参赛密码 ____________

（由组委会填写）

"华为杯"第十四届中国研究生数学建模竞赛

学　　校	北京交通大学	
参赛队号	10004025	
队员姓名	1.	何佳毅
	2.	黄丽丽
	3.	施沫寒

参赛密码 ____________

（由组委会填写）

“华为杯”第十四届中国研究生数学建模竞赛

题 目 **基于贪婪算法的航班恢复问题启发式求解**

摘 要

随着社会的发展，人们生活质量的提高、生活的节奏加快，航空出行已经成了人们日常生活中不可缺少的一部分。而由于各种各样的原因，飞机延误似乎也是普遍存在的，这也严重困扰着出行旅客。而作为航空公司来说，长时间的航班延误和旅客滞留将带来巨大的经济损失，因此，快速恢复航班通行能力、疏散滞留旅客显得尤为迫切。这是一个典型的 NP-hard 问题，因此，直接搜索全局最优解是件困难的时间，本论文旨在通过建立时间轴来记录航班起降，利用合理的假设对实际问题进行简化，基于贪婪算法对问题进行步步寻优，最终得到一个满意的航班恢复方案。

对于问题一，不用考虑飞机载客量且只考虑机型为 9 的航班，因此不用考虑飞机置换的成本，我们只需要确定飞机是否能够按照原定时间起飞即可。我们通过遍历时间轴，优先指派可尽快值机的飞机承担航班任务并得到新的航班安排，这其中共涉及 97 架航班，其中 14 架延误，航班准点率为 85.57%，总延误时长为 18 小时 59 分。

对于问题二，不用考虑飞机载客量但需要考虑 Schedules 中提到的所有航班任务，并且我们还需在问题一的基础上考虑不同型飞机替飞产生的延误成本，即发生一次不同型飞机替飞，飞机延误总时间将增加 30min。我们根据延误成本制定出详细的值机决策，在时间轴的每一点上逐步推进进行决策，最终通过遍历时间轴得到所有航班安排，共涉及 749 架航班，其中 112 架延误，航班准点率为 85.05%，航班总延误时长为 222 小时 49 分。

对于问题三，目标函数为旅客总体延误时长最短，需要在问题二的基础上进一步讨论载客量的影响，且题目中假设所有机型全部满载，若乘客无法登机则未登机乘客将每人增加延误成本 2h，因此我们更新了值机决策，即在不同型飞机置换替飞情况下优先考虑载客量较大的飞机。最终航班安排共涉及 749 架航班，其中 108 架延误，15 架取消，航班准点率为 83.58%。总延误时长为 211 小时 34 分，旅客总延误时长为 40171 小时 51 分。

对于问题四，可看作是在第二问和第三问基础上的拓展，虽不需要考虑不同型飞机置换的成本，但由于考虑了乘客的连程航班、同行旅客人数及换乘时间间隔而增加了求解难度。我们通过筛选航班信息，将所有不造成旅客延误的航班 delay 以提供更宽松的调度空间，尽量缩短旅客延误总时长。其次，连程乘客需在到达 45min 后方可搭乘下一架航班，而对于后程航班来说，往往有多个前序航班与之对应，即在连程航班环节中较后的航班，其乘客将会来自不同前序航班，且到达时间各异。这种情况下，我们应首先对是否等待延误到达的旅客登机以及等待时长作出决策，然后再对值机飞机进行决策。与问题三不同的是，问题四中涉及航班并不完全满载，因此在不同型飞机置换替飞情况下优先考虑载客量大于实际登机人数的飞机。最终航班安排共涉及 749 架航班，其中 96 架延误，118 架取消，航班准点率为 71.43%。总延误时长为 177 小时 58 分，旅客总延误时长为 37200 小时。

我们考虑到该问题方案的实际需求方，因此本文始终从航空公司的角度着手来考虑和制定航班恢复方案，使得方案跟贴近实际需求，力求能在实际应用中发挥作用。我们利用 Python 来对方案进行实现，取得了较好的效果。

关键词：贪婪算法；航班恢复；NP-hard 问题；航班安排。

1. 问题重述

1.1 研究背景

航空出行已经成为当下许多旅客的便捷选择，但是现实中如果飞机航班不能按照原本航空计划执行，就会给航空公司带来不可估量的经济损失，同时影响旅客的出行。通常情况下，造成航班不能正常执行的原因有几类：自然因素，不可预测的突发因素，管理方面因素等等。

目前大部分航空公司的航班安排比较稠密，如果某个航班出现故障，后续的一系列航班就有可能受到影响，成千上万的旅客出行将被延误。航班延误已经成为亟待解决的问题，与之对应的航班恢复问题也由于反常变化的气候和其他突发事件的增多受到世界各大航空公司和民航管理机构的重视。航班恢复问题解决的难点主要在于恢复方案的即时性。有经验的调度员通常可以人工调整，但这种情况只能考虑到影响航班的一些基本因素，而很难考虑到全局网络的优化，因此目前一般借助于计算机来求解这类数学优化模型。航班恢复问题有两个关键因素需要考虑：一是创建合适的数学优化模型；二是找到合适的算法快速求解该数学模型。学术界在航班恢复问题上研究已久，但是理论研究通常只限于有限的时间和空间，设置的约束条件也仅是实际情况的部分子集，这样的研究一般很难直接运用于实际生产中，因此创建更符合实际的数学模型和采用行之有效的求解算法是解决航班恢复问题的关键。

1.2 已知信息

（1）因受到暴风雪的影响，机场 OVS 将于 2016 年 4 月 22 日的 18:00 到 21:00 之间关闭，该时间段内 OVS 机场不能起飞或降落任何的航班，而该时间段外一切航班正常。

（2）由于跑道限制，OVS 机场每 5 分钟最多能起飞 5 架飞机，同时降落 5 架飞机，其它机场则不需要考虑跑道限制。

1.3 问题提出

基于已知信息建立数学模型，并在给定的航班数据下，求解以下 4 个问题：

问题一：不考虑旅客信息及其他机型，且满足原计划航班尽可能不被取消的条件下，重新规划机型 9 的航班计划，并给出起飞时间表和延误时间，保证机型 9 的所有航班总体延误时间最短。

问题二：不考虑旅客信息，假设同机型所有飞机载客量相同，同机型航班间调整没有成本，但在不同机型间调整有成本(等价于航班延误半小时)。在这种假设且满足原计划航班尽可能不被取消的条件下，重新规划所有机型的所有航班，并给出起飞时间表和延误时间，保证所有机型的所有航班总体延误时间最短。

问题三：进一步考虑飞机的载客量，假设在不同机型间调整的成本除了航班本身延误半小时外，还要加上不能登机旅客的成本（这里假定旅客行程都是直达的，并假设所有航班都是 100%的上座率）。一名旅客无法登机等价于该旅客延误 2 小时，在这种假设下重新规划所有机型的所有航班以保证旅客总体延误时间最短。

问题四：在第二题的基础上，假设在不同机型间调整航班不考虑成本。旅客数据中提供了行程信息（包括旅客号，同行旅客数量和相应的航班）。每个旅客行程中的相连航班间最少需要 45 分钟间隔时间用于中转，且旅客的延误按照旅客计划到达最终目的地时间为基准计算。在这种假设下重新规划所有机型的所有航班以保证旅客总体延误时间最短。

2、模型假设

1. 假设题目中涉及航班时间均为 UTC 国际标准时间。
2. 假设航班不存在提前值机的情况，即实际起飞时间不得早于原计划起飞时间。
3. 假设各航班的飞行时间为常量，即为该航班原计划到达时间与起飞时间之差。
4. 假设每架飞机的连续航程连贯，即前一航班的到达机场与后一航班的起飞机场必须相同，而且前一航班到达时间与后一航班起飞时间之间的最小间隔时间为 45 分钟。
5. 假设航班最大延误时间为 5 小时，超过 5 小时则该航班取消。
6. 所有飞机的执行的首个航班任务必须从该飞机的起点机场起飞。
7. 所有飞机的第一个航班不早于飞机的最早可用时间起飞，最后一个航班不能晚于最晚可用时间到达。
8. 航班决策根据时间轴向前推进。
9. 不考虑机场可停留飞机的容量。
10. 假设所有机场可以全天 24 小时工作。
11. 同一旅客号视为同一组，不能拆分登机。
12. 假设航班唯一编号与飞机尾号不绑定，按照一定优先顺序选择值机飞机。
13. 假设航班任务与旅客号绑定，即认为改签或退票等行为发生在旅客方，与航空公司决策无关，航空公司不会强制要求旅客改签或退票。
14. 第三问中，题目假设每个航班均为满载，则在该问中不考虑多人共享同一旅客号的条件。

3、符号说明

符号	意义
A	机场集合
a	某一机场
F	航班集合
Fa	机场 a 的航班集合
f	某一航班
K	同机型飞机集合
K_a	停留在机场 a 的飞机集合
k	同机型飞机
tq_f^k	飞机 k 执行航班 f 的起飞时间
tg_f^k	飞机 k 执行航班 f 的到达时间
t_c	最小间隔时间
R	飞机可行路径集合
r	某条路径
c_f	取消航班 f 的延误成本
d_f^k	用同型飞机 k 执行航班 f 的航班延误成本
x_f^k	用同型飞机 k 执行航班 f 的决策变量
y_f	航班 f 执行与否的决策变量
$f(k)$	航班 f 原本指派飞机同机型的飞机集合
$f(s)$	航班 f 原本指派飞机不同机型的飞机集合
s	不同机型飞机
d_f^s	用不同机型飞机 s 执行航班 f 的航班延误成本
d_s	用不同机型飞机执行航班 f 的额外延误成本

x_f^s	用同型飞机s执行航班 f 的决策变量
$f(m)$	航班 f 指派大于原机型载客量飞机集合
$f(l)$	航班 f 指派小于原机型载客量飞机集合
v_k	同机型飞机k载客量
v_l	载客量小于原机型的飞机l的载客量
d_{kl}	未登机旅客的延误成本
I	旅客集合
i	某一旅客
T_f	航班 f 原计划到达时间
p_f	航班 f 被取消时每个乘客的延误成本
h_f^k	用同机型飞机k执行航班 f 实际载客量
h_f^s	用不同机型飞机s执行航班 f 实际载客量

4、问题一模型建立与求解

4.1 问题描述及模型建立

问题一要求对所有受到 18:00 到 21:00 机场 OVS 关闭影响的机型 9 航班进行重新规划，使得原计划航班尽量不被取消，同时所有航班总体延误时间最短。题目中只考虑了单一机型，即飞机置换只在同一机型的飞机间发生，同型飞机置换无成本。我们需要对航班计划重新规划来最小化航班总体延误时间，并得到起飞时间的最优解。

针对问题一，由于航班只有执行和取消两种选择，我们考虑将目标函数航班总体延误时间分解为两部分：第一部分为被执行的所有航班的延误时间成本，第二部分为被取消的所有航班的延误时间成本。具体构建过程如下：

（1）目标函数建立

我们先定义以下符号：A 为机场集合，a 为其中的某一机场；F 为航班集合，Fa 为机场 a 的航班集合，f 为其中的某一航班；K 为可安排的机型 9 飞机集合，K_a 为停留在机场 a 的飞机集合，k 为其中的某架飞机；tq_f^k 为飞机 k 执行航班 f 的起飞时间，tg_f^k 为飞机 k 执行航班 f 的到达时间，t_c 为最小间隔时间，依据题目设为 45min；R 为飞机可行路径集合，r 为其中的某条路径；c_f 为取消航班 f 的延误成本，；d_f^k 为用飞机 k 执行航班 f 的航班延误成本。

根据航班的执行与否定义决策变量：

$$x_f^k = \begin{cases} 1, 如果飞机k执行航班f \\ 0，其他 \end{cases} \tag{4.1}$$

$$y_f = \begin{cases} 1, 如果航班f 被取消 \\ 0，其他 \end{cases} \tag{4.2}$$

应用上述符号，结合执行航班的延误成本和取消航班的延误成本，可建立如下目标函数

$$\min Z = \sum_{k\in K}\sum_{f\in F} d_f^k x_f^k + \sum_{f\in F} c_f y_f \tag{4.3}$$

其中第一项表示执行航班延误时间成本，第二项表示被取消航班的延误时间成本。由于航班被取消的延误成本较大，因此若要使目标函数的值最优，应做到让第二项为 0。

（2）约束条件

由题设，我们将已知条件作为目标函数的约束条件引入优化模型中：

i. 航班覆盖约束：每个航班只有被取消和被执行两种状态，故有约束如下

$$\sum_{k\in K} x_f^k + y_f = 1, f \in F \tag{4.4}$$

ii. 飞机平衡约束：某个机场飞行航班任务的飞机数量不能超过停留在该机场可供指派的飞机数量，故有如下约束

$$\sum_{k\in K}\sum_{f\in F}F_a x_{fa}^k \le K_a, a\in A \tag{4.5}$$

iii. 飞机指派约束：每架飞机最多只能执行一个航班，每个航班也最多指派一架飞机，故有如下约束

$$\begin{aligned}&\sum_{k\in K} x_{fr}^k \le 1,\ f\in 航线r上的航班,\ r\in R\\&\sum_{f\in F} x_{fr}^k \le 1,\ f\in 航线r上的航班,\ r\in R\end{aligned} \tag{4.6}$$

iv. 飞机最小间隔约束：每架飞机前一航班到达时间与后一航班起飞时间的最小间隔时间为 45 分钟，故有如下约束

$$tq_{f+1}^k \ge tg_f^k + t_c \tag{4.7}$$

综合上述公式，对问题一我们可以建立如下的单机型航班总延误时间成本优化模型：

$$\min Z = \sum_{k\in K}\sum_{f\in F} d_f^k x_f^k + \sum_{f\in F} c_f y_f$$

$$s.t.\begin{cases}\sum_{k\in K} x_f^k + y_f = 1,\ f\in F\\ \sum_{k\in K}\sum_{f\in F} F_a x_{fa}^k \le K_a, a\in A\\ \sum_{k\in K} x_{fr}^k \le 1,\ f\in 航线r上的航班,\ r\in R\\ \sum_{f\in F} x_{fr}^k \le 1,\ f\in 航线r上的航班,\ r\in R\\ tq_{f+1}^k \ge tg_f^k + t_c\\ x_f^k = 1,0,\ f\in F, k\in K\\ y_f = 0,1,\ f\in F\end{cases} \tag{4.8}$$

