队伍编号	MCB2202184
赛道	A

基于贪心算法的前瞻性动态家政订单分配

摘要

在 58 同城家政订单分配问题中, 阿姨的路线和订单的分配是对动态事件的响应。为确保消费者的需求得到满足，系统需要快速乃至动态的做出一个决策。本文基于考量三个因素的总体优化因子，提出了利用**贪心算法**进行**前瞻性分配**带时间窗的动态订单，来高效最优化包含阿姨服务分，平均通行距离和通行时间的总体优化因子。

对于问题一, 通过分析提供的订单及阿姨数据，发现订单服务起止时间都为半点整数倍的特点，将连续时间线简化为离散的时间点。基于对于某订单，选择目前时间点处于空闲中且对优化目标函数最有利的阿姨的思想，设计了**贪心算法**，建立了优化模型。以时间作为前进步长，在每次订单分配中，首先先根据现阶段阿姨的服务序列，更新判断阿姨是否有空的约束条件，并在空闲阿姨中选取对总体优化因子提升贡献最大者。最终得到了总体目标为 0.610183 对应的储存订单分配结果的 result1.txt。并根据前 50 个订单与中前 20 个阿姨所获得的阿姨服务序列进行行动轨迹绘制。

对于问题二, 在问题一模型的基础上，针对即使匹配场景，设立判断是否压单的阈值，每天的阈值根据前一天获得的数据和过往订单的特点进行动态变化，并根据获得的总体优化目标函数的大小进行调整。在问题一的约束条件的基础上增添了需要每个阿姨贡献的总体优化目标大于阈值的新约束，利用深度学习分析订单结构，建立相应的订单分配模型，模拟现实生活中动态家政订单的分配。最终得到了总体目标为 0.601100 对应的储存着订单决策结果的 result21.tex 和 result22.tex。

最后对本文所建立模型进行了讨论和分析，综合评价模型，并提出了改进和推广方向。

关键字： 即时配对　贪心算法　订单分配　动态路线　前瞻性策略

目录

一、 问题重述

1.1 问题背景

随着居民收入不断增加和对生活品质的追求, 家政服务的需求日渐增大。然而, 传统家政行业在面临越来越激烈的竞争环境中, 其管理和运营已经无法满足日益增长的需求。近几年, 互联网的快速发展使在线电商模式得到迅猛发展, 成为未来发展家政行业的重要方向。合理安排家政服务人员订单获取, 工作安排尽可能的聚集在一个或附近多个小区, 减少等待和出行时间。基于 58 到家的在线上门平台，设计高效快速的订单分配算法，能够提高企业运营能力和家政工作效率，给予客户方便、快捷、安全的服务, 同时使得服务效率得到显著提高。家政信息平台对解决家政行业长期存在的问题、实现平台和服务提供者阿姨的双赢，促进行业的发展, 具有重要的意义。

1.2 问题要求

- **问题一**　假设一天订单的总体情况已经了解，根据附件的数据，大体分析订单和阿姨的分布情况，建立起以阿姨、订单分配为自变量，优化目标为阿姨的平均服务分、每单的平均通行距离，阿姨服务订单的平均间隔时间构成的加权和的优化对象（$\alpha A - \beta B - \gamma C$），同时确保满足订单必须全部分配的约束条件的最优化模型：（a）问题中，将订单数据和阿姨数据同时输入，将最终的决策（订单基于阿姨的分配）填写到 result1.txt 中。（b）问题中，通过对前 50 订单和前 20 个阿姨的数据量进行分析，通过绘图得到直观的分配策略。
- **问题二**　通过问题一，b 小问的过渡，将整天的数据分割为每三十分钟为一个区间的不间断输入的数据集，同时将所有订单必须全部分配的约束条件具体化为压单订单必须满足服务开始的最早时间比当前时间小于于 2 个小时。在这种实时输入交互的数据集下，合理调整模型运行的机制，设计高效快速的订单分配算法，并将结果填写到 result21.txt 中。

二、 问题分析

以一段时间内产生 50 个订单，一个区域有 200 阿姨，那么对应的调度问题解空间规模将达到 50^{200}（部分为不可行解），这是一个天文数字！所以**枚举法取最优解**显然耗时长，且对解决实际订单分配问题无意义。即时分配对于优化算法的另一个要求是高实时性，算法只允许运行 2 3 秒钟的时间必须给出最终决策。

所以本题主要采用**贪心算法**，选取对目标函数的增长贡献最大的订单与阿姨的组合，优化订单分配时间。

2.1 问题一的分析

在离线派单的情况下，我们不用考虑平台和用户的实时交互关系，订单的下单时间不会直接对算法的结构产生影响，实际上，下单时间构成了一个约束条件，确保每个订单的接单时间晚于下单时间。订单的开始时间则构成另一个主要的约束条件，由于要保证订单必须被有效分配，阿姨的通行距离（即上一单坐标与下一单坐标间的欧氏距离）和上一单的结束时间影响着这个约束条件能否满足。因此，对于每个订单和接受该订单的阿姨，我们需要考虑该订单结束时间，该订单的位置距离订单结束时间后未分配的订单的整体距离。这些限制条件决定了所有订单能否别全部有效分配。在此基础上，为了使得平台有更好的口碑和竞争力，我们期望使得服务分高的阿姨尽可能有优先接取订单的优势。此时我们可以使用订单和阿姨配对的方式来遍历结果，但是此方式不满足高效快速的订单分配原则。为优化订单分配时间，我们采用贪心算法，将时间作为前进步长，评估下一个时间段中，使得最终关于三个分优化对象（A，B，C）的加权总优化对象最大的方式完成该分析。编写 Python 程序，得到最终结果。

对于问题（b），利用已经得到的算法分析前 50 个订单和前 20 个阿姨的分配情况，在订单数目远大于阿姨数目的情况下，我们发现此时将贪心算法前进的步长由最初的订单创建时间变化为订单开始的时间, 即在每次处理订单时按订单开始时间排序之后再遍历，可以防止订单因为编号靠后问题发生某订单无阿姨可分配的问题，能够进一步提高最终的优化结果。

