \to 30 \to 65 \to 40$
4	$54 \to 43 \to 27 \to 12 \to 6 \to 7 \to 49 \to 41$
5	$36 \to 3 \to 11 \to 21 \to 32$
6	$48 \to 22 \to 38 \to 42 \to 59 \to 20$
7	$22 \to 5 \to 60$
8	$28 \to 53 \to 55$
9	$37 \to 8 \to 34 \to 31$
10	$26 \to 57 \to 1 \to 23 \to 47$
11	$14 \to 13 \to 4 \to 10 \to 51 \to 44$
12	$46 \to 24 \to 15 \to 29 \to 9 \to 18$
13	$50 \to 30 \to 63$
14	$64 \to 39 \to 17 \to 56$
$T = \max T_j = 6.83$ h	

编号	运货路线
1	$22 \to 43 \to 25 \to 16 \to 11 \to 6$
2	$49 \to 22 \to 50 \to 37 \to 32 \to 22$
3	$46 \to 39 \to 53 \to 63 \to 58 \to 36$
4	$28 \to 62 \to 42 \to 9 \to 56 \to 8$
5	$41 \to 30 \to 26$
6	$52 \to 30 \to 34 \to 10$
7	$55 \to 31 \to 21 \to 57 \to 60 \to 64 \to 18$
8	$2 \to 3 \to 35 \to 17$
9	$15 \to 12 \to 4 \to 13 \to 24 \to 29 \to 19$
10	$1 \to 14 \to 47 \to 5 \to 59 \to 38$
11	$7 \to 45 \to 33 \to 44$
12	61
13	$23 \to 48 \to 2 \to 4 \to 65$
14	$27 \to 54 \to 20 \to 40 \to 51$
$T = \max T_j = 7.01$ h	

编号	运货路线
1	$4 \to 61 \to 22 \to 8 \to 14 \to 53$
2	$22 \to 44 \to 26 \to 46 \to 45$
3	$58 \to 31 \to 65 \to 42$
4	$2 \to 39 \to 22 \to 18 \to 47 \to 57 \to 11 \to 13 \to 30 \to 63$
5	$48 \to 55 \to 41 \to 20 \to 6 \to 28 \to 10 \to 1 \to 52$
6	$33 \to 62 \to 38 \to 24 \to 30 \to 50$
7	$29 \to 16$
8	$19 \to 64 \to 3 \to 54$
9	$35 \to 2 \to 51$
10	$4 \to 7 \to 37 \to 23 \to 60$
11	$17 \to 40 \to 12 \to 21 \to 32 \to 43$
12	$9 \to 56 \to 49 \to 5 \to 25$
13	$34 \to 36 \to 59$
14	$15 \to 27$
$T = \max T_j = 7.11$ h	

编号	运货路线
1	$59 \to 62 \to 8$
2	$20 \to 22 \to 19 \to 4$
3	$27 \to 24 \to 35 \to 52 \to 5 \to 9$
4	$3 \to 22$
5	$21 \to 53 \to 57 \to 34 \to 40 \to 11 \to 6 \to 30$
6	$2 \to 18 \to 41 \to 63 \to 33 \to 56$
7	$51 \to 46 \to 50 \to 26 \to 60 \to 58 \to 45 \to 13$
8	$17 \to 16 \to 15 \to 55 \to 23$
9	$36 \to 54 \to 37 \to 44 \to 61 \to 65 \to 43$
10	$7 \to 12 \to 22 \to 38 \to 31 \to 49$
11	$42 \to 14 \to 48 \to 10 \to 47 \to 29$
12	$30 \to 39 \to 1 \to 4 \to 28$
13	$32 \to 64 \to 2$
14	25
$T = \max T_j = 6.96$ h	

B　问题二的运算结果示例

编号	运货路线
1	$36 \to 34 \to 1 \to 5$
2	$45 \to 30 \to 14 \to 8 \to 10$
3	$37 \to 25 \to 16 \to 22$
4	$35 \to 61 \to 51 \to 9$
5	$53 \to 38 \to 32 \to 7 \to 29$
6	$4 \to 59 \to 57 \to 23$
7	$30 \to 33 \to 42 \to 43 \to 2 \to 50$
8	$60 \to 39 \to 44 \to 6 \to 2$
9	$65 \to 28 \to 15 \to 4 \to 58$
10	$26 \to 31 \to 13 \to 17 \to 55 \to 20$
11	$47 \to 48 \to 49 \to 63 \to 12$
12	$24 \to 40 \to 22 \to 3 \to 11$
13	$54 \to 64 \to 41 \to 52 \to 19 \to 27$
14	$56 \to 21 \to 22 \to 62 \to 46 \to 18$
$T = \max T_j = 10.96$ h	

编号	运货路线
1	$56 \to 21 \to 22 \to 63 \to 12$
2	$30 \to 44 \to 57 \to 43 \to 10 \to 50$
3	$35 \to 61 \to 42 \to 16 \to 46$
4	$53 \to 25 \to 52 \to 22 \to 58$
5	$65 \to 31 \to 6 \to 23 \to 55 \to 20$
6	$37 \to 28 \to 17$
7	$60 \to 40 \to 22 \to 1 \to 5$
8	$4 \to 39 \to 49 \to 3 \to 4$
9	$54 \to 34 \to 51 \to 62 \to 9 \to 27$
10	$45 \to 30 \to 14 \to 8 \to 2$
11	$26 \to 38 \to 32 \to 7 \to 11$
12	$36 \to 33 \to 13 \to 19 \to 18$