4.2 模型的求解

我们建立的上述航班排班优化模型是一个动态规划问题，且含有多个约束条件[1-5]，因此我们提出基于贪婪算法的启发式求解方法来求解该问题。贪婪算法是指在对优化问题求解时，总是采取在当前看来是最好的策略，即不考虑整体最优解，而寻找某种意义上的局部最优解。贪婪算法可以大大提高运算速度，加快收敛，为寻求最优解提供了方便[6]。

我们首先建立了时间轴，以时间线来遍历并提取出所有航班信息及其对应的安排，以此来记录飞机状态，如起飞、航行中、降落等，为之后的工作做好铺垫。根据题目中所给信息，考虑到 OVS 机场因暴风雪于 2016 年 4 月 22 日 18:00～21:00 关闭，则原计划在此时间段内于 OVS 机场起降的航班将会受到影响。为了尽量避免航空公司的损失，我们希望原定航班能够在飞行条件恢复后尽快复航。在航班调整中，我们主要考虑让目标函数中取消航班延误成本为零（即尽量不取消航班），并在约束条件的前提下给每个航班做出指派飞机的策略。具体的基于贪婪算法的启发式求解方法步骤如下：

step1. 我们将原计划于 4 月 22 日 18:00～21:00 期间从 OVS 机场起飞的航班一律延误至 21:00 排队起飞，原计划于该时间内到 OVS 机场降落的航班一律延误至 21:00 之后排队降落（由于 OVS 机场限制，每 5 分钟内最多起飞 5 架航班+降落 5 架航班）。

Step2. 找出起降不受暴风雪影响的航班，该部分航班仍按照原计划执行任务。

Step3. 对于受暴风雪影响的航班，我们通过建立 $airplane\_dic$ 字典来记录当前决策时刻下每个机场中的可用于执行航班任务的飞机。

Step4. 将航班指派给可用于执行航班任务的飞机上，若有多架飞机可以指派，则我们优先指派在当前决策时刻能够立刻值机的飞机，

Step5. 若没有可指派的飞机，则将进一步选择等待时间较短的飞机。

Step6. 遍历时间轴，完成所有航班的指派，并将结果输出。

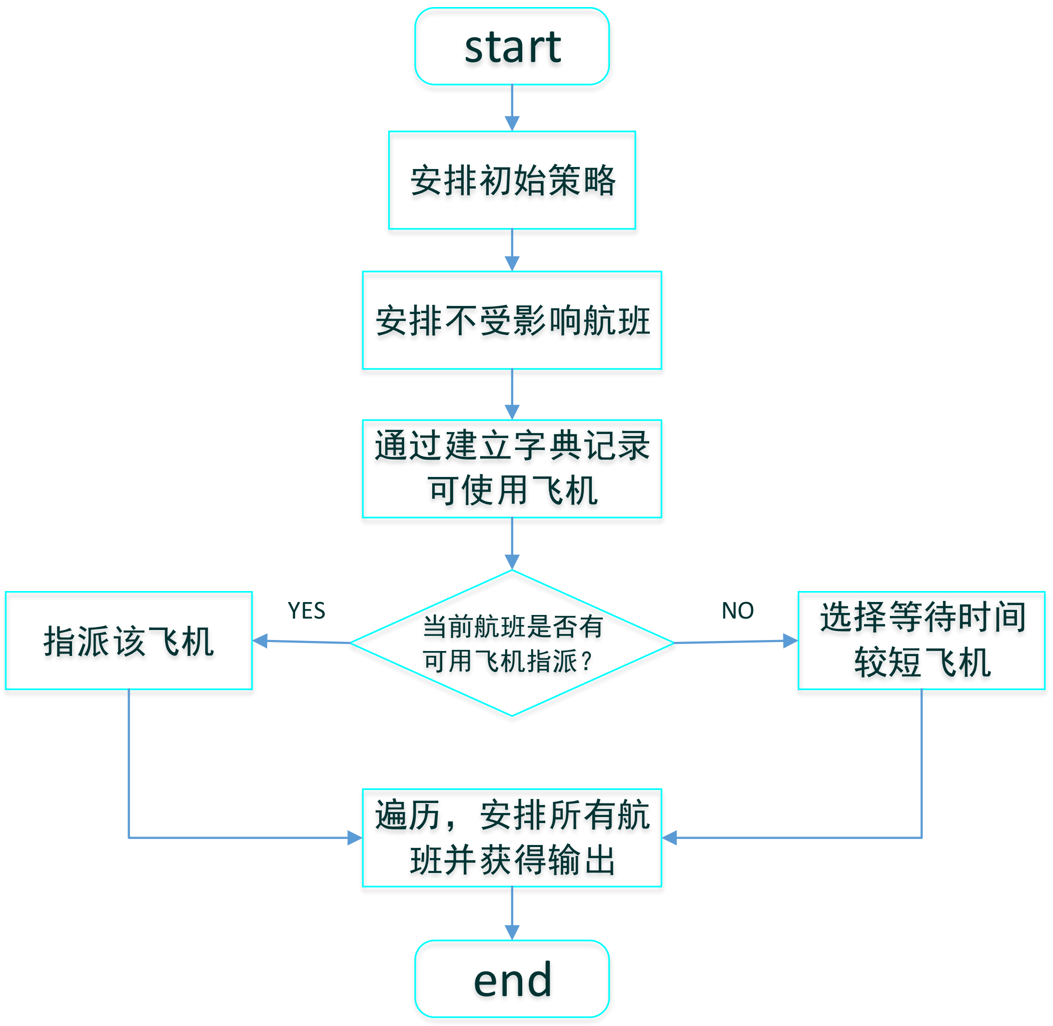
start
安排初始策略
安排不受影响航班
通过建立字典记录可使用飞机
当前航班是否有可用飞机指派？
YES
指派该飞机
NO
选择等待时间较短飞机
遍历，安排所有航班并获得输出
end

图 1 流程示意图

本题中最终航班安排共涉及 97 架航班，其中 14 架延误，航班准点率为 85.57%，航班总延误时长为 18 小时 59 分（部分结果见表 1，完整结果见附件）。

表 1 问题一航班安排（部分）

航班唯一编号	起飞时间	新起飞时间	到达时间	新到达时间	起飞机场	到达机场	飞机型号	新飞机型号	飞机尾号	新飞机尾号	航班延误时间	航班状态
174773328	2016/4/22 13:10	2016/4/22 13:10	2016/4/22 14:37	2016/4/22 14:37	QSM	OVS	9	9	23098	23098	0:00:00	as plan
174773332	2016/4/22 21:10	2016/4/22 21:10	2016/4/22 22:55	2016/4/22 22:55	OVS	HRA	9	9	42098	44098	0:00:00	as plan (substitute)
174773380	2016/4/22 15:10	2016/4/22 18:09	2016/4/22 18:01	2016/4/22 21:00	MJT	OVS	9	9	64098	64098	2:59:00	delayed
…	…	…	…	…	…	…	…	…	…	…	…	…
174773957	2016/4/22 17:25	2016/4/22 19:23	2016/4/22 19:02	2016/4/22 21:00	WTR	OVS	9	9	14098	14098	1:58:00	delayed
174773996	2016/4/22 11:46	2016/4/22 11:46	2016/4/22 14:28	2016/4/22 14:28	GAZ	OVS	9	9	51098	51098	0:00:00	as plan
174774048	2016/4/22 19:45	2016/4/22 21:00	2016/4/22 21:15	2016/4/22 22:30	OVS	QSM	9	9	85098	46098	1:15:00	delayed
…	…	…	…	…	…	…	…	…	…	…	…	…
174778566	2016/4/23 5:10	2016/4/23 5:10	2016/4/23 7:10	2016/4/23 7:10	OVS	HRA	9	9	23098	23098	0:00:00	as plan
174778580	2016/4/23 11:50	2016/4/23 11:50	2016/4/23 15:25	2016/4/23 15:25	OVS	GKS	9	9	14098	14098	0:00:00	as plan

注：as plan 表示按原计划时间由原计划飞机值机；as plan(substitute)表示按原计划时间由其他飞机替飞；Delayed 表示延误。

5、问题二模型建立与求解

5.1、问题描述及模型建立

问题二要求对所有机型受到机场 OVS 关闭影响的所有航班进行重新规划，使得原计划航班尽量不被取消，同时所有航班总体延误时间最短。问题二的目标函数同样为航班总体延误时间，但在问题一的基础上，问题二允许飞机置换在不同机型的飞机间发生，且有置换成本，这就需要我们在重新规划航班计划时，比较使用同机型飞机置换的延误成本和使用不同机型飞机置换的综合成本（延误+置换），并寻求起飞时间的最优解。

针对问题二，我们考虑将指派同机型飞机的延误成本与指派不同机型飞机的延误成本分开，因此目标函数航班总体延误时间分解为三部分：第一部分为被执行的指派同机型飞机的所有航班延误时间成本，第二部分为被执行的指派不同机型飞机的所有航班延误时间成本，第三部分为被取消的所有航班的延误时间成本。具体构建过程如下：

（1）目标函数建立

基于问题一的符号说明，我们引入以下符号定义：$f(k)$ 为航班 f 原本指派飞机同机型的飞机集合，k 为其中的某架飞机；$f(s)$ 为航班 f 原本指派飞机不同机型的飞机集合，s 为其中的某架飞机；d_f^k 为用同机型飞机 k 执行航班 f 的航班延误成本，d_f^s 为用不同机型飞机 s 执行航班 f 的航班延误成本，d_s 为用不同机型飞机 s 执行航班 f 的额外延误成本，依据题目设为 30min。

根据航班被同机型或不同机型飞机执行与否定义决策变量：

$$x_f^k=\begin{cases}1,\text{如果同机型飞机}k\text{执行航班}f\\0,\ \text{其他}\end{cases} \tag{5.1}$$

$$x_f^s=\begin{cases}1,\text{如果不同机型飞机}s\text{执行航班}f\\0,\ \text{其他}\end{cases} \tag{5.2}$$

$$y_f=\begin{cases}1,\text{如果航班}f\text{被取消}\\0,\ \text{其他}\end{cases} \tag{5.3}$$

同样可建立如下目标函数

$$\min Z=\sum_{k\in f(k)}\sum_{f\in F}d_f^k x_f^k+\sum_{s\in f(s)}\sum_{f\in F}(d_f^s+d_s)x_f^s+\sum_{f\in F}c_f y_f \tag{5.4}$$

其中第一项表示用同机型飞机的航班延误时间成本，第二项表示用不同机型飞机的航班延误时间成本，第三项表示被取消航班的延误时间成本。因此若要使目标函数最小，则需减少不同机型飞机置换以及尽量避免航班取消。

（2）约束条件

i. 航班覆盖约束：无论指派同机型还是不同机型飞机，每个航班只有被取消和被执行两种状态，故改进约束如下

$$\sum_{k\in f(k)}\sum_{s\in f(s)}(x_f^k+x_f^s)+y_f=1,\ f\in F \tag{5.5}$$

ii.　飞机平衡约束：某个机场执行飞行航班任务的所有机型飞机数量不能超过停留在该机场可供指派的飞机数量，故改进约束如下

$$\sum_{k\in f(k)}\sum_{s\in f(s)}\sum_{f\in F}F_a(x_{fa}^k+x_{fa}^s)\le K_a, a\in A \tag{5.6}$$

iii.　飞机指派约束：无论同机型还是不同机型，每架飞机最多只能执行一个航班，每个航班也最多指派一架飞机，故改进约束如下

$$\begin{aligned}&\sum_{f\in F}x_{fr}^k\le 1 or \sum_{f\in F}x_{fr}^s\le 1,\ f\in \text{航线}r\text{上的航班}, r\in R\\&\sum_{k\in f(k)}\sum_{s\in f(s)}(x_{fr}^k+x_{fr}^s)\le 1,\ f\in \text{航线}r\text{上的航班}, r\in R\end{aligned} \tag{5.7}$$

iv.　飞机最小间隔约束：无论同机型还是不同机型，每架飞机前一航班到达时间与后一航班起飞时间的最小间隔时间为 45 分钟，故改进约束如下

$$tq_{f+1}^k\ge tg_f^k+t_c or tq_{f+1}^s\ge tg_f^s+t_c \tag{5.8}$$

综合上述公式，对问题二我们可以建立如下的多机型航班总延误时间优化模型：

$$\min Z=\sum_{k\in f(k)}\sum_{f\in F}d_f^k x_f^k+\sum_{s\in f(s)}\sum_{f\in F}(d_f^s+d_s)x_f^s+\sum_{f\in F}c_f y_f$$

$$s.t.\begin{cases}\sum_{k\in f(k)}\sum_{s\in f(s)}(x_f^k+x_f^s)+y_f=1,\ f\in F\\ \sum_{k\in f(k)}\sum_{s\in f(s)}\sum_{f\in F}F_a(x_{fa}^k+x_{fa}^s)\le K_a, a\in A\\ \sum_{f\in F}x_{fr}^k\le 1 or \sum_{f\in F}x_{fr}^s\le 1,\ f\in \text{航线}r\text{上的航班}, r\in R\\ \sum_{k\in f(k)}\sum_{s\in f(s)}(x_{fr}^k+x_{fr}^s)\le 1,\ f\in \text{航线}r\text{上的航班}, r\in R\\ tq_{f+1}^k\ge tg_f^k+t_c or tq_{f+1}^s\ge tg_f^s+t_c\\ x_f^k=1,0,\ f\in F, k\in f(k)\\ x_f^s=1,0,\ f\in F, s\in f(s)\\ y_f=0,1,\ f\in F\end{cases} \tag{5.9}$$