2.2 问题二的分析

在线上派单的情况下，我们既要考虑当下订单分配对当下优化对象的影响，也要考虑是否有足够多的阿姨去分配之后时间段的新产生的订单。以及合适的阿姨是否能在合适的时间去接到订单使得接到订单的阿姨的平均服务分、平均通行距离和通行时间的总体优化因子在一个较高的收益范围中。对此我们可以采用一种前瞻性的贪心算法，在每个订单是否暂时不派单的问题上，在上一题离线派单的基础上，对每个订单设置一个接受阈值 (判断是否压单的临界点)，在大于这个阈值的情况下才会实时分配订单，否则就进行压单。设置一个审核机制可以保证订单的平均质量。

三、 模型假设

- 所有订单都要分配一个且只有一个阿姨，一个阿姨同时只能服务一个订单；
- 每个订单需要指定一个服务开始时间，这个时间的取值范围为 [最早时间，最晚时间]，且是半点的整数倍；
- 阿姨每天开始任务时必须从初始点位置出发；保洁阿姨的行驶速度为 15 千米/小时
- 任意两点的距离为欧式距离

四、 模型的建立和求解

4.1 问题一模型的建立和求解

4.1.1 宏观时间线分析

每个订单需要指定一个服务开始时间，这个时间的取值范围是确定的，并且这个开始时间必须是半点的整数倍，结合数据情况来看，服务时长依然是半点的整数倍，故起止时间均为半点的整数倍，即起止时间的数值满足：

$$(T_{k,\text{ bgn}})\,\%(30*60) == 0$$
$$(T_{k,\text{ end}})\,\%(30*60) == 0$$

其中，p 为订单的 id 。

因此，对于一个订单，起点时间的选取点并非是连续的，而是离散的。同时，观察开始时间的选取范围，同样以半点为起止，故整个宏观时间流程都是以半点为单位 (1800s) 进行的。

4.1.2 个体优化分析

为实现整体优化，需要计算每个订单对于总体盈利值的贡献或花费，贡献和花费有三种方式: 服务分平均值 A 、平均通行距离 B 、平均间隔时间 C 。

1 服务分 因为订单数是固定的，设订单数为 N ，那么总服务分为 $N \times A$ ，由此可得:

$$A = \sum_{m=0}^{n-1} \frac{A_t}{N} \times N_m$$

其中 n 为阿姨的总数，A_m 和 N_m 分别为 id 为 m 的阿姨的服务分和订单数。因此，对于 id 为 m 的阿姨，每一单的贡献数为

$$\frac{\alpha \times A_m}{N}$$

2 通行距离 同理，总通行距离为 $N \times B$ ，由此可得:

$$B = \sum_{k=0}^{N-1} \frac{B_k}{N}$$

而 B_k 取决于上一个订单的位置和本单的位置，因此对于每个订单，我们还需要得知上一订单的位置 $(x_{k,pre}, y_{k,pre})$ 和本订单的服务位置 (x_k, y_k) ，并且:

$$B_k = \sqrt{(x_k - x_{k,pre})^2 + (y_k - y_{k,pre})^2}$$

因此，对于 id 为 k 的订单，通行距离所导致的花费数为

$$\frac{\beta \times B_k}{N} = \frac{\beta \times \sqrt{(x_k - x_{k,pre})^2 + (y_k - y_{k,pre})^2}}{N}$$

3 间隔时间 同理，总间隔时间为 $N \times C$ ，由此可得:

$$C = \sum_{k=0}^{N-1} \frac{C_k}{N}$$

而 C_k 取决于上一个订单的结束时刻和本单的开始时刻，因此对于每个订单，我们还需要得知上一订单的时刻 $T_{k,pre}$ 和本订单的服务开始时刻 $T_{k,bgn}$ ，并且:

$$C_k = T_{k,bgn} - T_{k,pre}$$

因此，对于 id 为 k 的订单，间隔时间所导致的花费数为

$$\frac{\gamma \times C_k}{N} = \frac{\gamma \times (T_{k,bgn} - T_{k,pre})}{N}$$

并且，如果 $T_{k,pre}$ 无法被赋值，即阿姨刚开始工作，那么 $T_{k,pre} = T_{k,bgn} - 1800$

4.1.3 个体总分析

考虑优化完成的情况，那么我们所全部拥有的信息为:

1. 对于 id 为 m 的阿姨，我们得知她的出发位置 $S_{m,0}$ (为便于表示，与服务序列同变量名)，并且获得一个序列 $\{S_{m,1}, S_{m,2}, \ldots, S_{m,N_m}\}$ ，用以表示该阿姨完成的所有服务或者初始点，其中 $S = -(n), -(n-1), \ldots, -1, 0, 1, 2, \ldots, N-1$ 。对于每个 S ，我们存储一个 id 用于表示服务号，若为负值，则可以表示阿姨的初始点。

因此，对于一位阿姨，我们初始得知的信息有

$$\begin{cases} \text{编号: } m \\ \text{服务分: } A_m \\ \text{服务序列 (仅含出发位置) } : S_m = \{S_{m,0}\} \end{cases}$$

最终得知的信息有

$$\{ \text{服务序列: } S_m = \{S_{m,1}, S_{m,2}, \ldots, S_{m,N_m}\}$$

2. 对于 id 为 k 的服务,我们预先就拥有一个服务开始时间集合 $\{T_{k,bgn,1}, T_{k,bgn,2}, \ldots, T_{k,bgn,T_k}\}$ ，供我们选取服务开始时间，我们得知它的服务人员 s_k $(s_k = 0, 1, 2, \ldots, n-1)$ 以及服务开始时间 $T_{k,bgn}$ ，根据该服务人员 id，我们自然可以知道上一订单的位置 $(x_{k,pre}, y_{k,pre})$ 和上一服务的时间 $T_{k,pre}$ ，因此，在 (1) 的条件给定或者半给定的情况下，我们可以得知本单的总盈利值:

$$W_k = \frac{\alpha \times A_{s_k}}{N} - \frac{\beta \times \sqrt{(x_k - x_{k,pre})^2 + (y_k - y_{k,pre})^2}}{N} - \frac{\gamma \times (T_{k,bgn} - T_{k,pre})}{N}$$