13	$47 \to 48 \to 59 \to 29$
14	$24 \to 64 \to 41 \to 15 \to 2$
$T = \max T_j = 11.01$ h	

编号	运货路线
1	$47 \to 34 \to 15 \to 9$
2	$37 \to 59 \to 7 \to 4 \to 58$
3	$26 \to 64 \to 41 \to 23$
4	$54 \to 49 \to 52 \to 19$
5	$65 \to 32 \to 42 \to 3 \to 10$
6	$45 \to 28 \to 43$
7	$36 \to 33 \to 14 \to 16 \to 5 \to 18$
8	$35 \to 61 \to 51 \to 2$
9	$30 \to 30 \to 57 \to 29 \to 50$
10	$56 \to 48 \to 25 \to 6 \to 62 \to 12$
11	$53 \to 38 \to 44 \to 63 \to 22 \to 11$
12	$24 \to 21 \to 22 \to 8 \to 46 \to 27$
13	$60 \to 39 \to 31 \to 13 \to 17 \to 55 \to 20$
14	$4 \to 40 \to 22 \to 1 \to 2$
$T = \max T_j = 10.97$ h	

编号	运货路线
1	$45 \to 21 \to 22 \to 63$
2	$53 \to 34 \to 57 \to 43$
3	$47 \to 64 \to 41 \to 1 \to 4$
4	$54 \to 44 \to 6 \to 62 \to 9 \to 27$
5	$37 \to 30 \to 51$
6	$30 \to 33 \to 13 \to 3 \to 55 \to 50$
7	$56 \to 39 \to 28 \to 15 \to 22 \to 12$
8	$35 \to 61 \to 42 \to 8 \to 2$
9	$36 \to 25 \to 17 \to 46 \to 18$
10	$65 \to 31 \to 52 \to 29 \to 10 \to 20$
11	$4 \to 59 \to 23 \to 58$
12	$60 \to 40 \to 22 \to 16 \to 2 \to 11$
13	$24 \to 48 \to 49 \to 7 \to 5$
14	$26 \to 38 \to 32 \to 14 \to 19$
$T = \max T_j = 11.01$ h	

C 最短路模型的代码

```python
#!/usr/bin/env python3
'''
Floyd algorithm: shortest path between each two nodes
input from infile, and output result to outfile
2017/12/16
'''
infile = input('infile:')
outfile = input('outfile:')
num = int(input('city num:'))
dist = [[0 for col in range(num)] for row in range(num)]
fin = open(infile, "r")
fout = open(outfile, "w")
for i in range(num):
    data = fin.readline()
    dist[i] = [float(j) for j in data.split()]
for i in range(num):
    for j in range(num):
        if i < j:
            dist[j][i] = dist[i][j]
for k in range(num):
    for i in range(num):
        for j in range(num):
            if dist[i][k] + dist[k][j] < dist[i][j]:
                dist[i][j] = dist[i][k] + dist[k][j]
for i in dist:
    for j in i:
        fout.write(str(round(j, 3)) + ' ')
    fout.write('\n')
fin.close()
fout.close()
```

D 问题一遗传算法代码

```python
#!/usr/bin/env python3
# -*- coding: UTF-8 -*-
'''
Math modeling
genetic algorithms solving the MTSP question in Question 1
Yimin Gu
'''
from random import *
from math import *
import sys
import copy

realcity = 66   #66 real cities(but I use singular form to have a difference)
cities = 71     #all 71 cities, including 5 virtual cities
travelers = 14  #all 14 travelers
T = 1000        #generations
N = 100        #population

pc = 1.0        #crossover probability
pm = 0.4        #mutation probability

#mapping from cities(including virtual cities) to real city
cities2realcity = [0 for i in range(cities)]

#distances from one real city to another, read from floyd.py
dist = [[0 for col in range(realcity)] for row in range(realcity)]

#weight of each delivering task
weight = [0 for i in range(cities)]
#max weight of each car
maxweight = 14.0

```

```
speed = 60
#installation time of each city, in question 1, all is same.