5.2 模型的求解

基于第一问的求解方法，我们仍通过建立 $airplane\_dic$ 字典来记录当前决策时刻下每个机场中的可用于执行航班任务的飞机并优先指派在当前决策时刻能够立刻值飞的飞机，否则将进一步选择等待时间较短的飞机。但不同的是，本问涉及到所有机型，且要求由不同机型飞机置换时将额外产生 30min 的延误，因此在选择值机飞机时需考虑飞机型号的影响，即减小目标函数的第二项：不同机型执行航班的延误成本。具体的选择值机飞机的改进策略如下：

Step 1　我们首先设原计划执行下一航班任务的飞机即将到达机场的时间为 landing_time（注：为了操作方便，$landing\_time$、$landing\_time_{\text{同型}}$、

$landing_time_{不同型}$均表示飞机实际到达时间后+45min，即其下一时间点即可执行任务，下同），设即将值机的航班起飞时间为$target_time$，若有$landing_time - target_time \le 0$，则说明在航班起飞时原计划飞机可按计划值机。当然，由于同型机替换并不会造成额外延误，因此也可以选择置换为同型的其他飞机。反之，若$landing_time - target_time > 0$，则说明该航班在当前决策时原计划飞机不能按计划执行任务。

Step 2 当原计划飞机不能按计划执行任务时，对其他可用飞机的优先级安排如下：

i. 原计划中航班唯一编号对应的飞机（$+landing\_time - target\_time$）

ii. 原计划中航班唯一编号对应飞机的某同一型号飞机（$+landing\_time_{同型} - target\_time$）

iii. 原计划中航班唯一编号对应飞机的某不同型号飞机（$+landing\_time_{不同型} - target\_time + 30min$）

注： 当$landing_time_{同型} - target_time < 0$或$landing_time_{不同型} - target_time < 0$时，记为 0。

Step 3 将$landing_time - target_time$、$landing_time_{同型} - target_time$与$landing_time_{不同型} - target_time + 30min$的值进行比较，结合 step 2 优先级并选择值较小且包含于当前$airplane_dic$字典的飞机作为最终值机方案。

基于上述选择值机的策略遍历时间轴，完成对所有航班的指派，可得最后排班结果。部分见表 2，完整结果见附件。本题中航班安排共涉及 749 架航班，其中 112 架延误，航班准点率为 85.05%，航班总延误时长为 222 小时 49 分。

表 2 问题二航班安排（部分）

航班唯一编号	起飞时间	新起飞时间	到达时间	新到达时间	起飞机场	到达机场	飞机型号	新飞机型号	飞机尾号	新飞机尾号	航班延误时间	航班状态
174773318	2016/4/22 16:25	2016/4/22 18:54	2016/4/22 19:01	2016/4/22 21:30	PIS	OVS	32A	32A	UPBQV	UPBQV	2:29:00	delayed
174773320	2016/4/22 22:00	2016/4/22 22:30	2016/4/23 0:40	2016/4/23 1:10	OVS	PIS	32A	32A	AJBPV	LNBPV	0:30:00	delayed
174773324	2016/4/22 14:45	2016/4/22 14:45	2016/4/22 17:07	2016/4/22 17:07	VRM	OVS	320	320	PZBPV	PZBPV	0:00:00	as plan
…	…	…	…	…	…	…	…	…	…	…	…	…
174774470	2016/4/22 10:01	2016/4/22 10:01	2016/4/22 14:12	2016/4/22 14:12	OVS	VLT	321	321	OWBPV	OWBPV	0:00:00	as plan
174774472	2016/4/22 23:50	2016/4/22 23:50	2016/4/23 4:00	2016/4/23 4:00	BVO	OVS	320	320	YABQV	XABQV	0:00:00	as plan (substitute)
174774474	2016/4/22 22:30	2016/4/22 22:30	2016/4/23 1:05	2016/4/23 1:05	EIV	OVS	320	320	UQBPV	UQBPV	0:00:00	as plan
…	…	…	…	…	…	…	…	…	…	…	…	…
174778592	2016/4/23 6:25	2016/4/23 6:25	2016/4/23 7:40	2016/4/23 7:40	DEL	OVS	321	321	PUBPV	PUBPV	0:00:00	as plan
177411969	2016/4/22 13:20	2016/4/22 13:20	2016/4/22 15:28	2016/4/22 15:28	OVS	WAW	321	321	WRBPV	WRBPV	0:00:00	as plan

注：as plan 表示按原计划时间由原计划飞机值机；as plan(substitute)表示按原计划时间由其他飞机替飞；Delayed 表示延误。

6、问题三模型建立与求解

6.1、问题描述及模型建立

问题三要求进一步考虑飞机的载客量，在上一问不同机型飞机间有置换成本的基础上加入不能登机的旅客成本，并通过重新规划所有机型受影响的航班，使得旅客总体延误时间最短。问题三中每个航班均为满载，因此每个航班旅客的延误时间等价于航班的延误时间加上不能登机旅客的延误时间。基于这一点，我们可以将目标函数旅客总延误时间构造为每个航班每个旅客延误时间的求和，通过最优化目标函数寻求航班计划最优解。

具体来说，我们考虑三种航班计划：指派同机型飞机，指派载客量大于原航班机型的飞机以及指派载客量小于原航班机型的飞机。指派同机型飞机只有航班延误成本，指派载客量大于原航班机型的飞机的延误成本包括航班延误和飞机置换成本，指派载客量小于原航班机型的飞机的延误成本包括航班延误，飞机置换成本以及不能登机旅客成本。具体构建过程如下：

（1）目标函数建立

基于问题一、二的符号说明，我们引入如下符号定义：$f(k)$，$f(m)$，$f(l)$ 分别为航班 f 指派等于原机型载客量（同机型），大于原机型载客量和小于原机型载客量的飞机集合，k，m，l 分别为其中的某架飞机。v_k，v_l 分别为同机型飞机 k 和载客量小于原机型的飞机 l 的载客量，d_f^k，d_f^m，d_f^l 分别为用同机型飞机 k，载客量大于原机型的飞机 m 和载客量小于原机型的飞机 l 执行航班 f 的航班延误成本，d_m，d_l 为用不同机型飞机执行航班 f 的额外延误成本，$d_s = d_l = 30\min$，d_{kl} 为未登机旅客的延误成本，$d_{kl} = 2h$。

根据航班被同机型，载客量大于原机型或载客量小于原机型的飞机执行与否定义决策变量：

$$x_f^k = \begin{cases} 1, \text{如果同机型飞机}k\text{执行航班}f \\ 0,\ \text{其他} \end{cases} \tag{6.1}$$

$$x_f^m = \begin{cases} 1, \text{如果载客量大于原机型的飞机}m\text{执行航班}f \\ 0,\ \text{其他} \end{cases} \tag{6.2}$$

$$x_f^l = \begin{cases} 1, \text{如果载客量小于原机型的飞机}l\text{执行航班}f \\ 0,\ \text{其他} \end{cases} \tag{6.3}$$

$$y_f = \begin{cases} 1, \text{如果航班}f\text{被取消} \\ 0,\ \text{其他} \end{cases} \tag{6.4}$$

目标函数旅客总延误时间成本可建立如下

$$\min Z=\sum_{k\in f(k)}\sum_{f\in F}v_k d_f^k x_f^k+\sum_{m\in f(m)}\sum_{f\in F}v_k(d_f^m+d_m)x_f^m+ \sum_{l\in f(l)}\sum_{f\in F}(v_l(d_f^l+d_l)+(v_k-v_l)d_{kl})x_f^l+\sum_{f\in F}v_k c_f y_f \tag{6.5}$$

其中第一项为用同机型飞机执行的航班上旅客的总延误，第二项为用载客量大于原机型飞机执行的航班上旅客的总延误，第三项为用载客量小于原机型飞机执行的航班上旅客的总延误，第四项为航班取消时旅客的总延误。

（2）约束条件

基于问题二的航班覆盖约束，飞机平衡约束，飞机指派约束和飞机最小间隔约束，将不同机型飞机约束分为载客量大于原机型和载客量小于原机型飞机约束，可改进约束如下

i. 航班覆盖约束：

$$\sum_{k\in f(k)}\sum_{m\in f(m)}\sum_{l\in f(l)}(x_f^k+x_f^m+x_f^l)+y_f=1,\ f\in F \tag{6.6}$$

ii. 飞机平衡约束：

$$\sum_{k\in f(k)}\sum_{m\in f(m)}\sum_{l\in f(l)}\sum_{f\in F}F_a(x_{fa}^k+x_{fa}^m+x_{fa}^l)\le K_a, a\in A \tag{6.7}$$

iii. 飞机指派约束：

$$\begin{gathered}\sum_{f\in F}x_{fr}^k\le 1 or\sum_{f\in F}x_{fr}^m\le 1 or\sum_{f\in F}x_{fr}^l\le 1, f\in 航线r上的航班, r\in R\\ \sum_{k\in f(k)}\sum_{m\in f(m)}\sum_{l\in f(l)}(x_{fr}^k+x_{fr}^m+x_{fr}^l)\le 1,\ f\in 航线r上的航班, r\in R\end{gathered} \tag{6.8}$$

iv. 飞机最小间隔约束：

$$tq_{f+1}^k\ge tg_f^k+t_c ortq_{f+1}^m\ge tg_f^m+t_c ortq_{f+1}^l\ge tg_f^l+t_c \tag{6.9}$$

引入决策变量，并综合上述公式，对问题三我们可以建立如下的多机型旅客总延误时间优化模型：

$$\min Z=\sum_{k\in f(k)}\sum_{f\in F}v_k d_f^k x_f^k+\sum_{m\in f(m)}\sum_{f\in F}v_k(d_f^m+d_m)x_f^m+$$
$$\sum_{l\in f(l)}\sum_{f\in F}(v_l(d_f^l+d_l)+(v_k-v_l)d_{kl})x_f^l+\sum_{f\in F}v_k c_f y_f$$

$$s.t.\begin{cases}\sum_{k\in f(k)}\sum_{m\in f(m)}\sum_{l\in f(l)}(x_f^k+x_f^m+x_f^l)+y_f=1,\ f\in F\\ \sum_{k\in f(k)}\sum_{m\in f(m)}\sum_{l\in f(l)}\sum_{f\in F}F_a(x_{fa}^k+x_{fa}^m+x_{fa}^l)\le K_a,a\in A\\ \sum_{f\in F}x_{fr}^k\le 1or\sum_{f\in F}x_{fr}^m\le 1or\sum_{f\in F}x_{fr}^l\le 1,f\in\text{航线}r\text{上的航班},\ r\in R\\ \sum_{k\in f(k)}\sum_{m\in f(m)}\sum_{l\in f(l)}(x_{fr}^k+x_{fr}^m+x_{fr}^l)\le 1,\ f\in\text{航线}r\text{上的航班},\ r\in R\\ tq_{f+1}^k\ge tg_f^k+t_c ortq_{f+1}^m\ge tg_f^m+t_c ortq_{f+1}^l\ge tg_f^l+t_c\\ x_f^k=1,0,\ f\in F,k\in f(k)\\ x_f^m=1,0,\ f\in F,m\in f(m)\\ x_f^l=1,0,\ f\in F,l\in f(l)\\ y_f=0,1,\ f\in F\end{cases}\tag{6.10}$$

6.2 模型的求解

基于第三问在第二问的基础上进一步考虑了载客量的影响，且最终目的是要求旅客总延误时长最短。首先附件中已给出各机型的载客量并在题目中假设所有机型均为满载，则我们只需在第二问求解方法的基础上进一步加入考虑不能登机旅客延误成本的不同型飞机置换值机策略即可。具体的考虑旅客成本来选择值机飞机的改进策略如下：

Step 1 我们仍设原计划执行下一航班任务的飞机即将到达机场的时间为 $landing\_time$，设即将值机的航班起飞时间为 $target\_time$，若有