同时，我们还可以记录下该单的服务结束时间 $T_{t,\text{ end}}$。

因此，对于一个服务，我们初始得知的信息有

$$\begin{cases} \text{编号: } k \\ \text{服务开始时间集合 : } T_k = \{T_{k,bgn,1}, T_{k,bgn,2}, \ldots, T_{k,bgn,T_k}\} \\ \text{本单坐标 : } (x_k, y_k) \end{cases}$$

我们最终得知的信息有

$$\begin{cases} \text{服务人员: } s_k \\ \text{服务分 : } A_{s_k} \\ \text{前单时间: } T_{k,pre} \\ \text{前单坐标: } (x_{k,pre}, y_{k,pre}) \\ \text{服务开始时间: } T_{k,bgn} \\ \text{服务结束时间: } T_{k,\text{ end}} \\ \text{盈利值 : } W_k \end{cases}$$

4.1.4 原始赋值流程

我们先随机选取一位 id 为 m 的阿姨，它的服务序列目前为空。现在选取任意服务 k 作为该阿姨的第一个服务，并且选取一个服务开始时间 $T_{k,bgn}$ 。。

我们将该服务的服务人员标记为 $s_k = m$ ，同时我们获得上一个服务的结束时间 $T_{k,pre}$ 以及前单坐标 $(x_{k,pre}, y_{k,pre})$ 。

那么我们可以计算出该订单的服务结束时间 $T_{k,\text{ end}}$ 和盈利值 W_k ，然后，将该服务添加在 id 为 m 的阿姨的序列中。

重复上述步骤，直至所有服务完成，计算所有的盈利值，即为总盈利值。我们想要设计的算法的目标，就是使这个总盈利值最大。

4.1.5 贪心算法的实现

约束 1: 一个订单的执行要在订单开始时间后进行

$$f_{i,B} + \frac{l_{\text{pos }(i,B),pos(i,C)}}{\max_j v_j} \leq fr_{(i,C)} + u_i = f_{i,C}$$
$$\forall i = 1, 2, \cdots, n, \forall j = 1, 2, \cdots, m$$

约束 2: 订单服务需要阿姨到达都具备后才能进行

$$f_{i,B} \geq r_i + p_i \quad \forall i = 1, 2, \cdots, n$$
$$f_{i,B} \geq fr_{(i,C)} \quad \forall i = 1, 2, \cdots, n$$

约束 3: 阿姨需要从上个订单位置出发, 到达下个订单的位置

以上约束条件用于判断：针对 id 为 k 的订单，id 为 m 的阿姨是否满足在已知其目前服务序列 $S_m = \{S_{m,1}, S_{m,2}, \ldots, S_{m,N_m}\}$ 的条件下，能够插入该订单服务。

1. 先将订单按照订单的开始时间排序，便于约束条件的满足。
2. 对于每个订单按以上顺序取出所有满足以上三个约束条件的所有阿姨序列。
3. 在这个阿姨序列中，计算每个阿姨贡献的总体优化目标，取出其中使得总体优化目标最大的阿姨。
4. 将每个阿姨执行的订单按阿姨 id 排序并输出到 result1.txt

程序使用 Python 语言实现，具体参照附件 Question 1.py。最终的总体优化目标大小为 0.610183。

4.1.6 阿姨执行任务列表

0 [-1, 36]

1 [-2, 7, 6]

2 [-3, 22, 10, 49]

3 [-4, 32, 20, 38, 43, 19]

4 [-5, 46, 26, 27]

5 [-6, 17, 45, 44]

6 [-7, 24, 48]

7 [-8, 2]

8 [-9, 13, 28, 33, 23, 37]

9 [-10, 15, 47, 25]

10 [-11, 30, 9, 0, 3, 16, 40]

11 [-12, 18, 31, 21]

12 [-13, 14]

13 [-14, 35]

14 [-15, 11, 12, 34]

15 [-16, 41]

16 [-17, 4]

17 [-18, 1, 42]

18 [-19, 29, 8]

19 [-20, 39, 5]