#I'll take the average value, 1.1 hours, and exchange time to distance
#with speed 60km/h
#speed will be multiplied when initialization
installtimelength = [1.1 for i in range(cities)]
#the start point 0 don't need install time
installtimelength[0] = 0

class individual:
    #init with a random route for each indiviudal
    def __init__(self):
        #fitness of each individual, smaller is better
        #fitness is the biggest length of the 14 travelers
        #fitness2 is total length
        self.fitness = 999999
        self.fitness2 = 999999
        #the chromosomes -- S
        #each individual from the population:
        #each traveler -- S_i: S_1 to S_travelers
        #   m-bit non-negative integer number(m means cities)
        #       S_i_1 to S_i_m
        #   each digit not larger than m
        #S_i_j = k:traveler i goto city k in the j-th step globally
        #S_i_j = 0:traveler i don't move in the j-th step globally
        #note that S_*_j only has only one non-zero value while *
        #goes from 1 to traveler
        self.chrm = [[0 for i in range(cities)] for j in range(travelers)]
        t = [x for x in range(cities)]
        shuffle(t)
        for i in range(cities):
```

```
64              self.chrm[randint(0, travelers - 1)][i] = t[i]
65
66  M = 25
67  #M best individual in-history to make selection better
68  bestindi = [individual() for i in range(M)]
69
70  #global minimum distance
71  mindistancenow = 999999.99
72  mindistancetotalnow = 999999.99
73
74  mindistancehistory = 999999.99
75  mindistancetotalhistory = 999999.99
76  minindividualhistory = individual()
77
78  #the population and popnew for temp storage
79  population = [individual() for i in range(N)]
80  popnew = [individual() for i in range(N)]
81
82  def initialize():
83      if len(sys.argv) == 2:
84          if sys.argv[1] != "-q":
85              print("initialize...")
86      #read destdata.txt to get
87      global weight
88      global cities2realcity
89      fin1 = open("destdata.txt", "r")
90      for i in range(cities):
91          (cities2realcity[i], nouse, weight[i]) = fin1.readline
                ().split()
92      for i in range(cities):
93          cities2realcity[i] = int(cities2realcity[i])
94          weight[i] = float(weight[i])
95          installtimelength[i] = speed * (float(
                installtimelength[i]))
96      fin1.close()
97      #read data2.txt containing shortest path generated by
```

```
       floyd.py
98      global dist
99      fin = open("data2.txt", "r")
100     for i in range(realcity):
101         data = fin.readline()
102         dist[i] = [float(j) for j in data.split()]
103     fin.close()
104
105 #calculate the fitness of each individual #2, improved
106 def calc_fitness2():
107     global mindistancenow
108     global mindistancetotalnow
109     global mindistancetotalhistory
110     global mindistancehistory
111     global minindividualhistory
112     if len(sys.argv) == 2:
113         if sys.argv[1] != "-q":
114             print("calulate fitness #2...")
115     mindistancenow = 999999
116     for i in range(N):
117         mindistancetotalnow = 0
118         mindistancenowt = -1
119         for j in range(travelers):
120             length = 0
121             weightnow = 0
122             #start from city 0
123             citynow = 0
124             for k in range(cities):
125                 citynext = population[i].chrm[j][k]
126                 if citynext != 0:
127                     length += dist[cities2realcity[citynow]][
                           cities2realcity[citynext]]
128                     mindistancetotalnow += dist[
                           cities2realcity[citynow]][
                           cities2realcity[citynext]]
129                     citynow = citynext
```

```
130                     #"mindistancenowt" at-install
131                     length += installtimelength[citynow]
132                     mindistancetotalnow += installtimelength[
                            citynow]
133                     weightnow += weight[citynow]
134             #overload?
135             if weightnow > maxweight:
136                 #just replace the individual and calc again
137                 #until all is OK
138                 #to get better result
139                 population[i] = individual()
140                 j -= 1
141                 continue
142             #go back to city 0
143             length += dist[cities2realcity[citynow]][0]
144             mindistancetotalnow += dist[cities2realcity[
                    citynow]][0]
145             mindistancenowt = max(length, mindistancenowt)
146         population[i].fitness = mindistancenowt
147         population[i].fitness2 = mindistancetotalnow
148         mindistancenow = min(mindistancenow, mindistancenowt)
149         #update history
150         if mindistancenow < mindistancehistory:
151             mindistancehistory = mindistancenow
152             minindividualhistory = copy.deepcopy(population[i
                    ])
153         mindistancetotalhistory = min(mindistancetotalhistory,
                mindistancetotalnow)
154
155 #improved tournament selection
156 #to keep the best individuals in history always be used
157 def select2_2():
158     if sys.argv == 2:
159         if sys.argv[1] != "-q":
160             print("select...")