$landing\_time-target\_time\le 0$，则说明在航班起飞时原计划飞机可按计划值机。当然，由于同型机替换并不会造成额外延误，因此也可以选择置换为同型的其他飞机。反之，若 $landing\_time-target\_time>0$，则说明该航班在当前决策时原计划飞机不能按计划执行任务。

Step 2 当原计划飞机不能按计划执行任务时，对其他可用飞机的优先级安排如下：

i. 原计划中航班唯一编号对应的飞机（$+landing\_time-target\_time$）

ii. 原计划中航班唯一编号对应飞机的某同一型号飞机

（$+\ landing\ \_time_{\text{同型}}-target\_time$）

iii. 原计划中航班唯一编号对应飞机的某不同型号且载客量大于原飞机的飞机（$+landing\_time_{不同型}-target\_time\ +30min$）

iv. 原计划中航班唯一编号对应飞机的某不同型号且载客量小于原飞机的飞机（$+landing_time_{不同型}-target_time\ +30min+$载客量差$*2h$）

注： 当$landing\ _time_{同型}-target_time<0$或$landing_time_{不同型}-target_time<0$时，记为 0。

Step 3 将$landing_time-target_time$、$landing\ \_time_{同型}-target\_time$、$landing_time_{不同型}-target_time\ +30min$与$landing_time_{不同型}-target_time\ +30min+$载客量差$*2h$的值进行比较，结合 step 2 优先级并选择值较小且包含于当前$airplane_dic$字典的飞机作为最终值机决策。

基于上述选择值机的策略遍历时间轴，完成对所有航班的指派，可得最后排班结果。部分结果如表 2 所示，完整结果见附件。最终航班安排共涉及 749 架航班，其中 108 架延误，15 架取消，航班准点率为 83.58%。总延误时长为 211 小时 34 分，旅客总延误时长为 40171 小时 51 分。

表 3 问题三航班安排（部分）

航班唯一编号	起飞时间	新起飞时间	到达时间	新到达时间	起飞机场	到达机场	飞机型号	新飞机型号	飞机尾号	新飞机尾号	旅客延误时间	航班延误时间	航班状态
174773318	2016/4/22 16:25	2016/4/22 18:54	2016/4/22 19:01	2016/4/22 21:30	PIS	OVS	32A	32A	UPBQV	UPBQV	392:22:00	2:29:00	delayed
174773320	2016/4/22 22:00	2016/4/22 22:20	2016/4/23 0:40	2016/4/23 1:00	OVS	PIS	32A	32A	AJBPV	UPBQV	52:40:00	0:20:00	delayed
174773324	2016/4/22 14:45	2016/4/22 14:45	2016/4/22 17:07	2016/4/22 17:07	VRM	OVS	320	320	PZBPV	PZBPV	0:00:00	0:00:00	as plan
...	...	...	...	...	...	...	...	...	...	...	...	...	
174774470	2016/4/22 10:01	2016/4/22 10:01	2016/4/22 14:12	2016/4/22 14:12	OVS	VLT	321	321	OWBPV	OWBPV	0:00:00	0:00:00	as plan
174774472	2016/4/22 23:50				BVO	OVS	320	canceled	YABQV	canceled	280:00:00	canceled	canceled
174774474	2016/4/22 22:30	2016/4/22 22:30	2016/4/23 1:05	2016/4/23 1:05	EIV	OVS	320	320	UQBPV	UQBPV	0:00:00	0:00:00	as plan
...	...	...	...	...	...	...	...	...	...	...	...	...	
174778592	2016/4/23 6:25	2016/4/23 6:25	2016/4/23 7:40	2016/4/23 7:40	DEL	OVS	321	321	PUBPV	PUBPV	0:00:00	0:00:00	as plan
177411969	2016/4/22 13:20	2016/4/22 13:20	2016/4/22 15:28	2016/4/22 15:28	OVS	WAW	321	321	WRBPV	WRBPV	0:00:00	0:00:00	as plan

注：as plan 表示按原计划时间由原计划飞机值机；as plan(substitute)表示按原计划时间由其他飞机替飞；Delayed 表示延误。

7、问题四模型建立与求解

7.1 问题描述及模型建立

问题四要求在第二题的基础上引入每个旅客的航班与行程信息，同行旅客数量等，并且不考虑不同机型间飞机置换的成本，通过重新规划所有机型受影响的航班，使得旅客总体延误时间最短。由于本问中飞机实际载客量不固定，且考虑了乘客的连程航班、同行旅客人数及换乘时间间隔等因素，因此我们将目标函数构建为对每个乘客延误的求和，具体为对每个旅客乘坐的航班的实际到达时间与原计划到达时间的差的求和。具体构建过程如下：

（1）目标函数建立

基于问题一、二、三的符号说明，我们引入以下符号定义：I 为旅客集合，i 为其中某一旅客，T_f 为航班 f 原计划到达时间，p_f 为航班 f 被取消时每个乘客的延误成本，tq_f^i，tg_f^i 分别为旅客 i 搭乘航班 f 的起飞和降落时间，h_f^k，h_f^s 分别为用同机型飞机 k 执行航班 f 和用不同机型飞机 s 执行航班 f 时飞机的实际载客量。本题的决策变量构造与问题二相同，故不赘述。目标函数旅客总延误时间成本可建立如下

$$\min Z = \sum_{i\in I}\sum_{k\in f(k)}\sum_{s\in f(s)}\sum_{f\in F}[(tg_f^k - T_f)x_f^k + (tg_f^s - T_f)x_f^s + p_f y_f] \qquad (7.1)$$

其中中括号里第一项为旅客 i 乘坐由同机型飞机 k 执行的航班 f 的延误成本，第二项为旅客 i 乘坐由不同机型飞机 s 执行的航班 f 的延误成本，第三项为旅客 i 乘坐的航班 f 被取消的延误成本。

（2）约束条件

本题模型的航班覆盖约束，飞机平衡约束，飞机指派约束和飞机最小间隔约束与问题二相同，此外还引入了两个新的约束条件：

i. 旅客换乘最小间隔约束：每个旅客行程中的相连航班间最少需要 45min 间隔时间用于中转，故有约束如下

$$tq_{f+1}^i \ge tg_f^i + t_c \qquad (7.2)$$

ii. 飞机载客量约束：每个飞机的实际载客量不能超过该机型规定的载客量，故有约束如下

$$h_f^k \le v_k, or h_f^k \le v \qquad (7.3)$$

引入决策变量，并综合上述公式，对问题四我们可以建立如下的多机型旅客总延误时间优化模型：

$$\min Z=\sum_{i\in I}\sum_{k\in f(k)}\sum_{s\in f(s)}\sum_{f\in F}[(tg_f^k-T_f)x_f^k+(tg_f^s-T_f)x_f^s+p_f y_f]$$

$$s.t.\begin{cases}\sum_{k\in f(k)}\sum_{s\in f(s)}(x_f^k+x_f^s)+y_f=1,\ f\in F\\ \sum_{k\in f(k)}\sum_{s\in f(s)}\sum_{f\in F}F_a(x_{fa}^k+x_{fa}^s)\le K_a,a\in A\\ \sum_{f\in F}x_{fr}^k\le 1 or\sum_{f\in F}x_{fr}^s\le 1,\ f\in \text{航线}r\text{上的航班},\ r\in R\\ \sum_{k\in f(k)}\sum_{s\in f(s)}(x_{fr}^k+x_{fr}^s)\le 1,\ f\in \text{航线}r\text{上的航班},\ r\in R\\ tq_{f+1}^k\ge tg_f^k+t_c,ortq_{f+1}^s\ge tg_f^s+t_c\\ tq_{f+1}^i\ge tg_f^i+t_c\\ h_f^k\le v_k,orh_f^k\le v_s\\ x_f^k=1,0,\ f\in F,k\in f(k)\\ x_f^s=1,0,\ f\in F,s\in f(s)\\ y_f=0,1,\ f\in F\end{cases}\qquad (7.4)$$

7.2 模型的求解

第四问实际上是在第二问和第三问基础上的拓展，虽然本小题不需要考虑不同型飞机替换的成本，但由于考虑了乘客的连程航班、同行旅客人数及换乘时间间隔而增加了求解难度[7,8]。

首先，根据题目中的信息提示我们对航班号进行了筛选，分别提取出了 Paxinfo_real（注：即提纯后的 Paxinfo 附件数据，删去了 Schedules 附件中不存在的航班唯一编号）和 non_cost_flight（注：即为 Schedules 附件中存在而 Paxinfo 附件中不存在的航班唯一编号，该类航班被认为没有旅客信息，因此延误该类航班并不影响本题最终结果）。本题在求解过程中将 non_cost_flight 涉及的航班信息全部 cancell，为其他载客航班最大限度提供调度空间，尽量缩短旅客延误总时长。

其次，连程乘客需要 45 分钟的时间进行换乘，即到达 45min 后方可搭乘下一架航班。对于后程航班来说，往往有多个前序航班与之对应，也就是说，在连程航班环节中较后的航班，其乘客将会来自不同前序航班，且到达时间各异。我们设某后程航班 X 其前序航班共有 n 架，分别记为 $x1, x2,\dots,xn$，即其旅客将会分 n 批到达。若这 n 架航班均能在航班 X 起飞前顺利到达机场，则该航班可正点按计划起飞，否则将产生航班及旅客的延误成本，而这种情况下，航空公司应作出决定是否等待延误到达的旅客登机以及等待时长。

我们用 $delay\_passengers$ 来记录每一批乘客的旅客号（$delay\_passengers.id$）、同行旅客人数（$delay\_passengers.num$）、到达时间（$delay\_passengers.time$）等信息，同样用 $target\_time$ 来表示航班原计划起飞时

间。设$delay.time = delay_passengers.time - target_time$，将$delay.time$的数值从小到大排列将会得到一组旅客延误时长（注：负值记为 0）。记$delay.time(i)$为第i个前序航班的延误时间，若$max\{delay.time(i)\} \le 0$，则说明在航班起飞时原计划飞机可按计划值机。当然，由于本问题中，更换机型也不会造成额外成本，因此也可以选择置换为同型或载客量大于实际登机人数到其他飞机。反之，$max\{delay.time(i)\} > 0$，则说明该航班存在部分旅客不能按时登机，这种情况下我们需要进行进一步决策：

（1）等齐所有旅客，旅客延误成本$W1 = \sum Xi * max\{ delay.time(i)\}$；

（2）只等部分旅客，设可等待时长为$tolerance \in delay.time$，则旅客延误成本

$$W2 = \sum delay\_passengers.num(i)\ I_{\{delay.time(i) <= tolerance\}} * tolerance + \sum delay\_passengers.num(i)\ I_{\{delay.time(i) > tolerance\}} * 24h$$

可通过遍历 tolerance 得到只等部分延误旅客的最小延误成本$min\ W2$。（注：$I_{\{delay.time(i) > tolerance\}}$为示性函数，表示$delay.time(i) > tolerance$时取值为 1，否则为 0，下同）

（3）不等待旅客，旅客延误成本

$$W3 = \sum delay\_passengers.num(i)\ I_{\{delay.time(i) > 0\}} * 24h$$

将$W1$、$min\ W2$与$W3$的值进行比较可得到最佳等待策略。值得注意的是，由于航班延误最长时限为 5 小时，故，若旅客延误超过$5h$则认为无法登机，延误成本为同行旅客数$*24h$。

在值机安排上，考虑到实际航班并不全都满载，因此在选择不同型飞机置换时我们将策略进行了调整。具体的选择值机飞机的策略如下：

Step1 和第二、三问相同的，我们仍然设原计划执行下一航班任务的飞机即将到达机场的时间为$landing_time$，设即将值机的航班起飞时间为$target_time$，则，若有$landing_time - target_time \le 0$，则说明在航班起飞时原计划飞机可按计划值机。当然，由于机型替换并不会造成额外延误，因此也可以选择置换为同型的其他飞机或载客量大于实际登机人数的不同型飞机。反之，若$landing_time - target_time > 0$，则说明该航班在当前决策时原计划飞机不能按计划执行任务，

Step2 当原计划飞机不能按计划执行任务时，对其他可用飞机的优先级安排如下：

i.　原计划中航班唯一编号对应的飞机

（$+(landing\_time - target\_time)$*实际登机人数$+min\{W1, min\ W2, W3\}$）

ii.　原计划中航班唯一编号对应飞机的某同一型号飞机

（$+(landing\_time_{同型} - target\_time)$*实际登机人数$+min\{W1, min\ W2, W3\}$）

iii.　原计划中航班唯一编号对应飞机的某不同型号且载客量大于实际登机人数的飞机 ($+(landing\_time_{不同型} - target\_time)$*实际登机人数

$+min\{W1, min\ W2, W3\}$)

iv.　原计划中航班唯一编号对应飞机的某不同型号且载客量小于实际登机人数的飞机 ($+(landing\_time_{不同型} - target\_time)$*实际登机人数

$+min\{W1, min\ W2, W3\}$ + 载客量差$*24h$)

Step 3 将 $(landing\_time - target\_time)$*实际登机人数、

$(landing\_time_{同型} - target\_time)$*实际登机人数、

$(landing\_time_{不同型} - target\_time)$*实际登机人数与

$(landing\_time_{不同型} - target\_time)$*实际登机人数+载客量差*$24h$的值进行比较，

结合 step 2 选择值较小且包含于当前 $airplane\_dic$ 字典的飞机作为最终值机方案。

基于上述选择值机的策略遍历时间轴，完成对所有航班的指派，可得最后排班结果。部分结果如表 2 所示，完整结果见附件。最终航班安排共涉及 749 架航班，其中 96 架延误，118 架取消，航班准点率为 71.43%。总延误时长为 177 小时 58 分，旅客总延误时长为 37200 小时。