格式：**阿姨 id[-(阿姨 id+1), 订单 id]**。

其中订单 id 根据每个阿姨接单的时间顺序从先到后排列。

4.1.7 阿姨的行动轨迹图

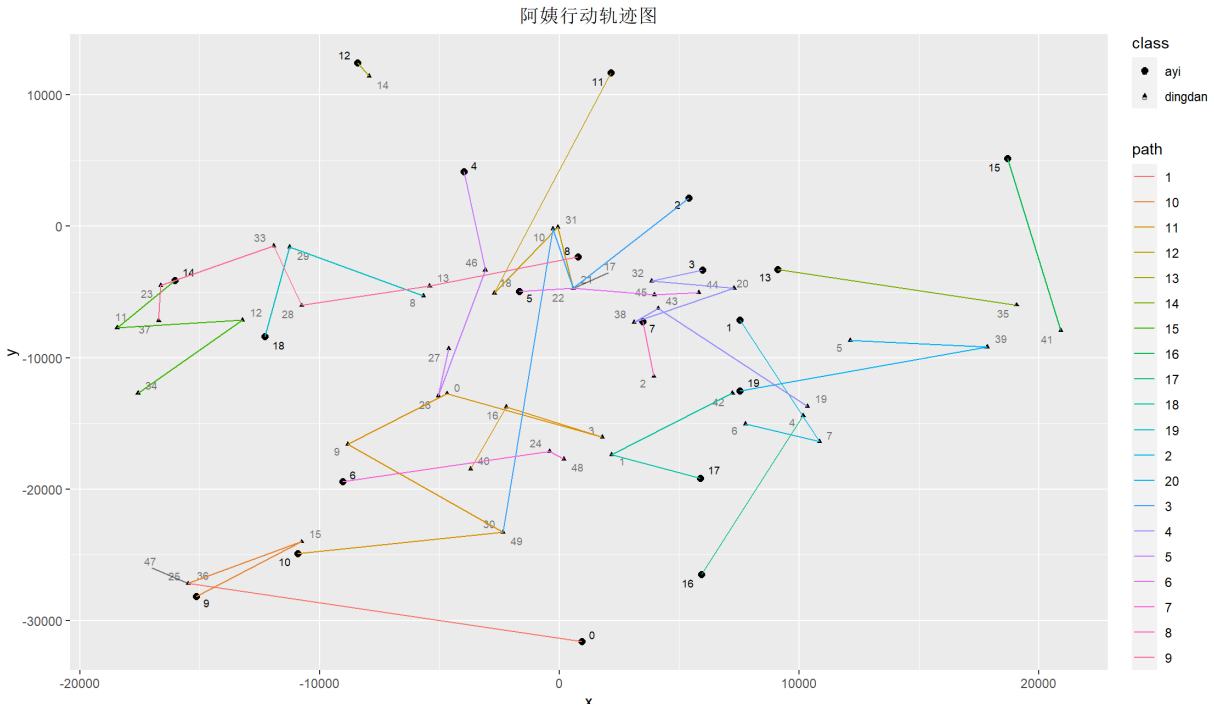

图 **1**　问题 **1**：阿姨的行动轨迹图

基于 (a) 的算法，请对附件 1 中的前 50 个订单与附件 2 中前 20 个阿姨，重新运行算法，给出阿姨的执行任务列表，并画出阿姨的行动轨迹图。（源代码参考附件 Question1_figure.R）

图 1 图例中，

1. “ayi”对应着阿姨的起始坐标，同时也为每条行动路径的起始点，规定了路径的行进方向，图中注释了每个阿姨的 id（从 0 到 19）。
2. “dingdan”对应每个订单的位置，图中注释了每个订单的 id（从 0 到 49）
3. “path”所对应的序号为“阿姨 id+1”，根据路径颜色进行区分。

4.2 问题二模型的建立和求解

4.2.1 问题二模型的修改

约束 1: 一个订单的执行要在订单开始时间后进行

$$f_{i,B} + \frac{l_{\text{pos }(i,B),pos(i,C)}}{\max_j v_j} \leq fr_{(i,C)} + u_i = f_{i,C}$$
$$\forall i = 1, 2, \cdots, n, \forall j = 1, 2, \cdots, m$$

约束 2: 订单取货需要阿姨到达都具备后才能进行

$$f_{i,B} \geq r_i + p_i \quad \forall i = 1, 2, \cdots, n$$
$$f_{i,B} \geq fr_{(i,C)} \quad \forall i = 1, 2, \cdots, n$$

约束 3: 阿姨需要从上个订单位置出发, 到达下个订单的位置

约束 4: 每个阿姨贡献的总体优化目标需要大于阈值

对于每个阿姨共享的总体优化目标所预先设定的阈值，我们需要基于过往订单的特点来设置。

为此，我们可以针对即时匹配场景，利用深度学习建立了相应的订单分配模型，分析订单结构，开发配送分配系统。系统能够模拟真实的订单生成过程和线上系统不同阈值的分配过程，并给出按照该阈值下的最终优化目标的结果。

1. 先将订单按照订单的开始时间排序，便于约束条件的满足。
2. 对于每个订单按以上顺序取出所有满足以上三个约束条件的所有阿姨序列。
3. 在这个阿姨序列中，计算每个阿姨贡献的总体优化目标，取出其中使得总体优化目标最大的阿姨。
4. 将每个订单决策写入 result22.txt 中
5. 最终将所有订单的决策结果按阿姨 id 排序并输出到 result21.txt

程序使用 Python 语言实现，具体参照附件 Question 2.py，其中我们采用的阈值为 $0.6 * 0.95$，是运用模型测试基于附件 1 中全天的订单信息得出的。最终的总体优化目标大小为 0.601100。

五、 模型的评价

5.1 模型的优点

- 模型建立过程中，将阿姨的移动速度、位置和订单的位置抽象为欧式距离，避免过多算力投入到最短路径的计算中，简化了计算形式，计算过程更加清晰。
- 在分配订单中提前根据订单开始时间将订单排序，有效避免了订单无法分配的问题，并且优化了结果。

- 在利用贪心算法的前瞻性模型中，对各订单单独进行分析求解，迭代次数少计算速度快，且设置了合适的阈值，避免收益过低的分配。

5.2 模型的缺点

- 缺少足够多的订单数据，无法分析每天的订单特点，设计出更加合适的分配阈值来决定是否压单。
- 忽略了订单和阿姨的地区特点，无法按地区分类对不同的地区进行更具针对性的分配。

5.3 模型的改进和推广方向

- 进一步丰富模型的适用空间，在实际应用中若出现订单远大于阿姨，订单无人可分配的情况下，可以进行回溯调整。
- 在确定压单阈值及算法采用方面，能够进一步优化模型。