161     global population
```

```
162     global popnew
163     global bestindi
164     population = sorted(population, key = lambda x:(x.fitness,
            x.fitness2))
165     poptmp = [x for x in bestindi + population[:M]]
166     poptmp = sorted(poptmp, key = lambda x:(x.fitness, x.
           fitness2))
167     bestindi = [copy.deepcopy(x) for x in poptmp[:M]]
168     for i in range(N):
169         popt = sorted([population[randint(0, N - 1)],
               population[randint(0, N - 1)]], key = lambda x:(x.
               fitness, x.fitness2))
170         popnew[i] = popt[0]
171     population = popnew
172     population = sorted(population, key = lambda x:(x.fitness,
            x.fitness2))
173     population = population[:-M] + [copy.deepcopy(x) for x in
           bestindi]
174
175 #input a individual's chromosome
176 #return striped(without zeros, a one-dimension array) chrm
177 def strip(chrm):
178     return [sum(chrm[j][i] for j in range(travelers)) for i in
            range(cities)]
179 #input a striped chromosome and the individual's original
       chromosome
180 #return expanded chrm
181 def expand(schrm, chrm):
182     schrm = [x for x in schrm if x]
183     flag = 0
184     for i in range(cities):
185         for j in range(travelers):
186             if chrm[j][i] != 0:
187                 t = schrm[flag]
188                 chrm[j][i] = t
189                 flag += 1
```

```
                break
    return chrm
def crossover():
    #order crossover (OX)
    #choose 2 random crossover points
    #exchange crossover parts
    #keep relative sequence of cities
    #a1 = 0 1 2 3 4 5 6 7 8 9
    #a2 = 9 8 7 6 5 4 3 2 1 0
    #OX points: ^1  ^2
    #b1 = 0 1 2|6 5 4|6 7 8 9
    #b2 = 9 8 7|3 4 5|6 7 8 9
    #a1 from point 2:6 7 8 9 0 1 2 3 4 5
    #remove 6 5 4:7 8 9 0 1 2 3
    #refill from point 2:1 2 3|6 5 4|7 8 9 0
    if len(sys.argv) == 2:
        if sys.argv[1] != "-q":
            print("crossover...")
    #pair randomly using shuffled array
    h = [i for i in range(N)]
    shuffle(h)
    for i in range(int((N - 1) / 2)):
        if random() < pc:
            #crossover i and N-i-1
            #strip
            a1 = strip(population[h[i]].chrm)
            a2 = strip(population[h[N - i - 1]].chrm)
            pos1, pos2 = sorted((randint(0, cities - 1),
               randint(0, cities - 1)))
            length = pos2 - pos1 + 1
            ta = a1[pos1:pos2 + 1]
            tb = a2[pos1:pos2 + 1]
            #remove
            t1 = [x for x in (a1[pos2 + 1:] + a1[:pos2 + 1])
               if not x in tb]
            t2 = [x for x in (a2[pos2 + 1:] + a2[:pos2 + 1])
```

```
                if not x in ta]
            #refill
            afinal = t1[pos1:] + tb + t1[:pos1]
            bfinal = t2[pos1:] + ta + t2[:pos1]
            #expand
            population[h[i]].chrm = expand(afinal, population[
                h[i]].chrm)
            population[h[N - i - 1]].chrm = expand(bfinal,
                population[h[N - i - 1]].chrm)

def mutation():
    if len(sys.argv) == 2:
        if sys.argv[1] != "-q":
            print("mutation...")
    for i in range(N):
        #two kind of mutation, each probability is 0.5 * pm
        rand = random()
        if rand < pm * 0.5:
            #1. reverse a "block" of chrm between ranint1 and
                ranint2
            #  ----A+++++c----    ----c+++++A----
            #  ----B+++++e----    ----e+++++B----
            #  ----C+++++d----    ----e+++++C----
            #  ----++++++++---- => ----++++++++----
            #  ----++++++++----    ----++++++++----
            #  ----++++++++----    ----++++++++----
            #  ----E+++++a----    ----a+++++E----
            #t is population[i].chrm[j] with zeros striped
            pnt1, pnt2 = sorted((randint(0, cities - 1),
                randint(1, cities - 1)))
            for j in range(travelers):
                population[i].chrm[j] = population[i].chrm[j
                    ][:pnt1] + population[i].chrm[j][pnt1:pnt2 +
                     1][::-1] + population[i].chrm[j][pnt2 + 1:]
        rand = random()
        if rand < pm * 0.5:
```

```
253             #2. exchange.