表 4 问题四航班安排（部分）

航班唯一编号	起飞时间	新起飞时间	到达时间	新到达时间	起飞机场	到达机场	飞机型号	新飞机型号	飞机尾号	新飞机尾号	旅客延误时间	航班延误时间	航班状态
174773318	2016/4/22 16:25	2016/4/22 18:44	2016/4/22 19:01	2016/4/22 21:20	PIS	OVS	32A	32A	UPBQV	HSBQV	0:00:00	2:19:00	delayed
174773320	2016/4/22 22:00	2016/4/22 22:30	2016/4/23 0:40	2016/4/23 1:10	OVS	PIS	32A	32A	AJBPV	AJBPV	0:00:00	0:30:00	delayed
174773324	2016/4/22 14:45	2016/4/22 14:45	2016/4/22 17:07	2016/4/22 17:07	VRM	OVS	320	32A	PZBPV	WPBQV	0:00:00	0:00:00	as plan (substitute)
…	…	…	…	…	…	…	…	…	…	…	…	…	
174774454	2016/4/22 18:10	2016/4/22 20:00	2016/4/22 20:00	2016/4/22 21:50	DGK	OVS	321	321	RQBPV	RQBPV	0:00:00	1:50:00	delayed
174774456					OCF	OVS	321	canceled	CDBPV	canceled	0:00:00		cancell
174774462	2016/4/22 21:45	2016/4/22 21:45	2016/4/23 1:25	2016/4/23 1:25	RHL	OVS	321	321	LTBPV	LTBPV	0:00:00	0:00:00	as plan
…	…	…	…	…	…	…	…	…	…	…	…	…	
174778592	2016/4/23 6:25	2016/4/23 6:25	2016/4/23 7:40	2016/4/23 7:40	DEL	OVS	321	321	PUBPV	PUBPV	0:00:00	0:00:00	as plan
177411969	2016/4/22 13:20	2016/4/22 13:20	2016/4/22 15:28	2016/4/22 15:28	OVS	WAW	321	321	WRBPV	WRBPV	0:00:00	0:00:00	as plan

注：as plan 表示按原计划时间由原计划飞机值机；as plan(substitute)表示按原计划时间由其他飞机替飞；Delayed 表示延误。

8、模型的总结与评价

前两问我们建立了基于航班延误时间最小的单机型和多机型航班恢复整数规划模型，并将实际条件作为模型约束，客观合理地抽象描述了具体的单机型与多机型航班恢复问题。在求解方面，我们结合贪婪算法，并引入了启发式策略改进，使得问题得到简化，运算速度得到提高，所得到的解为基于该贪婪算法下的模型局部最优解，也是满意解。

第三问是在前两问的基础上增加了不能登机旅客成本，因此我们将目标函数改为旅客总延误时间，并联系实际，建立了基于旅客总延误时间最小的多机型航班恢复整数规划模型。在求解方面，基于前两问提出的求解算法，我们结合模型实际，对选择值机飞机的策略做了进一步的优化改进，考虑了载客量的影响。

第四问在前几问基础上引入了具体旅客行程信息，因此在模型的建立过程中从考虑航班延误转变成考虑旅客每人的延误。基于这一点我们构造了多机型旅客恢复整数规划模型。在求解方面，我们对具体的策略结合实际做了更进一步的细分和优化改进，考虑了更复杂的飞机优先级。

本文所建立的单机型或多机型模型是对具体问题的合理描述，在生活实践应用中有很高的参照性；本文提出的基于贪婪算法的求解方法提高了求解模型的速度，但由于是按照当前最优的策略执行的，所得到的解只能是一种局部最优的可行解，不是全局最优解。

9、参考文献

[1] Jon D. Petersen, Gustaf Sölveling, John-Paul Clarke, Ellis L. Johnson, and Sergey Shebalov (2012). An Optimization Approach to Airline Integrated Recovery. Transportation Science, 46(4), 482-500

[2] J. M. Rosenberger, E. L. Johnson, G. L. Nemhauser(2003). Rerouting Aircraft for Airline Recovery. Transportation Science, 37(4), 408–421.

[3] Ahmad I. Z. Jarrah, Gang Yu, Nirup Krishnamurthy, and Ananda Rakshit (1993). A Decision Support Framework for Airline Flight Cancellations and Delays. Transportation Science, 27(3), 266-280.

[4] Stephen J. Maher (2015). Solving the Integrated Airline Recovery Problem Using Column-and-Row Generation. Transportation Science, 50(1), 216-237

[5] 乐美龙, 王婷婷, 吴聪聪. 多机型不正常航班恢复的时空网络模型[J]. 四川大学学报, 2013, 50(3), 477-483.

[6] 程望斌, 冯彩英, 曾毅, 罗百通, 向灿群. 航班计划的优化设计研究[J]. 湖南理工学院学报, 2016, 29(2), 38-42.

[7] 姜茂, 韩晓龙. 基于航班延误的飞机和乘客恢复模型[J]. 华中师范大学学报, 2015, 49(6), 876-882.

[8] 李晓岚, 乐美龙. 多目标飞机和旅客恢复分阶段启发式算法[J]. 计算机应用研究, 2014, 31(8), 2270-2274.

附录 Python 源程序

（1） 问题一

```
import pandas as pd
import copy

import datetime

class Flight(object):
    flight_id = 0
    takeoff_time = 0
    landing_time = 0
    plane = 0
    takeoff_airport = ''
    landing_airport = ''
    annotation = ''

    def __init__(self, flight_id, takeoff_time, landing_time, plane,
takeoff_airport, landing_airport):
        self.flight_id = flight_id
        self.takeoff_time = takeoff_time
        self.landing_time = landing_time
        self.plane = plane
        self.takeoff_airport = takeoff_airport
        self.landing_airport = landing_airport

class Event(object):
    flight_id = 0
    time = 0
    action = ''
    plane = ''
    airport = ''
    annotation = ''

    def __init__(self, flight_id, time, action, plane, airport):
        self.flight_id = flight_id
        self.time = time
        self.action = action
        self.plane = plane
        self.airport = airport

def get_plane_type(plane_id, df):
    return df[df['飞机尾号'] == plane_id].iloc[0, 1]

def get_future_landing_time_and_airport(log, plane_id):
    time = 0
    flight_id = 0
    for event in log:
        if event.plane == plane_id and event.action == 'takeoff':
            if(event.time > time):
                time = event.time
                flight_id = event.flight_id
    for event in log:
        if event.flight_id == flight_id and event.action == 'landing':
            return event.time, event.airport

def get_takeoff_event(log, flight_id):
    for event in log:
        if(event.flight_id == flight_id and event.action == 'takeoff'):
            return event

def get_landing_event(log, flight_id):
    for event in log:
        if(event.flight_id == flight_id and event.action == 'landing'):
            return event

def get_canceled_event(log, flight_id):
    for event in log:
        if(event.flight_id == flight_id and event.action == 'canceled'):
            return event

def get_duration(plan, flight_id):
    for event in plan:
        if(event.flight_id == flight_id and event.action == 'landing'):
            end = event.time
        if(event.flight_id == flight_id and event.action == 'takeoff'):
            start = event.time
    return end - start - 45 * 60

def if_flight_take_off(log, flight_id):
    for event in log:
        if event.flight_id == flight_id:
            return True
    return False

def from_unix_stamp(stamp):
    stamp -= 60*60*8
    time = datetime.datetime.fromtimestamp(stamp)
    return time
```

```
def key_time(event):
    return event.time

def key_flight_id(event):
    return event.flight_id

def adjust_plan(plan):
    dict = {}
    for event in plan:
        key = str(event.time) + event.action + event.airport
        dict.setdefault(key, 0)
        dict[key] += 1
        if(dict[key] > 5):
            event.time += 10 * 60
            if(event.action == 'takeoff'):
                landing_event = get_landing_event(plan, event.flight_id)
                landing_event.time += 10 * 60
                landing_event.annotation = 'delayed'
            if(event.action == 'landing'):
                take_off_event = get_takeoff_event(plan, event.flight_id)
                take_off_event.time += 10 * 60
                take_off_event.annotation = 'delayed'
            return False
    return True

def get_landing_soon(log, time, airport):
    result = []
    for event in log:
        if event.action == 'landing' and event.airport == airport and event.time >
time:
            result.append(event)
    return result

def to_csv(log, plan, filename):
    log.sort(key=key_flight_id)
    canceled_log = [event for event in log if event.action == 'canceled']
    log = [event for event in log if event.action != 'canceled']

    list = []
    length = len(log)
    i = -1
    lastid = -1
    while (i < length - 1):
          i += 1
          id = log[i].flight_id
          if(id == lastid): continue
          lastid = id
          landing_event = get_landing_event(log, id)
          take_off_event = get_takeoff_event(log, id)
          landing_event_in_plan = get_landing_event(plan, id)
          take_off_event_in_plan = get_takeoff_event(plan, id)
          tmp = []
          tmp.append(log[i].flight_id)  # id
          tmp.append(take_off_event_in_plan.time) #take off time in plan
          tmp.append(take_off_event.time) #take off time
          tmp.append(landing_event_in_plan.time) #landing_time_in_plan
          tmp.append(landing_event.time) #landing time
          tmp.append(take_off_event.airport)  # take off airport
          tmp.append(landing_event.airport)  # landing airport
          tmp.append(get_plane_type(take_off_event_in_plan.plane, df1)) #plane
type in plan
          tmp.append(get_plane_type(take_off_event.plane, df1)) # plane type
          tmp.append(take_off_event_in_plan.plane) # plane id in plan
          tmp.append(take_off_event.plane) # plane id
          tmp.append(take_off_event.time - take_off_event_in_plan.time)
          tmp.append(str(from_unix_stamp(take_off_event.time)))
          tmp.append(str(from_unix_stamp(landing_event.time)))
          tmp.append(take_off_event.annotation)
          list.append(tmp)

      length = len(canceled_log)
      i = 0
      while (i < length):
          id = canceled_log[i].flight_id
          landing_event_in_plan = get_landing_event(plan, id)
          take_off_event_in_plan = get_takeoff_event(plan, id)
          tmp = []
          tmp.append(canceled_log[i].flight_id)  # id
          tmp.append(take_off_event_in_plan.time)  # take off time in plan
          tmp.append('canceled')  # take off time
          tmp.append(landing_event_in_plan.time)  # landing_time_in_plan
          tmp.append('canceled')  # landing time
          tmp.append(take_off_event_in_plan.airport)  # take off airport
          tmp.append(landing_event_in_plan.airport)  # landing airport
          tmp.append(get_plane_type(take_off_event_in_plan.plane, df1))  #plane
type in plan
          tmp.append('canceled')  # plane type
```

```
            tmp.append(take_off_event_in_plan.plane)  # plane id in plan
            tmp.append('canceled')  # plane id
            tmp.append(24 * 60 * 60)
            tmp.append('canceled')
            tmp.append('canceled')
            tmp.append(canceled_log[i].annotation)
            list.append(tmp)
            i += 1

        result_df = pd.DataFrame(list)
        result_df.columns = [u'航班唯一编号',u'起飞时间',u'新起飞时间',u'到达时间
',u'新到达时间',u'起飞机场',u'到达机场',\
                             u'飞机型号',u'新飞机型号',u'飞机尾号',u'新飞机尾号
',u'延误时间',u'起飞时间（一般表述）',u'到达时间（一般表述）',u'注释']
        result_df.to_csv(filename, encoding='GBK', index=False)

    #开始
    df = pd.read_csv('./miniSchedules.csv', encoding='GBK')
    df['飞机尾号'] = df['飞机尾号'].astype('str')
    df1 = pd.read_csv('./miniAircrafts.csv', encoding='GBK')
    df1 = df1.iloc[:, 0:5]
    df1['飞机尾号'] = df1['飞机尾号'].astype('str')

    #初始化
    BLIZZARD_START = 1461348000
    BLIZZARD_END = 1461358800

    tmp = df.iloc[:, 3].values.tolist()
    tmp.extend(df.iloc[:, 4].values.tolist())
    AIRPORTS = list(set(tmp))

    #初始飞机位置
    airplane_dic = {}
    events = []
    for airport in AIRPORTS:
        airplane_dic[airport] = []

    for index, row in df1.iterrows():
        init_airport = row[4]
        plane_id = row[0]
        airplane_dic[init_airport].append(plane_id)


    #暴风雪的影响
```

```
for index, row in df.iterrows():
    #不能按时起飞
    if(row[1] > BLIZZARD_START and row[1] < BLIZZARD_END and row[3]=='OVS'):
        delay = BLIZZARD_END - row[1]
        df.iloc[index, 1] = row[1] + delay
        df.iloc[index, 2] = row[2] + delay
    #不能按时降落
    if(row[2] > BLIZZARD_START and row[2] < BLIZZARD_END and row[4]=='OVS'):
        delay = BLIZZARD_END - row[2]
        df.iloc[index, 1] = row[1] + delay
        df.iloc[index, 2] = row[2] + delay

#转化为对象
for index, row in df.iterrows():
    take_off_event = Event(row[0], row[1], 'takeoff', str(row[6]), row[3])
    landing_event = Event(row[0], row[2], 'landing', str(row[6]), row[4])
    events.append(take_off_event)
    events.append(landing_event)

#这样不必考虑 45 分钟维护的问题
for event in events:
    if(event.action == 'landing'):
        event.time += 45 * 60

#按时间排序
events.sort(key=key_time)
result = []

#开始处理
print("start")
index = -1
for event in events:
    if(index % 100 == 0):
        print(str(index) + '/' + str(len(events)))
    index += 1
    id = event.flight_id
    time = event.time
    airport = event.airport
    plane_in_plan = event.plane
    action = event.action
    duration = get_duration(events, id)
    plane_type = df1[df1['飞机尾号'] == plane_in_plan].iloc[0, 1]
    earlist = df1[df1['飞机尾号'] == plane_in_plan].iloc[0, 2]
    latest = df1[df1['飞机尾号'] == plane_in_plan].iloc[0, 3]
```

```
if event.action == 'landing' and if_flight_take_off(result, id):
    if ('-' + plane_in_plan) in airplane_dic[airport]:
        airplane_dic[airport].remove('-' + plane_in_plan)
    else:
        airplane_dic[airport].append(plane_in_plan)
    continue



substitute_list = []
tmp = get_future_landing_time_and_airport(result, plane_in_plan)
landing_soon = get_landing_soon(result, time, airport)

if (tmp == None):
    for p in landing_soon:
        substitute_list.append('*' + p.plane)
elif (tmp[1] != airport):
    for p in landing_soon:
        substitute_list.append('*' + p.plane)
else:
    future_landing_time = tmp[0]
    for p in landing_soon:
        if p.time < future_landing_time:
            substitute_list.append('*' + p.plane)

for plane in airplane_dic[airport]:
    if(plane[0] == '-'): continue
    _earlist = df1[df1['飞机尾号'] == plane].iloc[0, 2]
    _latest = df1[df1['飞机尾号'] == plane].iloc[0, 3]
    _plane_type = df1[df1['飞机尾号'] == plane].iloc[0, 1]
    if (time >= _earlist and time + duration <= _latest):
        substitute_list.append(plane)
#按计划起飞
if plane_in_plan in airplane_dic[airport]:
    event.annotation = 'as plan'
    landing_event = get_landing_event(events, id)
    landing_event.annotation = 'as plan'
    tmp = copy.deepcopy(event)
    result.append(tmp)
    tmp = copy.deepcopy(landing_event)
    result.append(tmp)
    airplane_dic[airport].remove(plane_in_plan)
#替飞
elif len(substitute_list) > 0:
    substitute = substitute_list[0]
```

```
            flag = 1 if substitute[0] == '*' else 0
            substitute = substitute[1:] if flag == 1 else substitute
            tmp = copy.deepcopy(event)
            tmp.plane = substitute
            tmp.annotation = 'substitute'
            result.append(tmp)
            landing_event = get_landing_event(events, id)
            tmp = copy.deepcopy(landing_event)
            tmp.plane = substitute
            tmp.annotation = 'substitute'
            result.append(tmp)
            if flag == 0:
                airplane_dic[airport].remove(substitute)
            else:
                airplane_dic[airport].append('-' + substitute)
            tmp = sorted(events[index + 1:], key=key_time)
            events[index + 1:] = tmp
        #延误或取消
        else:
            landing_time, landing_airport = 
get_future_landing_time_and_airport(result, plane_in_plan)
            destination_landing_time = landing_time + duration
            if(landing_time - time <= 60 * 60 * 5 and landing_airport == airport and 
not (landing_time > BLIZZARD_START and landing_time < BLIZZARD_END)\
               and destination_landing_time - 45 * 60 < latest and landing_time > 
earlist):
                tmp = copy.deepcopy(event)
                tmp.time = landing_time
                tmp.annotation = 'delayed'
                result.append(tmp)
                landing_event = get_landing_event(events, id)
                tmp = copy.deepcopy(landing_event)
                tmp.time += landing_time - time
                tmp.annotation = 'delayed'
                result.append(tmp)
                airplane_dic[airport].append('-' + plane_in_plan)
            else:
                result.append(Event(id, event.time, 'canceled', event.plane, ''))

  for event in result:
      if(event.action == 'landing'):
          event.time -= 45 * 60

  for event in events:
        if(event.action == 'landing'):
            event.time -= 45 * 60

    result.sort(key=key_time)

    #处理跑道冲突
    while(not adjust_plan(result)):
        pass

    result.sort(key=key_time)

    #输出
    for event in result:
        print('flight id ' + str(event.flight_id) + ' ' + str(event.action) + ' at
' + str(from_unix_stamp(event.time)) + '(unix stamp:' \
              + str(event.time) + ')' + ' in ' + str(event.airport) + '. plane: ' +
str(event.plane) + ', ' + event.annotation)

    df = pd.read_csv('./miniSchedules.csv', encoding='GBK')
    df['飞机尾号'] = df['飞机尾号'].astype('str')

    events = []

    for index, row in df.iterrows():
        take_off_event = Event(row[0], row[1], 'takeoff', str(row[6]), row[3])
        landing_event = Event(row[0], row[2], 'landing', str(row[6]), row[4])
        events.append(take_off_event)
        events.append(landing_event)

    to_csv(result, events, './result.csv')
```