附录 A　文件列表

表 **1**　文件概要

文件名	文件描述
Question1	问题 1 程序
Question1_figure	问题 1 路径图源代码
Question2	问题 2 程序

附录 B　代码

Question1.py

```
1
2  import numpy as np
3  import cmath as cm
4
5
6  g_aunt_data = None  # 2D-np-array, read from txt
7  g_order_data = None  # 2D-np-array, read from txt
8  g_N_aunt = int()  # num of aunt
9  g_N_order = int()  # num of order
10 g_aunt_orders_rec = None  # lst-lst-2D-array, aunts' orders
11 g_order_times = None  # lst-tpe-2D-array, orders' posbe start time
12 g_order_bgn_rec = None  # lst, orders' bgn time
13 g_order_end_rec = None  # lst, orders' end time
14 g_order_aunt_rec = None  # lst, orders' aunt
15 g_order_win_rec = None  # lst, orders' win
16
17
18 # this func is used for reading datas
19 # it returns 2D-array
20 def readData(typ: int, file_name: str):
21     datas = list()
22     with open('data/{0}'.format(file_name)) as f:
23         for line in f:
24             line = line[:-1]
25             datas.append(tuple(line.split(",")))
26     '''
27     订单数据字段:
28     0: id (int)     1: create_time (int)
29     2: service_first_time (int)     3: service_last_time (int)
30     4: service_unit_time (int)      5: x (int)      6: y (int)
```

```
    '''
    order_type = np.dtype({
        'names': ['id', 'T_cc', 'T_ff', 'T_ll', 'T_uu', 'x', 'y'],
        'formats': ['i', 'i', 'i', 'i', 'i', 'i', 'i']})
    '''
    阿姨数据字段:
    0: id   1: service_score
    2: x    3 :y
    '''
    aunt_type = np.dtype({
        'names': ['id', 'score', 'x', 'y'],
        'formats': ['i', 'f', 'i', 'i']})

    if (typ == 1):
        a = np.array(datas, dtype=order_type)
    elif (typ == 2):
        a = np.array(datas, dtype=aunt_type)
    return a


# fill in the order time list
def fillInOrdTmLst() -> None:
    # global var used
    global g_N_order
    global g_order_data
    global g_order_times
    for order in g_order_data:
        # UNIT: second
        g_order_times[order['id']] = tuple(
            range(order['T_ff'], order['T_ll'] + 1800, 1800))

# get the most good aunt and time for the order


def getPri(order: int) -> None:

    # global var used
    global g_N_order
    global g_N_aunt
    global g_order_data
    global g_aunt_data
    global g_order_times
    global g_aunt_orders_rec
    print("getPri({0}), id:{1}".format(
        order, g_order_data[order]['id']), end=' ')
    # output info
    output_aunt = int()  # aunt id for output
```

```
    output_max_win = -float('inf')
    output_T_cur = int()  # current start time, UNIT: sec
    # info from order id
    T_uu = g_order_data[order]['T_uu']  # UNIT: min
    T_start_tp = g_order_times[order]  # tp, UNIT: sec
    cur_x = g_order_data[order]['x']  # UNIT: m
    cur_y = g_order_data[order]['y']  # UNIT: m
    # traverse all the aunts
    '''
    the algorithm for the traversing:
    1. if the aunt never finished any order, then T_pre = 0
       else T_pre should be got from the previous order
       T_pre is the ending time for the last order of the aunt
    2. search all the posbe time, if T_pre = 0, then the
       T_diff = 1800(s), no matter the distance , we cal the win
       and then break at the first posbe time,
    3. else, if the T_diff is not suitable for the aunt to arrive,
       then continue, else cal the win and then break at that time
    '''
    for i_aunt in range(g_N_aunt):
        score = g_aunt_data[i_aunt]['score']
        pre_order = g_aunt_orders_rec[i_aunt][-1]
        if (pre_order < 0):
            pre_x = g_aunt_data[i_aunt]['x']  # UNIT: m
            pre_y = g_aunt_data[i_aunt]['y']  # UNIT: m
            T_pre = 0  # UNIT: sec
        else:
            pre_x = g_order_data[pre_order]['x']  # UNIT: m
            pre_y = g_order_data[pre_order]['y']  # UNIT: m
            T_pre = g_order_end_rec[pre_order]  # UNIT: sec

        for j_time in T_start_tp:
            T_cur = j_time  # UNIT: sec
            if (T_pre == 0):
                T_pre = T_cur - 1800
                res = calcWin(score, cur_x, cur_y, pre_x, pre_y, T_cur, T_pre)
                if (res > output_max_win):
                    output_max_win = res
                    output_aunt = i_aunt
                    output_T_cur = T_cur
            else:
                T_diff = T_cur - T_pre
                dis = abs(cm.sqrt((abs(cur_x-pre_x))**2
                                  + (abs(cur_y-pre_y))**2)) / 1000
                if ((dis / 15) * 60 * 60) > T_diff:
                    continue
                else:
```

```
                    res = calcWin(score, cur_x, cur_y,
                                  pre_x, pre_y, T_cur, T_pre)
                    if (res > output_max_win):
                        output_max_win = res
                        output_aunt = i_aunt
                        output_T_cur = T_cur
    '''
    aunts' order lst unpdated
    order ending time lst updated
    order win lst unpdated
    '''
    if (output_max_win == -float('inf')):
        return False
    g_aunt_orders_rec[output_aunt].append(order)
    g_order_aunt_rec[order] = output_aunt
    g_order_bgn_rec[order] = output_T_cur
    g_order_end_rec[order] = output_T_cur + T_uu * 60
    g_order_win_rec[order] = output_max_win
    print(output_max_win)
    # if getPri
    return output_max_win


# calculate the win value (float)
def calcWin(score: float, cur_x: int, cur_y: int,
            pre_x: int, pre_y: int, T_cur: int, T_pre: int) -> float:
    ALP = 0.78
    BET = 0.025
    GAM = 0.195
    '''
    the UNIT for distance should be km = 1000 m
    the UNIT for time should be hour = 3600 sec
    '''
    win = (ALP * score
           - BET * abs(cm.sqrt((abs(cur_x-pre_x))**2
                               + (abs(cur_y-pre_y))**2)) / 1000
           - GAM * (T_cur - T_pre) / 60 / 60)
    # print("dist: ", BET * abs(cm.sqrt( (abs(cur_x-pre_x))**2
    # + (abs(cur_y-pre_y))**2 )) / 1000)
    # print(T_cur, T_pre, T_cur-T_pre)
    # print("tist: ", GAM * (T_cur - T_pre) / 60 / 60)
    return win


if __name__ == '__main__':
    # choose the question
    print('choose 1a or 1b')
```

```
    question = input('Question: ')
    # read data from txt
    order_file_name = 'order_data_{0}.txt'.format(question)
    aunt_file_name = 'aunt_data_{0}.txt'.format(question)
    g_order_data = readData(1, order_file_name)  # 2D-array
    # sorted by start time, used for traversing
    g_sorted_order_data = np.sort(g_order_data, axis=0, order=['T_ff', 'T_ll'])
    g_aunt_data = readData(2, aunt_file_name)  # 2D-array
    # the num of aunts and order
    g_N_order = len(g_order_data)
    g_N_aunt = len(g_aunt_data)
    print('g_N_order:', g_N_order)
    print('g_N_aunt:', g_N_aunt)
    # updated the times lst for order
    g_order_times = [None for _ in range(g_N_order)]
    fillInOrdTmLst()
    # recording arrays init
    g_aunt_orders_rec = [[-1-i] for i in range(g_N_aunt)]
    g_order_aunt_rec = [None for _ in range(g_N_order)]
    g_order_bgn_rec = [None for _ in range(g_N_order)]
    g_order_end_rec = [None for _ in range(g_N_order)]
    g_order_win_rec = [None for _ in range(g_N_order)]
    # traverse order prior in 'T_ff', 'T_ll'
    for i in range(g_N_order):
        getPri(g_sorted_order_data[i]['id'])
    # print the aunts' order lst for painting the graph
    for i in range(len(g_aunt_orders_rec)):
        print(i, g_aunt_orders_rec[i])
    # calculate the result
    res = 0
    for i in g_order_win_rec:
        if i != None:
            res += i
    print("res:", res)
    print("avg_res:", res / g_N_order)
    print('-'*50)
    # result for the answer
    for i in range(g_N_order):
        print('{0},{1},{2}'.format(i, g_order_bgn_rec[i], g_order_aunt_rec[i]))
```