254             #randomly find s_i_k = 0, s_j_l != 0, and swap
                   them (showed in <>)
255             #exist s_i'_k !=0, i' in {1 to n} \ {i}
256             #random j' in {1 to n}
257             #swap s_i'_k and s_j'_l (showed in ())
258             #n equals travelers
259             #       l      k            l      k
260             #j'----(0)-----0--    ----(X)-----0--
261             #j ----<A>-----0--    ----<0>-----0--
262             #  -----0------0--    -----0------0--
263             #  -----0------0-- => -----0------0--
264             #i'-----0-----(X)-    -----0-----(0)-
265             #i -----0-----<0>-    -----0-----<A>-
266             #  -----0------0--    -----0------0--
267             #with prefix s to avoid conflict with loop
                   variables
268             #count the num of 0 and non-0, for random
269             cnt0 = 0; cntnon0 = 0
270             for j in range(travelers):
271                 for k in range(cities):
272                     if population[i].chrm[j][k] == 0:
273                         cnt0 += 1
274                     else:
275                         cntnon0 += 1
276             rand0 = 0; randnon0 = 0
277             #these ==0 cases won't happen normally
278             if cnt0 != 0:
279                 rand0 = randint(1, cnt0)
280             if cntnon0!= 0:
281                 randnon0 = randint(1, cntnon0)
282             cnt0 = 0; cntnon0 = 0
283             for j in range(travelers):
284                 for k in range(cities):
285                     if population[i].chrm[j][k] == 0:
286                         cnt0 += 1
```

```
287                     else:
288                         cntnon0 += 1
289                     #now judge
290                     if cnt0 == rand0:
291                         si = j
292                         sk = k
293                         #a stupid way to make sure si sk not
                               be changed again
294                         cnt0 = 999999
295                     if cntnon0 == randnon0:
296                         sj = j
297                         sl = k
298                         cntnon0 = 999999
299             #find i'
300             sii = 0
301             for j in range(travelers):
302                 if population[i].chrm[j][sk] != 0:
303                     sii = j
304                     break
305             #random j'
306             sjj = randint(0, travelers - 1)
307             #swap #1
308             population[i].chrm[si][sk], population[i].chrm[sj
                   ][sl] = \
309                 (population[i].chrm[sj][sl], population[i].
                       chrm[si][sk])
310             #swap #2
311             population[i].chrm[sii][sk], population[i].chrm[
                   sjj][sl] = \
312                 (population[i].chrm[sjj][sl], population[i].
                       chrm[sii][sk])
313
314 def print_result():
315     maxi = 0
316     for i in range(N):
317         if population[i].fitness < population[maxi].fitness:
```

```
318             maxi = i
319     print("Min distance in history:", round(mindistancehistory
           , 5))
320     print("Min distance total in history:", round(
           mindistancetotalhistory, 5))
321     for i in range(travelers):
322         print("traveler ", i, ":")
323         print([cities2realcity[minindividualhistory.chrm[i][x
               ]] for x in range(cities) if minindividualhistory.
               chrm[i][x]])
324     fout = open("ans.txt", "a")
325     fout.write('-----')
326     fout.write('\n')
327     fout.write("Min distance in history:" + str(round(
           mindistancehistory, 5)))
328     fout.write('\n')
329     fout.write("Min distance total in history:" + (str(round(
           mindistancetotalhistory, 5))))
330     fout.write('\n')
331     for i in range(travelers):
332         fout.write("traveler " + str(i))
333         fout.write('\n')
334         fout.write(str([cities2realcity[minindividualhistory.
               chrm[i][x]] for x in range(cities) if
               minindividualhistory.chrm[i][x]]))
335         fout.write('\n')
336     fout.write('-----')
337     fout.write('\n\n')
338 def printmsg():
339     if len(sys.argv) == 2:
340         if sys.argv[1] == "-q":
341             print('-->', str(round(min([x.fitness for x in
                   population]), 5)))
342     else:
343         print("Generation " + str(t))
344         print("Min distance: " + str(round(mindistancenow, 5))
```

```
            )
345         print("Min mindistance total:" + str(round(
               mindistancetotalnow, 5)))
346 #begin of main
347 initialize()
348 #main loop
349 calc_fitness2()
350 for t in range(T):
351     printmsg()
352     crossover()
353     mutation()
354     calc_fitness2()
355     select2_2()
356 print_result()
357 print("end.")