（2） 问题二

```
    import pandas as pd
    import copy

    import datetime

    class Flight(object):
        flight_id = 0
        takeoff_time = 0
        landing_time = 0
        plane = 0
        takeoff_airport = ''
        landing_airport = ''
        annotation = ''
    def __init__(self, flight_id, takeoff_time, landing_time, plane,
takeoff_airport, landing_airport):
        self.flight_id = flight_id
        self.takeoff_time = takeoff_time
        self.landing_time = landing_time
        self.plane = plane
        self.takeoff_airport = takeoff_airport
        self.landing_airport = landing_airport

class Event(object):
    flight_id = 0
    time = 0
    action = ''
    plane = ''
    airport = ''
    annotation = ''

    def __init__(self, flight_id, time, action, plane, airport):
        self.flight_id = flight_id
        self.time = time
        self.action = action
        self.plane = plane
        self.airport = airport

def get_plane_type(plane_id, df):
    return df[df['飞机尾号'] == plane_id].iloc[0, 1]

def get_future_landing_time_and_airport(log, plane_id):
    time = 0
    flight_id = 0
    for event in log:
        if event.plane == plane_id and event.action == 'takeoff':
            if(event.time > time):
                time = event.time
                flight_id = event.flight_id
    for event in log:
        if event.flight_id == flight_id and event.action == 'landing':
            return event.time, event.airport

def get_takeoff_event(log, flight_id):
    for event in log:
        if(event.flight_id == flight_id and event.action == 'takeoff'):
            return event
```

```
def get_landing_event(log, flight_id):
    for event in log:
        if(event.flight_id == flight_id and event.action == 'landing'):
            return event

def get_canceled_event(log, flight_id):
    for event in log:
        if(event.flight_id == flight_id and event.action == 'canceled'):
            return event

def get_duration(plan, flight_id):
    for event in plan:
        if(event.flight_id == flight_id and event.action == 'landing'):
            end = event.time
        if(event.flight_id == flight_id and event.action == 'takeoff'):
            start = event.time
    return end - start - 45 * 60

def if_flight_take_off(log, flight_id):
    for event in log:
        if event.flight_id == flight_id:
            return True
    return False

def from_unix_stamp(stamp):
    stamp -= 60*60*8
    time = datetime.datetime.fromtimestamp(stamp)
    return time

def key_time(event):
    return event.time

def key_flight_id(event):
    return event.flight_id

def get_landing_soon(log, time, airport):
    result = []
    for event in log:
        if event.action == 'landing' and event.airport == airport and event.time > 
time and event.time - time <= 5 * 60 * 60:
            result.append(event)
    return result
```

```
def adjust_plan(plan):
    dict = {}
    for event in plan:
        key = str(event.time) + event.action + event.airport
        dict.setdefault(key, 0)
        dict[key] += 1
        if(dict[key] > 5):
            event.time += 10 * 60
            if(event.action == 'takeoff'):
                landing_event = get_landing_event(plan, event.flight_id)
                landing_event.time += 10 * 60
                landing_event.annotation = 'delayed'
            if(event.action == 'landing'):
                take_off_event = get_takeoff_event(plan, event.flight_id)
                take_off_event.time += 10 * 60
                take_off_event.annotation = 'delayed'
            return False
    return True

def to_csv(log, plan, filename):
    log.sort(key=key_flight_id)
    canceled_log = [event for event in log if event.action == 'canceled']
    log = [event for event in log if event.action != 'canceled']

    list = []
    length = len(log)
    i = -1
    lastid = -1
    while (i < length - 1):
        i += 1
        id = log[i].flight_id
        if(id == lastid): continue
        lastid = id
        landing_event = get_landing_event(log, id)
        take_off_event = get_takeoff_event(log, id)
        landing_event_in_plan = get_landing_event(plan, id)
        take_off_event_in_plan = get_takeoff_event(plan, id)
        tmp = []
        tmp.append(log[i].flight_id)  # id
        tmp.append(take_off_event_in_plan.time) #take off time in plan
        tmp.append(take_off_event.time) #take off time
        tmp.append(landing_event_in_plan.time) #landing_time_in_plan
        tmp.append(landing_event.time) #landing time
        tmp.append(take_off_event.airport)  # take off airport
```

```
            tmp.append(landing_event.airport)  # landing airport
            tmp.append(get_plane_type(take_off_event_in_plan.plane, df1)) #plane 
type in plan
            tmp.append(get_plane_type(take_off_event.plane, df1)) # plane type
            tmp.append(take_off_event_in_plan.plane) # plane id in plan
            tmp.append(take_off_event.plane) # plane id
            tmp.append(take_off_event.time - take_off_event_in_plan.time)
            tmp.append(str(from_unix_stamp(take_off_event.time)))
            tmp.append(str(from_unix_stamp(landing_event.time)))
            tmp.append(take_off_event.annotation)
            list.append(tmp)

        length = len(canceled_log)
        i = 0
        while (i < length):
            id = canceled_log[i].flight_id
            landing_event_in_plan = get_landing_event(plan, id)
            take_off_event_in_plan = get_takeoff_event(plan, id)
            tmp = []
            tmp.append(canceled_log[i].flight_id)  # id
            tmp.append(take_off_event_in_plan.time)  # take off time in plan
            tmp.append('canceled')  # take off time
            tmp.append(landing_event_in_plan.time)  # landing_time_in_plan
            tmp.append('canceled')  # landing time
            tmp.append(take_off_event_in_plan.airport)  # take off airport
            tmp.append(landing_event_in_plan.airport)  # landing airport
            tmp.append(get_plane_type(take_off_event_in_plan.plane, df1))  #plane 
type in plan
            tmp.append('canceled')  # plane type
            tmp.append(take_off_event_in_plan.plane)  # plane id in plan
            tmp.append('canceled')  # plane id
            tmp.append(24 * 60 * 60)
            tmp.append('canceled')
            tmp.append('canceled')
            tmp.append(canceled_log[i].annotation)
            list.append(tmp)
            i += 1

        result_df = pd.DataFrame(list)
        result_df.columns = [u'航班唯一编号',u'起飞时间',u'新起飞时间',u'到达时间
',u'新到达时间',u'起飞机场',u'到达机场',\
                             u'飞机型号',u'新飞机型号',u'飞机尾号',u'新飞机尾号
',u'延误时间',u'起飞时间（一般表述）',u'到达时间（一般表述）',u'注释']
        result_df.to_csv(filename, encoding='GBK', index=False)
```

```
#开始
df = pd.read_csv('./Schedules.csv', encoding='GBK')
df['飞机尾号'] = df['飞机尾号'].astype('str')
df1 = pd.read_csv('./Aircrafts.csv', encoding='GBK')
df1 = df1.iloc[:, 0:5]
df1['飞机尾号'] = df1['飞机尾号'].astype('str')

#初始化
BLIZZARD_START = 1461348000
BLIZZARD_END = 1461358800

tmp = df.iloc[:, 3].values.tolist()
tmp.extend(df.iloc[:, 4].values.tolist())
AIRPORTS = list(set(tmp))

#初始飞机位置
airplane_dic = {}
events = []
for airport in AIRPORTS:
    airplane_dic[airport] = []

for index, row in df1.iterrows():
    init_airport = row[4]
    plane_id = row[0]
    airplane_dic[init_airport].append(plane_id)


#暴风雪的影响
for index, row in df.iterrows():
    #不能按时起飞
    if(row[1] > BLIZZARD_START and row[1] < BLIZZARD_END and row[3]=='OVS'):
        delay = BLIZZARD_END - row[1]
        df.iloc[index, 1] = row[1] + delay
        df.iloc[index, 2] = row[2] + delay
    #不能按时降落
    if(row[2] > BLIZZARD_START and row[2] < BLIZZARD_END and row[4]=='OVS'):
        delay = BLIZZARD_END - row[2]
        df.iloc[index, 1] = row[1] + delay
        df.iloc[index, 2] = row[2] + delay

#转化为对象
for index, row in df.iterrows():
    take_off_event = Event(row[0], row[1], 'takeoff', str(row[6]), row[3])
```

```
        landing_event = Event(row[0], row[2], 'landing', str(row[6]), row[4])
        events.append(take_off_event)
        events.append(landing_event)

    #这样不必考虑 45 分钟维护的问题
    for event in events:
        if(event.action == 'landing'):
            event.time += 45 * 60

    #按时间排序
    events.sort(key=key_time)
    result = []

    #开始处理
    print("start")
    index = -1
    for event in events:
        if(index % 100 == 0):
            print(str(index) + '/' + str(len(events)))
        index += 1
        id = event.flight_id
        time = event.time
        airport = event.airport
        plane_in_plan = event.plane
        action = event.action
        duration = get_duration(events, id)
        plane_type = df1[df1['飞机尾号'] == plane_in_plan].iloc[0, 1]
        earlist = df1[df1['飞机尾号'] == plane_in_plan].iloc[0, 2]
        latest = df1[df1['飞机尾号'] == plane_in_plan].iloc[0, 3]
        if event.action == 'landing' and if_flight_take_off(result, id):
            if ('-' + plane_in_plan) in airplane_dic[airport]:
                airplane_dic[airport].remove('-' + plane_in_plan)
            else:
                airplane_dic[airport].append(plane_in_plan)
            continue

        tmp = get_future_landing_time_and_airport(result, plane_in_plan)
        landing_soon = get_landing_soon(result, event.time, airport)
        same_type_substitute_list = []
        diff_type_substitute_list = []
        if (tmp == None):
            for p in landing_soon:
                if get_plane_type(p.plane, df1) == plane_type:
                    same_type_substitute_list.append('*' + p.plane)
            else:
                diff_type_substitute_list.append('*' + p.plane)
    elif (tmp[1] != airport):
        for p in landing_soon:
            if get_plane_type(p.plane, df1) == plane_type:
                same_type_substitute_list.append('*' + p.plane)
            else:
                diff_type_substitute_list.append('*' + p.plane)
    else:
        future_landing_time = tmp[0]
        for p in landing_soon:
            if get_plane_type(p.plane, df1) == plane_type and p.time <
future_landing_time:
                same_type_substitute_list.append('*' + p.plane)
            if get_plane_type(p.plane, df1) != plane_type and p.time + 30 <
future_landing_time:
                diff_type_substitute_list.append('*' + p.plane)
    for plane in airplane_dic[airport]:
        if(plane[0] == '-'): continue
        _earlist = df1[df1['飞机尾号'] == plane].iloc[0, 2]
        _latest = df1[df1['飞机尾号'] == plane].iloc[0, 3]
        _plane_type = df1[df1['飞机尾号'] == plane].iloc[0, 1]
        if (time >= _earlist and time + duration <= _latest):
            if(_plane_type == plane_type):
                same_type_substitute_list.append(plane)
            else:
                diff_type_substitute_list.append(plane)
    if(len(same_type_substitute_list) > 0):
        substitute_list = same_type_substitute_list
    else:
        substitute_list = diff_type_substitute_list
    #按计划起飞
    if plane_in_plan in airplane_dic[airport]:
        event.annotation = 'as plan'
        landing_event = get_landing_event(events, id)
        landing_event.annotation = 'as plan'
        tmp = copy.deepcopy(event)
        result.append(tmp)
        tmp = copy.deepcopy(landing_event)
        result.append(tmp)
        airplane_dic[airport].remove(plane_in_plan)
    #替飞
    elif len(substitute_list) > 0:
        substitute = substitute_list[0]
```

```
            flag = 1 if substitute[0] == '*' else 0
            substitute = substitute[1:] if flag == 1 else substitute
            tmp = copy.deepcopy(event)
            tmp.plane = substitute
            tmp.annotation = 'substitute'
            result.append(tmp)
            landing_event = get_landing_event(events, id)
            tmp = copy.deepcopy(landing_event)
            tmp.plane = substitute
            tmp.annotation = 'substitute'
            result.append(tmp)
            if flag == 0:
                airplane_dic[airport].remove(substitute)
            else:
                airplane_dic[airport].append('-' + substitute)
            tmp = sorted(events[index + 1:], key=key_time)
            events[index + 1:] = tmp
        #延误或取消
        else:
            landing_time, landing_airport =
get_future_landing_time_and_airport(result, plane_in_plan)
            destination_landing_time = landing_time + duration
            if(landing_time - time <= 60 * 60 * 5 and landing_airport == airport and
not (landing_time > BLIZZARD_START and landing_time < BLIZZARD_END)\
               and destination_landing_time - 45 * 60 < latest and landing_time >
earlist):
                tmp = copy.deepcopy(event)
                tmp.time = landing_time
                tmp.annotation = 'delayed'
                result.append(tmp)
                landing_event = get_landing_event(events, id)
                tmp = copy.deepcopy(landing_event)
                tmp.time += landing_time - time
                tmp.annotation = 'delayed'
                result.append(tmp)
                airplane_dic[airport].append('-' + plane_in_plan)
            else:
                result.append(Event(id, event.time, 'canceled', event.plane, ''))

  for event in result:
      if(event.action == 'landing'):
          event.time -= 45 * 60

  for event in events:
        if(event.action == 'landing'):
            event.time -= 45 * 60

    result.sort(key=key_time)

    #处理跑道冲突
    while(not adjust_plan(result)):
        pass

    result.sort(key=key_time)

    #输出
    for event in result:
        print('flight id ' + str(event.flight_id) + ' ' + str(event.action) + ' at
' + str(from_unix_stamp(event.time)) + '(unix stamp:' \
              + str(event.time) + ')' + ' in ' + str(event.airport) + '. plane: ' +
str(event.plane) + ', ' + event.annotation)

    df = pd.read_csv('./Schedules.csv', encoding='GBK')
    df['飞机尾号'] = df['飞机尾号'].astype('str')

    events = []

    for index, row in df.iterrows():
        take_off_event = Event(row[0], row[1], 'takeoff', str(row[6]), row[3])
        landing_event = Event(row[0], row[2], 'landing', str(row[6]), row[4])
        events.append(take_off_event)
        events.append(landing_event)

    to_csv(result, events, './result2.csv')
```