Question1_figure.R

```
theUrl <-"附件1：订单数据.txt"
dingdan <-read.table(file=theUrl, header=TRUE, sep=",")
theUr2<-"附件2：阿姨数据.txt"
ayi<-read.table(file=theUr2, header=TRUE, sep=",")
theUr3 <-"Result.txt"
result<-read.table(file=theUr3, header=FALSE, sep=",")
```

```
#Result.txt:
#0,1662786000,10
#1,1662768000,17
#2,1662771600,7
#3,1662793200,10
#4,1662800400,16
#5,1662775200,19
#6,1662773400,1
#7,1662768000,1
#8,1662773400,18
#9,1662777000,10
#10,1662795000,2
#11,1662768000,14
#12,1662773400,14
#13,1662771600,8
#14,1662782400,12
#15,1662768000,9
#16,1662800400,10
#17,1662768000,5
#18,1662789600,11
#19,1662802200,3
#20,1662778800,3
#21,1662802200,11
#22,1662786000,2
#23,1662793200,8
#24,1662768000,6
#25,1662786000,9
#26,1662786000,4
#27,1662793200,4
#28,1662777000,8
#29,1662768000,18
#30,1662768000,10
#31,1662795000,11
#32,1662771600,3
#33,1662784200,8
#34,1662786000,14
#35,1662768000,13
#36,1662786000,0
#37,1662800400,8
#38,1662786000,3
#39,1662768000,19
#40,1662807600,10
#41,1662800400,15
#42,1662773400,17
#43,1662795000,3
#44,1662786000,5
```

```
#45,1662777000,5
#46,1662775200,4
#47,1662773400,9
#48,1662773400,6
#49,1662800400,2

dingdan_50<-dingdan[1:50,]
dingdan_50<-cbind(dingdan_50,result[,2])

ayi_20<-ayi[1:20,]
ayi_20_new<-ayi_20[,-2]
ayi_20_new$class="ayi"

dingdan_50
dingdan_50_new=dingdan_50[,-c(2,3,4,5)]
dingdan_50_new$class="dingdan"

ayi_20_new$path<-c("1","2","3","4","5","6","7","8","9","10","11","12","13","14","15
    ","16","17","18","19","20")
ayi_20_new

dingdan_50_new[37,"path"]="1"
dingdan_50_new[c(8,7),"path"]="2"
dingdan_50_new[c(23,11,50),"path"]="3"
dingdan_50_new[c(33,21,39,44,20),"path"]="4"
dingdan_50_new[c(47,27,28),"path"]="5"
dingdan_50_new[c(18,46,45),"path"]="6"
dingdan_50_new[c(25,49),"path"]="7"
dingdan_50_new[3,"path"]="8"
dingdan_50_new[c(14,29,34,24,38),"path"]="9"
dingdan_50_new[c(16,48,26),"path"]="10"
dingdan_50_new[c(31,10,1,4,17,41),"path"]="11"
dingdan_50_new[c(19,32,22),"path"]="12"
dingdan_50_new[15,"path"]="13"
dingdan_50_new[36,"path"]="14"
dingdan_50_new[c(12,13,35),"path"]="15"
dingdan_50_new[42,"path"]="16"
dingdan_50_new[5,"path"]="17"
dingdan_50_new[c(2,43),"path"]="18"
dingdan_50_new[c(30,9),"path"]="19"
dingdan_50_new[c(40,6),"path"]="20"
dingdan_50_new<-dingdan_50_new[order(dingdan_50_new$'result[, 2]'),]
dingdan_50_new=dingdan_50_new[,-4]
data1<-rbind(ayi_20_new,dingdan_50_new)


library(ggrepel)
```

```
q1<-ggplot(data=data1)+
  geom_point(aes(x=x,y=y,shape=class,size=class))+
  scale_size_manual(values=c(2,1))+
  scale_alpha_manual(values = c(1,0.5))+
  geom_text_repel(aes(x=x,y=y,label=id,alpha=class),size=2.5,max.overlaps = 40)+
  geom_path(aes(x=x,y=y,color=path))+
  ggtitle("阿姨行动轨迹图")+
  theme(plot.title =element_text(hjust = 0.5))
q1
```