```

E 问题二的代码

```
#!/usr/bin/env python3
# -*- coding: UTF-8 -*-
'''
Math modeling
genetic algorithms solving the MTSP question with time window in Question 2
Yimin Gu
'''
from random import *
from math import *
import sys
import copy

realcity = 66   #66 real cities(but I use singular form to have a difference)
cities = 71     #all 71 cities, including 5 virtual cities
travelers = 14  #all 14 travelers
T = 10        #generations
N = 100         #population

pc = 0.2        #crossover probability
pm = 0.4        #mutation probability

#mapping from cities(including virtual cities) to real city
cities2realcity = [0 for i in range(cities)]

#distances from one real city to another, read from floyd.py
dist = [[0 for col in range(realcity)] for row in range(realcity)]

#weight of each delivering task
weight = [0 for i in range(cities)]
#max weight of each car
maxweight = 14.0
```

```
32
33 speed = 60
34 #installation time of each city, in question 2, all is same.
35 #I'll take the average value, 1.1 hours, and exchange time to
      distance
36 #with speed 60km/h
37 #speed will be multiplied when initialization
38 installtimelength = [1.1 for i in range(cities)]
39 #the start point 0 don't need install time
40 installtimelength[0] = 0
41 #the time window
42 timewindow = [0 for i in range(cities)]
43
44 class individual:
45     #init with a random route for each individual
46     def __init__(self):
47         #fitness of each individual, smaller is better
48         #fitness is the biggest length of the 14 travelers
49         #fitness2 is total length
50         self.fitness = 999999
51         self.fitness2 = 999999
52         #0 means non timeout, 1 means timed-out
53         self.istimeout = 1
54         #departure time of each travelers in individual
55         self.timedep = [0 for i in range(travelers)]
56         #the chromosomes -- S
57         #each individual from the population:
58         #each traveler -- S_i: S_1 to S_travelers
59         #   m-bit non-negative integer number(m means cities)
60         #       S_i_1 to S_i_m
61         #   each digit not larger than m
62         #S_i_j = k:traveler i goto city k in the j-th step
              globally
63         #S_i_j = 0:traveler i don't move in the j-th step
              globally
64         #note that S_*_j only has only one non-zero value
```

```
            while *
        #goes from 1 to traveler
        self.chrm = [[0 for i in range(cities)] for j in range
            (travelers)]
        #NOTE: chrm will be inited in initialization function

M = 25
#M best individual in-history to make selection better
bestindi = [individual() for i in range(M)]

#global minimum distance
mindistancenow = 999999.99
mindistancetotalnow = 999999.99

mindistancehistory = 999999.99
mindistancetotalhistory = 999999.99
minindividualhistory = individual()

#the population and popnew for temp storage
population = [individual() for i in range(N)]
popnew = [individual() for i in range(N)]

#this cannot make sure ok, but some kind of reasonable
def initindi(indi):
    t = [x for x in range(1, cities)]
    t = sorted(t, key = lambda x:timewindow[x])
    # print([timewindow[x] for x in t])
    for j in range(cities - 1):
        oklist = []
        for k in range(travelers):
            if indi.chrm[k][0] == 0:
                oklist.append(k)
            else:
                l = 0
                while indi.chrm[k][l]:
                    l += 1
```

```
99                  l -= 1
100                 if timewindow[indi.chrm[k][l]] + 0.5 +
                       installtimelength[indi.chrm[k][l]] / speed +
                        \
101                         dist[cities2realcity[indi.chrm[k][l
                               ]]][cities2realcity[t[j]]] / speed
                               <= timewindow[t[j]] + 0.5:
102                     oklist.append(k)
103         #failed, again
104         if oklist == []:
105             indi.chrm = [[0 for n in range(cities)] for m in
                   range(travelers)]
106             return initindi(indi)
107         pos = sample(oklist, 1)
108         l = 0
109         if indi.chrm[pos[0]][0] == 0:
110             l = -1
111         else:
112             while indi.chrm[pos[0]][l]:
113                 l += 1
114             l -= 1
115         indi.chrm[pos[0]][l + 1] = t[j]
116     return indi
117
118 def initialize():
119     if len(sys.argv) == 2:
120         if sys.argv[1] != "-q":
121             print("initialize...")
122     #read destdata.txt to get
123     global weight
124     global cities2realcity
125     global timewindow
126     fin1 = open("destdata.txt", "r")
127     for i in range(cities):
128         (cities2realcity[i], nouse, weight[i], timewindow[i])
              = fin1.readline().split()
```

```
129     for i in range(cities):
130         cities2realcity[i] = int(cities2realcity[i])
131         weight[i] = float(weight[i])
132         installtimelength[i] = speed * (float(
                installtimelength[i]))
133         hour, minute = timewindow[i].split(':')
134         timewindow[i] = round(int(hour) + float(int(minute)
                /60), 3)
135     fin1.close()
136     #read data2.txt containing shortest path generated by
            floyd.py
137     global dist
138     fin = open("data2.txt", "r")
139     for i in range(realcity):
140         data = fin.readline()
141         dist[i] = [float(j) for j in data.split()]
142     fin.close()
143     #now init chrms in in population
144     #a inited individual will be in time limit
145     for i in range(N):
146         population[i] = initindi(population[i])
147 
148 #calculate the fitness of each individual #2, improved
149 def calc_fitness2():
150     global mindistancenow
151     global mindistancetotalnow
152     global mindistancetotalhistory
153     global mindistancehistory
154     global minindividualhistory
155     if len(sys.argv) == 2:
156         if sys.argv[1] != "-q":
157             print("calculate fitness #2...")