（3）　问题三

```
    import pandas as pd
    import copy
    import numpy as np

    import datetime

    class Flight(object):
        flight_id = 0
        takeoff_time = 0
        landing_time = 0
        plane = 0
        takeoff_airport = ''
        landing_airport = ''
    annotation = ''

    def __init__(self, flight_id, takeoff_time, landing_time, plane,
takeoff_airport, landing_airport):
        self.flight_id = flight_id
        self.takeoff_time = takeoff_time
        self.landing_time = landing_time
        self.plane = plane
        self.takeoff_airport = takeoff_airport
        self.landing_airport = landing_airport

class Event(object):
    flight_id = 0
    time = 0
    action = ''
    plane = ''
    airport = ''
    annotation = ''

    def __init__(self, flight_id, time, action, plane, airport):
        self.flight_id = flight_id
        self.time = time
        self.action = action
        self.plane = plane
        self.airport = airport

def get_substitute(df, substitute_list, plane_in_plan):
    def comp(tuple):
        return tuple[1]
    _, in_plan_capacity = get_plane_capacity_tuple(df, plane_in_plan)
    candidates = []
    for candidate in substitute_list:
        candidates.append(get_plane_capacity_tuple(df, candidate))
    candidates.sort(key = comp)
    for i in range(len(candidates)):
        if(candidates[i][1] >= in_plan_capacity):
            return candidates[i][0]
    return candidates[-1][0]

def get_capacity(df, plane_id):
    return get_plane_capacity_tuple(df, plane_id)[1]

def get_plane_capacity_tuple(df, plane_id):
    return (plane_id, df[df['飞机尾号'] == plane_id].iloc[0, 5])
```

```
def get_plane_type(plane_id, df):
    return df[df['飞机尾号'] == plane_id].iloc[0, 1]

def get_future_landing_time_and_airport(log, plane_id):
    time = 0
    flight_id = 0
    for event in log:
        if event.plane == plane_id and event.action == 'takeoff':
            if(event.time > time):
                time = event.time
                flight_id = event.flight_id
    for event in log:
        if event.flight_id == flight_id and event.action == 'landing':
            return event.time, event.airport

def get_takeoff_event(log, flight_id):
    for event in log:
        if(event.flight_id == flight_id and event.action == 'takeoff'):
            return event

def get_landing_event(log, flight_id):
    for event in log:
        if(event.flight_id == flight_id and event.action == 'landing'):
            return event

def get_canceled_event(log, flight_id):
    for event in log:
        if(event.flight_id == flight_id and event.action == 'canceled'):
            return event

def get_duration(plan, flight_id):
    for event in plan:
        if(event.flight_id == flight_id and event.action == 'landing'):
            end = event.time
        if(event.flight_id == flight_id and event.action == 'takeoff'):
            start = event.time
    return end - start - 45 * 60

def if_flight_take_off(log, flight_id):
    for event in log:
        if event.flight_id == flight_id:
            return True
    return False
```

```
def from_unix_stamp(stamp):
    stamp -= 60*60*8
    time = datetime.datetime.fromtimestamp(stamp)
    return time

def key_time(event):
    return event.time

def key_flight_id(event):
    return event.flight_id

def adjust_plan(plan):
    dict = {}
    for event in plan:
        key = str(event.time) + event.action + event.airport
        dict.setdefault(key, 0)
        dict[key] += 1
        if(dict[key] > 5):
            event.time += 10 * 60
            if(event.action == 'takeoff'):
                landing_event = get_landing_event(plan, event.flight_id)
                landing_event.time += 10 * 60
                landing_event.annotation = 'delayed'
            if(event.action == 'landing'):
                take_off_event = get_takeoff_event(plan, event.flight_id)
                take_off_event.time += 10 * 60
                take_off_event.annotation = 'delayed'
            return False
    return True

def to_csv(log, plan, filename):
    log.sort(key=key_flight_id)
    canceled_log = [event for event in log if event.action == 'canceled']
    log = [event for event in log if event.action != 'canceled']

    list = []
    length = len(log)
    i = -1
    lastid = -1
    while (i < length - 1):
        i += 1
        id = log[i].flight_id
        if(id == lastid): continue
```

```
            lastid = id
            landing_event = get_landing_event(log, id)
            take_off_event = get_takeoff_event(log, id)
            landing_event_in_plan = get_landing_event(plan, id)
            take_off_event_in_plan = get_takeoff_event(plan, id)
            tmp = []
            tmp.append(log[i].flight_id)  # id
            tmp.append(take_off_event_in_plan.time) #take off time in plan
            tmp.append(take_off_event.time) #take off time
            tmp.append(landing_event_in_plan.time) #landing_time_in_plan
            tmp.append(landing_event.time) #landing time
            tmp.append(take_off_event.airport)  # take off airport
            tmp.append(landing_event.airport)  # landing airport
            tmp.append(get_plane_type(take_off_event_in_plan.plane, artifact_df))
#plane type in plan
            tmp.append(get_plane_type(take_off_event.plane, artifact_df)) # plane
type
            tmp.append(take_off_event_in_plan.plane) # plane id in plan
            tmp.append(take_off_event.plane) # plane id
            capacity_in_plan = get_capacity(artifact_df,
take_off_event_in_plan.plane)
            capacity = get_capacity(artifact_df, take_off_event.plane)
            loss = 0
            if(capacity < capacity_in_plan):
                loss += (capacity_in_plan - capacity) * 60 * 60 * 2
            loss += (take_off_event.time - take_off_event_in_plan.time) * capacity
            tmp.append(loss) # delay
            tmp.append(str(from_unix_stamp(take_off_event.time)))
            tmp.append(str(from_unix_stamp(landing_event.time)))
            tmp.append(take_off_event.annotation)
            list.append(tmp)

        length = len(canceled_log)
        i = 0
        while (i < length):
            id = canceled_log[i].flight_id
            landing_event_in_plan = get_landing_event(plan, id)
            take_off_event_in_plan = get_takeoff_event(plan, id)
            tmp = []
            tmp.append(canceled_log[i].flight_id)  # id
            tmp.append(take_off_event_in_plan.time)  # take off time in plan
            tmp.append('canceled')  # take off time
            tmp.append(landing_event_in_plan.time)  # landing_time_in_plan
            tmp.append('canceled')  # landing time
            tmp.append(take_off_event_in_plan.airport)  # take off airport
            tmp.append(landing_event_in_plan.airport)  # landing airport
            tmp.append(get_plane_type(take_off_event_in_plan.plane, artifact_df))
# plane type in plan
            tmp.append('canceled')  # plane type
            tmp.append(take_off_event_in_plan.plane)  # plane id in plan
            tmp.append('canceled')  # plane id
            tmp.append(2 * 60 * 60 * get_capacity(artifact_df,
take_off_event_in_plan.plane))
            tmp.append('canceled')
            tmp.append('canceled')
            tmp.append(canceled_log[i].annotation)
            list.append(tmp)
            i += 1

        result_df = pd.DataFrame(list)
        result_df.columns = [u'航班唯一编号',u'起飞时间',u'新起飞时间',u'到达时间
',u'新到达时间',u'起飞机场',u'到达机场',\
                             u'飞机型号',u'新飞机型号',u'飞机尾号',u'新飞机尾号
',u'延误时间',u'起飞时间（一般表述）',u'到达时间（一般表述）',u'注释']
        result_df.to_csv(filename, encoding='GBK', index=False)

    #开始
    schedule_df = pd.read_csv('./Schedules.csv', encoding='GBK')
    schedule_df['飞机尾号'] = schedule_df['飞机尾号'].astype('str')
    artifact_df = pd.read_csv('./Aircrafts.csv', encoding='GBK')
    artifact_df = artifact_df.iloc[:, 0:6]
    artifact_df['飞机尾号'] = artifact_df['飞机尾号'].astype('str')

    #初始化
    BLIZZARD_START = 1461348000
    BLIZZARD_END = 1461358800

    tmp = schedule_df.iloc[:, 3].values.tolist()
    tmp.extend(schedule_df.iloc[:, 4].values.tolist())
    AIRPORTS = list(set(tmp))

    #初始飞机位置
    airplane_dic = {}
    events = []
    for airport in AIRPORTS:
        airplane_dic[airport] = []

    for index, row in artifact_df.iterrows():
```

```
    init_airport = row[4]
    plane_id = row[0]
    airplane_dic[init_airport].append(plane_id)


#暴风雪的影响
for index, row in schedule_df.iterrows():
    #不能按时起飞
    if(row[1] > BLIZZARD_START and row[1] < BLIZZARD_END and row[3]=='OVS'):
        delay = BLIZZARD_END - row[1]
        schedule_df.iloc[index, 1] = row[1] + delay
        schedule_df.iloc[index, 2] = row[2] + delay
    #不能按时降落
    if(row[2] > BLIZZARD_START and row[2] < BLIZZARD_END and row[4]=='OVS'):
        delay = BLIZZARD_END - row[2]
        schedule_df.iloc[index, 1] = row[1] + delay
        schedule_df.iloc[index, 2] = row[2] + delay

#转化为对象
for index, row in schedule_df.iterrows():
    take_off_event = Event(row[0], row[1], 'takeoff', str(row[6]), row[3])
    landing_event = Event(row[0], row[2], 'landing', str(row[6]), row[4])
    events.append(take_off_event)
    events.append(landing_event)

#这样不必考虑 45 分钟维护的问题
for event in events:
    if(event.action == 'landing'):
        event.time += 45 * 60

#按时间排序
events.sort(key=key_time)
result = []

#开始处理
print("start")
index = -1
for event in events:
    if(index % 100 == 0):
        print(str(index) + '/' + str(len(events)))
    index += 1
    id = event.flight_id
    time = event.time
    airport = event.airport
```

```
        plane_in_plan = event.plane
        action = event.action
        duration = get_duration(events, id)
        plane_type = artifact_df[artifact_df['飞机尾号'] == plane_in_plan].iloc[0,
1]
        earlist = artifact_df[artifact_df['飞机尾号'] == plane_in_plan].iloc[0, 2]
        latest = artifact_df[artifact_df['飞机尾号'] == plane_in_plan].iloc[0, 3]
        if event.action == 'landing' and if_flight_take_off(result, id):
            if ('-' + plane_in_plan) in airplane_dic[airport]:
                airplane_dic[airport].remove('-' + plane_in_plan)
            else:
                airplane_dic[airport].append(plane_in_plan)
            continue
        same_type_substitute_list = []
        diff_type_substitute_list = []
        for plane in airplane_dic[airport]:
            if(plane[0] == '-'): continue
            _earlist = artifact_df[artifact_df['飞机尾号'] == plane].iloc[0, 2]
            _latest = artifact_df[artifact_df['飞机尾号'] == plane].iloc[0, 3]
            _plane_type = artifact_df[artifact_df['飞机尾号'] == plane].iloc[0, 1]
            if (time >= _earlist and time + duration <= _latest):
                if(_plane_type == plane_type):
                    same_type_substitute_list.append(plane)
                else:
                    diff_type_substitute_list.append(plane)
        if(len(same_type_substitute_list) > 0):
            substitute_list = same_type_substitute_list
        else:
            substitute_list = diff_type_substitute_list
        #按计划起飞
        if plane_in_plan in airplane_dic[airport]:
            event.annotation = 'as plan'
            landing_event = get_landing_event(events, id)
            landing_event.annotation = 'as plan'
            tmp = copy.deepcopy(event)
            result.append(tmp)
            tmp = copy.deepcopy(landing_event)
            result.append(tmp)
            airplane_dic[airport].remove(plane_in_plan)
        #替飞
        elif len(substitute_list) > 0:
            substitute = get_substitute(artifact_df, substitute_list,
plane_in_plan)
            flag = 1 if substitute[0] == '*' else 0
            substitute = substitute[1:] if flag == 1 else substitute
            tmp = copy.deepcopy(event)
            tmp.plane = substitute
            tmp.annotation = 'substitute'
            result.append(tmp)
            landing_event = get_landing_event(events, id)
            tmp = copy.deepcopy(landing_event)
            tmp.plane = substitute
            tmp.annotation = 'substitute'
            result.append(tmp)
            if flag == 0:
                airplane_dic[airport].remove(substitute)
            else:
                airplane_dic[airport].append('-' + substitute)
            tmp = sorted(events[index + 1:], key=key_time)
            events[index + 1:] = tmp
        #延误或取消
        else:
            landing_time, landing_airport =
get_future_landing_time_and_airport(result, plane_in_plan)
            destination_landing_time = landing_time + duration
            if(landing_time - time <= 60 * 60 * 2 and landing_airport == airport and
not (landing_time > BLIZZARD_START and landing_time < BLIZZARD_END)\
               and destination_landing_time - 45 * 60 < latest and landing_time >
earlist):
                tmp = copy.deepcopy(event)
                tmp.time = landing_time
                tmp.annotation = 'delayed'
                result.append(tmp)
                landing_event = get_landing_event(events, id)
                tmp = copy.deepcopy(landing_event)
                tmp.time += landing_time - time
                tmp.annotation = 'delayed'
                result.append(tmp)
                airplane_dic[airport].append('-' + plane_in_plan)
            else:
                result.append(Event(id, event.time, 'canceled', event.plane, ''))

  for event in result:
      if(event.action == 'landing'):
          event.time -= 45 * 60

  for event in events:
      if(event.action == 'landing'):
        event.time -= 45 * 60

result.sort(key=key_time)

#处理跑道冲突
while(not adjust_plan(result)):
    pass

result.sort(key=key_time)

#输出
for event in result:
    print('flight id ' + str(event.flight_id) + ' ' + str(event.action) + ' at
' + str(from_unix_stamp(event.time)) + '(unix stamp:' \
          + str(event.time) + ')' + ' in ' + str(event.airport) + '. plane: ' +
str(event.plane) + ', ' + event.annotation)

schedule_df = pd.read_csv('./Schedules.csv', encoding='GBK')
schedule_df['飞机尾号'] = schedule_df['飞机尾号'].astype('str')

events = []

for index, row in schedule_df.iterrows():
    take_off_event = Event(row[0], row[1], 'takeoff', str(row[6]), row[3])
    landing_event = Event(row[0], row[2], 'landing', str(row[6]), row[4])
    events.append(take_off_event)
    events.append(landing_event)

to_csv(result, events, './result3.csv')
```