Question2.py

```

import numpy as np
import cmath as cm



g_aunt_data = None # 2D-np-array, read from txt
g_order_data = None # 2D-np-array, read from txt
g_N_aunt = int() # num of aunt
g_N_order = int() # num of order
g_aunt_orders_rec = None # lst-lst-2D-array, aunts' orders
g_order_times = None # lst-tpe-2D-array, orders' posbe start time
g_order_bgn_rec = None # lst, orders' bgn time
g_order_end_rec = None # lst, orders' end time
g_order_aunt_rec = None # lst, orders' aunt
g_order_win_rec = None # lst, orders' win
g_cut_num = 0.6 * 0.95
g_oper_rec = None # record every operation

'''
订单数据字段:
0: id (int)     1: create_time (int)
2: service_first_time (int)     3: service_last_time (int)
4: service_unit_time (int)      5: x (int)      6: y (int)
'''
order_type = np.dtype({
    'names':['id', 'T_cc', 'T_ff', 'T_ll', 'T_uu', 'x', 'y'],
    'formats':['i', 'i', 'i', 'i', 'i', 'i', 'i'] })
'''
阿姨数据字段:
0: id   1: service_score
2: x    3 :y
'''
aunt_type = np.dtype({
    'names':['id', 'score', 'x', 'y'],
    'formats':['i', 'f', 'i', 'i'] })
```

```
37
38
39  # this func is used for reading datas
40  # it returns 2D-array
41  def readData(typ:int, file_name:str):
42      datas = list()
43      with open('data/{0}'.format(file_name)) as f:
44          for line in f:
45              line = line[:-1]
46              datas.append(tuple(line.split(",")))
47      if (typ == 1):
48          a = np.array(datas, dtype = order_type)
49      elif (typ == 2):
50          a = np.array(datas, dtype = aunt_type)
51      return a
52
53
54  # fill in the order time list
55  def fillInOrdTmLst() -> None:
56      # global var used
57      global g_N_order
58      global g_order_data
59      global g_order_times
60      for order in g_order_data:
61          # UNIT: second
62          g_order_times[order['id']] = tuple(
63              range(order['T_ff'], order['T_ll'] + 1800, 1800))
64
65  # get the most good aunt and time for the order
66  def getPri(order:int) -> None:
67
68      # global var used
69      global g_N_order
70      global g_N_aunt
71      global g_order_data
72      global g_aunt_data
73      global g_order_times
74      global g_aunt_orders_rec
75      print("getPri({0}), id:{1},".format(order, g_order_data[order]['id']), end=' ')
76      # output info
77      output_aunt = int() # aunt id for output
78      output_max_win = -float('inf')
79      output_T_cur = int() # current start time, UNIT: sec
80      # info from order id
81      T_uu = g_order_data[order]['T_uu'] # UNIT: min
82      T_start_tp = g_order_times[order] # tp, UNIT: sec
83      cur_x = g_order_data[order]['x'] # UNIT: m
```

```
    cur_y = g_order_data[order]['y'] # UNIT: m
    # traverse all the aunts
    '''
    the algorithm for the traversing:
    1. if the aunt never finished any order, then T_pre = 0
       else T_pre should be got from the previous order
       T_pre is the ending time for the last order of the aunt
    2. search all the posbe time, if T_pre = 0, then the
       T_diff = 1800(s), no matter the distance , we cal the win
       and then break at the first posbe time,
    3. else, if the T_diff is not suitable for the aunt to arrive,
       then continue, else cal the win and then break at that time
    '''
    for i_aunt in range(g_N_aunt):
        score = g_aunt_data[i_aunt]['score']
        pre_order = g_aunt_orders_rec[i_aunt][-1]
        if (pre_order < 0):
            pre_x = g_aunt_data[i_aunt]['x'] # UNIT: m
            pre_y = g_aunt_data[i_aunt]['y'] # UNIT: m
            T_pre = 0 # UNIT: sec
        else:
            pre_x = g_order_data[pre_order]['x'] # UNIT: m
            pre_y = g_order_data[pre_order]['y'] # UNIT: m
            T_pre = g_order_end_rec[pre_order] # UNIT: sec

        for j_time in T_start_tp:
            T_cur = j_time # UNIT: sec
            if (T_pre == 0):
                T_pre = T_cur - 1800
                res = calcWin(score, cur_x, cur_y, pre_x, pre_y, T_cur, T_pre)
                if (res > output_max_win):
                    output_max_win = res
                    output_aunt = i_aunt
                    output_T_cur = T_cur
            else:
                T_diff = T_cur - T_pre
                dis = abs(cm.sqrt( (abs(cur_x-pre_x))**2
                    + (abs(cur_y-pre_y))**2 )) / 1000
                if ((dis / 15) * 60 * 60) > T_diff:
                    continue
                else:
                    res = calcWin(score, cur_x, cur_y, pre_x, pre_y, T_cur, T_pre)
                    if (res > output_max_win):
                        output_max_win = res
                        output_aunt = i_aunt
                        output_T_cur = T_cur
    '''
```

```
    aunts' order lst unpdated
    order ending time lst updated
    order win lst unpdated
    '''
    # if (output_max_win == -float('inf')):
    #     return False
    g_aunt_orders_rec[output_aunt].append(order)
    g_order_aunt_rec[order] = output_aunt
    g_order_bgn_rec[order] = output_T_cur
    g_order_end_rec[order] = output_T_cur + T_uu * 60
    g_order_win_rec[order] = output_max_win
    print('win:', output_max_win)
    # if getPri