158     mindistancenow = 999999
159     for i in range(N):
160         population[i].istimeout = 0
161         mindistancetotalnow = 0
```

```
162         mindistancenowt = -1
163         for j in range(travelers):
164             length = 0
165             weightnow = 0
166             time = 0
167             #start from city 0
168             citynow = 0
169             for k in range(cities):
170                 citynext = population[i].chrm[j][k]
171                 if citynext != 0:
172                     #the first city's time tells when to
                            departure
173                     if time == 0:
174                         time = population[i].timedep[j] = \
175                                 timewindow[citynext] - dist[
                                    cities2realcity[citynow]][
                                    cities2realcity[citynext]] /
                                     speed
176                     time += dist[cities2realcity[citynow]][
                            cities2realcity[citynext]] / speed
177                     #have to wait
178                     if time < timewindow[citynext]:
179                         length += (timewindow[citynext] - time
                                ) * speed
180                         time = timewindow[citynext]
181                     #late, kill it
182                     elif time > timewindow[citynext] + 0.5:
183                         #to make it a dead individual
184                         weightnow = 999
185                         break
186                     else:
187                         pass
188                     length += dist[cities2realcity[citynow]][
                            cities2realcity[citynext]]
189                     mindistancetotalnow += dist[
                            cities2realcity[citynow]][
```

```
                        cities2realcity[citynext]]
                    citynow = citynext
                    #"mindistancenow" at-install
                    length += installtimelength[citynow]
                    mindistancetotalnow += installtimelength[
                        citynow]
                    time += installtimelength[citynow] / speed
                    weightnow += weight[citynow]
            #overload?
            if weightnow > maxweight:
                #just replace the individual and calc again
                #until all is OK
                #to get better result
                population[i] = initindi(individual())
                j -= 1
                continue;
            #go back to city 0
            length += dist[cities2realcity[citynow]][0]
            mindistancetotalnow += dist[cities2realcity[
                citynow]][0]
            mindistancenowt = max(length, mindistancenowt)
        mindistancenow = min(mindistancenow, mindistancenowt)
        population[i].fitness = mindistancenowt
        population[i].fitness2 = mindistancetotalnow
        #update history
        if population[i].istimeout == 0 and mindistancenow <
            mindistancehistory:
            mindistancehistory = mindistancenow
            minindividualhistory = copy.deepcopy(population[i
                ])
        mindistancetotalhistory = min(mindistancetotalhistory,
            mindistancetotalnow)

#improved tournament selection
#to keep the best individuals in history always be used
def select2_2():
```

```
220     if sys.argv == 2:
221         if sys.argv[1] != "-q":
222             print("select...")
223     global population
224     global popnew
225     global bestindi
226     #non-time-out individual is best
227     population = sorted(population, key = lambda x:(x.
           istimeout, x.fitness, x.fitness2))
228     poptmp = [x for x in bestindi + population[:M]]
229     poptmp = sorted(poptmp, key = lambda x:(x.istimeout, x.
           fitness, x.fitness2))
230     bestindi = [copy.deepcopy(x) for x in poptmp[:M]]
231     print([int(x.fitness) for x in bestindi])
232     print([x.istimeout for x in bestindi])
233     for i in range(N):
234         popt = sorted([population[randint(0, N - 1)],
              population[randint(0, N - 1)]], key = lambda x:(x.
              istimeout, x.fitness, x.fitness2))
235         popnew[i] = popt[0]
236     population = sorted(population, key = lambda x:(x.
           istimeout, x.fitness, x.fitness2))
237     population = popnew
238     population = population[:-M] + [copy.deepcopy(x) for x in
           bestindi]
239
240 def crossover():
241     #NEW crossover (may be a bad one) written by me
242     if len(sys.argv) == 2:
243         if sys.argv[1] != "-q":
244             print("crossover...")