    # print(g_aunt_orders_rec[output_aunt])
    return output_aunt, order, output_max_win, output_T_cur

def unsolve(aunt:int, order: int):
    global g_aunt_orders_rec
    global g_order_aunt_rec
    global g_order_bgn_rec
    global g_order_end_rec
    global g_order_win_rec
    print("unsolve({0}), id:{1}, aunt_id:{2}".format(order, order, aunt))
    if (g_aunt_orders_rec[aunt][-1] == order):
        g_aunt_orders_rec[aunt].pop()
        g_order_aunt_rec[order] = None
        g_order_bgn_rec[order] = None
        g_order_end_rec[order] = None
        g_order_win_rec[order] = None
    else:
        return False
    # print(g_aunt_orders_rec[aunt])
    return order

# calculate the win value (float)
def calcWin(score:float, cur_x:int, cur_y:int,
            pre_x:int, pre_y:int, T_cur:int, T_pre:int) -> float:
    ALP = 0.78
    BET = 0.025
    GAM = 0.195
    '''
    the UNIT for distance should be km = 1000 m
    the UNIT for time should be hour = 3600 sec
    '''
    win = ( ALP * score
          - BET * abs(cm.sqrt( (abs(cur_x-pre_x))**2
                    + (abs(cur_y-pre_y))**2 )) / 1000
```

```
            - GAM * (T_cur - T_pre) / 60 / 60)
    # print("dist: ", BET * abs(cm.sqrt( (abs(cur_x-pre_x))**2
                    # + (abs(cur_y-pre_y))**2 )) / 1000)
    # print(T_cur, T_pre, T_cur-T_pre)
    # print("tist: ", GAM * (T_cur - T_pre) / 60 / 60)
    return win



if __name__ == '__main__':
    # choose the question
    # print('choose 1a or 1b')
    question = '2'
    # read data from txt
    order_file_name = 'order_data_{0}.txt'.format(question)
    aunt_file_name = 'aunt_data_{0}.txt'.format(question)
    g_order_data = readData(1, order_file_name) # 2D-array
    g_aunt_data = readData(2, aunt_file_name) # 2D-array
    # g_sorted_order_data = np.sort(g_order_data, axis=0, order=['T_ff', 'T_ll'])
    # the num of aunts and order
    g_N_order = len(g_order_data)
    g_N_aunt = len(g_aunt_data)
    print('g_N_order:', g_N_order)
    print('g_N_aunt:', g_N_aunt)
    # updated the times lst for order
    g_order_times = [ None for _ in range(g_N_order) ]
    fillInOrdTmLst()
    # recording arrays init
    g_aunt_orders_rec = [ [-1-i] for i in range(g_N_aunt) ]
    g_order_aunt_rec = [ None for _ in range(g_N_order) ]
    g_order_bgn_rec = [ None for _ in range(g_N_order) ]
    g_order_end_rec = [ None for _ in range(g_N_order) ]
    g_order_win_rec = [ None for _ in range(g_N_order) ]
    g_oper_rec = [ ]
    # traverse order prior in 'T_ff', 'T_ll'
    start_idx = 0
    end_idx = 0
    start_time = g_order_data[start_idx]['T_cc']
    start_time = start_time - start_time % 1800
    end_time = start_time + 1800
    pre_data = list()
    # pre_data = list()
    while end_idx <= g_N_order - 1:
        # get next end_idx
        while (end_idx <= g_N_order - 1 and g_order_data[end_idx]['T_cc'] <
    end_time):
            end_idx += 1
```

```
        # current new data
        g_cur_data = g_order_data[ start_idx: end_idx]
        # add previous data
        for i in pre_data:
            g_cur_data = np.r_[g_cur_data,[i]]
        sorted_cur_data = np.sort(g_cur_data, axis=0, order=['T_ff', 'T_ll'])

        # get the new current info
        new_order = list()
        new_aunt = list()
        new_win = list()
        new_T_start = list()
        # current info record
        for i in range(len(sorted_cur_data)):
            aunt, order, win, T_start = getPri(sorted_cur_data[i]['id'])
            new_order.append(order)
            new_aunt.append(aunt)
            new_win.append(win)
            new_T_start.append(T_start)

        # get the pre_data np-array and do oper record
        oper_rec = [0 for _ in range(len(sorted_cur_data))]
        pre_data = list()
        for i in range(len(new_win)):
            if sorted_cur_data[i]['T_ff'] - end_time > 7200:
                if new_win[i] < g_cut_num:
                    if (unsolve(new_aunt[i], new_order[i])) != False:
                        pre_data.append(sorted_cur_data[i])
                        oper_rec[i] = -1
        pre_data = np.array(pre_data, dtype = order_type)

        # add the operation to record
        for i in range(len(oper_rec)):
            if (oper_rec[i] == 0):
                g_oper_rec.append((end_time, new_order[i], new_T_start[i], new_aunt
    [i], 0))
            else:
                g_oper_rec.append((end_time, new_order[i], -1, -1, 1))

        # next time period
        start_idx = end_idx
        if end_idx >= g_N_order:
            break
        start_time = end_time
        end_time = start_time + 1800

    # print the aunts' order lst for painting the graph
```

```
    for i in range(len(g_aunt_orders_rec)):
        print(i, g_aunt_orders_rec[i])
    # calculate the result
    win = 0
    for i in g_order_win_rec:
        if i != None: win += i
    print("res:", win)
    print("avg_res:", win / g_N_order)
    print('-'*50)
    # result for the answer
    for i in range(g_N_order):
        print('{0},{1},{2}'.format(i, g_order_bgn_rec[i], g_order_aunt_rec[i]))
    print('-'*50)
    for i in g_oper_rec:
        print('{0},{1},{2},{3},{4}'.format(i[0], i[1], i[2], i[3], i[4]))
```