245     for i in range(N):
246         h = [j for j in range(travelers)]
247         shuffle(h)
248         for j in range(int((travelers) / 2) - 1):
249             if random() < pc:
```

```
250                 bak1 = copy.deepcopy(population[i].chrm[h[j]])
251                 bak2 = copy.deepcopy(population[i].chrm[h[
                        travelers - j - 1]])
252                 citypool = [x for x in population[i].chrm[h[j
                        ]] + population[i].chrm[h[travelers - j -
                        1]] if x]
253                 citypool = sorted(citypool, key = lambda x:
                        timewindow[x])
254                 population[i].chrm[h[j]] = [0 for x in range(
                        cities)]
255                 population[i].chrm[h[travelers - j - 1]] = [0
                        for x in range(cities)]
256                 for k in range(len(citypool)):
257                     oklist = []
258                     for m in (h[j], h[travelers - j - 1]):
259                         if population[i].chrm[m][0] == 0:
260                             oklist.append(m)
261                         else:
262                             l = 0
263                             while population[i].chrm[m][l]:
264                                 l += 1
265                             l -= 1
266                             if timewindow[population[i].chrm[m
                                ][l]] + 0.5 + installtimelength[
                                population[i].chrm[m][l]] /
                                speed + dist[cities2realcity[
                                population[i].chrm[m][l]]][
                                cities2realcity[citypool[k]]] /
                                speed <= timewindow[citypool[k]]
                                 + 0.5:
267                                 oklist.append(m)
268                     #failed, reset
269                     if oklist == []:
270                         population[i].chrm[h[j]] = bak1
271                         population[i].chrm[h[travelers - j -
                                1]] = bak2
```

```
                        break
                    pos = sample(oklist, 1)
                    l = 0
                    if population[i].chrm[pos[0]][0] == 0:
                        l = -1
                    else:
                        while population[i].chrm[pos[0]][l]:
                            l += 1
                        l -= 1
                    population[i].chrm[pos[0]][l + 1] =
                       citypool[k]

def print_result():
    maxi = 0
    for i in range(N):
        if population[i].fitness < population[maxi].fitness:
            maxi = i
    print("Min distance in history:", round(mindistancehistory
       , 5))
    print("Min distance total in history:", round(
       mindistancetotalhistory, 5), "time out:",
       minindividualhistory.istimeout)
    for i in range(travelers):
        print("traveler ", i, ":")
        print([cities2realcity[minindividualhistory.chrm[i][x
           ]] for x in range(cities) if minindividualhistory.
           chrm[i][x]])
    fout = open("ans.txt", "a")
    fout.write('-----')
    fout.write('\n')
    fout.write("Min distance in history:" + str(round(
       minindividualhistory.fitness, 5)) + \
            "time out:" + str(minindividualhistory.istimeout))
    fout.write('\n')
    fout.write("Min distance total:" + str(round(
       minindividualhistory.fitness2, 5)))
```

```
300     fout.write('\n')
301     for i in range(travelers):
302         fout.write("traveler " + str(i))
303         fout.write('\n')
304         fout.write("\tdep. time:" + str(round(
              minindividualhistory.timedep[i])) + str((round((
              minindividualhistory.timedep[i] - int(
              minindividualhistory.timedep[i])) * 60))))
305         fout.write('\n')
306         fout.write(str([cities2realcity[minindividualhistory.
              chrm[i][x]] for x in range(cities) if
              minindividualhistory.chrm[i][x]]))
307         fout.write('\n')
308     fout.write('-----')
309     fout.write('\n\n')
310 def printmsg():
311     if len(sys.argv) == 2:
312         if sys.argv[1] == "-q":
313             print(str(round(mindistancenow, 5)))
314     else:
315         print("Generation " + str(t))
316         print("Min distance: " + str(round(mindistancenow, 5))
              )
317         print("Min mindistance total:" + str(round(
              mindistancetotalnow, 5)))
318         print("Timed out?", str(minindividualhistory.istimeout
              ))
319
320 #begin of main
321 initialize()
322 #main loop
323 calc_fitness2()
324 for t in range(T):
325     printmsg()
326     crossover()
327     calc_fitness2()
```

```
328     select2_2()
329 print_result()
330 print("end.")
```

F 问题三添加不确定因素的程序

```
#!/usr/bin/env python3
#NOTE: run this ONLY ONCE!
from random import *
realcity = 66
dist = [[0 for col in range(realcity)] for row in range(
   realcity)]
fin = open("data2.txt", "r")
for i in range(realcity):
    data = fin.readline()
    dist[i] = [float(j) for j in data.split()]
fin.close()
print("sure to jam?")
print("Run this ONLY ONCE!")
print("[N/y]")
choice = input()
if choice == 'y':
    for i in range(jammed):
        x = randint(0, realcity - 1)
        y = randint(0, realcity - 1)
        #jam, distance three times!
        dist[x][y] = dist[x][y] * 3
        dist[y][x] = dist[y][x] * 3
    fout = open("data2.txt", "w")
    for i in dist:
        for j in i:
            fout.write(str(round(j, 3)) + ' ')
        fout.write('\n')
    fout.close()
```