（4） 问题四

```
import pandas as pd
import copy
import datetime
import sys
import numpy as np

aircrafts_df = pd.read_csv('./Aircrafts.csv', encoding='GBK')
aircrafts_df['飞机尾号'] = aircrafts_df['飞机尾号'].astype('str')
schedule_df = pd.read_csv('./Schedules.csv', encoding='GBK')
schedule_df['飞机尾号'] = schedule_df['飞机尾号'].astype('str')
paxinfo_df = pd.read_csv('./in_Paxinfo.csv', encoding='GBK')
noncost_df = pd.read_csv('./noncostflight.csv', encoding='GBK')

class Travel(object):
        flight_id = 0
        takeoff_time = 0
        landing_time = 0
        plane = ''
        passenger_id = 0
        num_passenger = 0

        def __init__(self, flight_id, takeoff_time, landing_time, plane,
passenger_id, num_passenger):
            self.flight_id = flight_id
            self.takeoff_time = takeoff_time
            self.landing_time = landing_time
            self.plane = plane
            self.passenger_id = passenger_id
            self.num_passenger = num_passenger

    class Flight(object):
        flight_id = 0
        takeoff_time = 0
        landing_time = 0
        plane = 0
        takeoff_airport = ''
        landing_airport = ''
        annotation = ''

        def __init__(self, flight_id, takeoff_time, landing_time, plane,
takeoff_airport, landing_airport):
            self.flight_id = flight_id
            self.takeoff_time = takeoff_time
            self.landing_time = landing_time
            self.plane = plane
            self.takeoff_airport = takeoff_airport
            self.landing_airport = landing_airport

    class Event(object):
        flight_id = 0
        time = 0
        action = ''
        plane = ''
        airport = ''
        annotation = ''
        passengers = []
        loss = 0
    def __init__(self, flight_id, time, action, plane, airport):
        self.flight_id = flight_id
        self.time = time
        self.action = action
        self.plane = plane
        self.airport = airport

    def add_annotation(self, annotation):
        if self.annotation == '':
            self.annotation = annotation

def get_takeoff_event(events, flight_id):
    for event in events:
        if(event.flight_id == flight_id and event.action == 'takeoff'):
            return event

def get_landing_event(events, flight_id):
    for event in events:
        if(event.flight_id == flight_id and event.action == 'landing'):
            return event

def get_takeoff_passengers(df, flight_id):
    return df[df['航班唯一编号'] == flight_id].iloc[:,0].values.tolist()

def get_landing_passenger(in_travel_passenger, flight_id):
    passengers = []
    for p in in_travel_passenger:
        if p.flight_id == flight_id:
            passengers.append(p)
    return passengers

def get_pre_flights_for_passenger(passenger_flight_dic, passengers):
    result = []
    for passenger in passengers:
        tmp = passenger_flight_dic.get(passenger)
        if tmp != None:
            result.append(tmp)
    return result

def get_delayed_passengers(in_travel_passengers, take_off_passengers):
    result = []
    for passenger in take_off_passengers:
        for in_travel_passenger in in_travel_passengers:
            if in_travel_passenger.flight_id == passenger:
```

```
            result.append(in_travel_passenger)
    return result

def get_plane_type(plane_id, df):
    try:
        tmp = df[df['飞机尾号'] == plane_id].iloc[0, 1]
    except:
        print(plane_id)
    return tmp

def key_time(event):
    return event.time

def key_flight_id(event):
    return event.flight_id

def get_duration(plan, flight_id):
    for event in plan:
        if(event.flight_id == flight_id and event.action == 'landing'):
            end = event.time
        if(event.flight_id == flight_id and event.action == 'takeoff'):
            start = event.time
    return end - start - 45 * 60

def get_future_landing_time_and_airport(log, plane_id):
    time = 0
    flight_id = 0
    for event in log:
        if event.plane == plane_id and event.action == 'takeoff':
            if(event.time > time):
                time = event.time
                flight_id = event.flight_id
    for event in log:
        if event.flight_id == flight_id and event.action == 'landing':
            return event.time, event.airport

def if_flight_take_off(log, flight_id):
    for event in log:
        if event.flight_id == flight_id:
            return True
    return False

def best_delay(delayed_passengers, total, time, time_in_plan):
    def comp(passenger):
            return passenger[0]
        tmp = 0
        for p in delayed_passengers:
            tmp += p.num_passenger
        num_on_time_passenger = total - tmp
        candidates = [(0, num_on_time_passenger)]
        for delayed_passenger in delayed_passengers:
            if not delayed_passenger.landing_time - time in candidates and
delayed_passenger.landing_time - time_in_plan < 5 * 60 * 60:
                candidates.append((delayed_passenger.landing_time - time,
delayed_passenger.num_passenger))
        candidates.sort(key= comp)
        loss = total * candidates[-1][0]
        result = candidates[-1][0]
        for i in range(len(candidates)):
            tmp_loss = np.sum(np.array(candidates)[:i,1]) * candidates[i][0] +
np.sum(np.array(candidates)[i:,1]) * 24 * 60 * 60
            if(tmp_loss < loss):
                loss = tmp_loss
                result = candidates[-1][0]
        return result, loss

    def get_num_total(df, flight_id):
        tmp = df[df['航班唯一编号'] == flight_id].iloc[:, 2]
        return np.sum(tmp)

    a = get_num_total(paxinfo_df, 174777728)

    def get_landing_soon(log, time, airport):
        result = []
        for event in log:
            if event.action == 'landing' and event.airport == airport and event.time >
time and event.time - time <= 5 * 60 * 60:
                result.append(event)
        return result

    def adjust_plan(plan):
        dict = {}
        for event in plan:
            key = str(event.time) + event.action + event.airport
            dict.setdefault(key, 0)
            dict[key] += 1
            if(dict[key] > 5):
                event.time += 10 * 60
            if(event.action == 'takeoff'):
                landing_event = get_landing_event(plan, event.flight_id)
                landing_event.time += 10 * 60
                landing_event.annotation = 'delayed'
            if(event.action == 'landing'):
                take_off_event = get_takeoff_event(plan, event.flight_id)
                take_off_event.time += 10 * 60
                take_off_event.annotation = 'delayed'
            return False
    return True

def from_unix_stamp(stamp):
    stamp -= 60*60*8
    time = datetime.datetime.fromtimestamp(stamp)
    return time

def to_csv(log, plan, filename):
    log.sort(key=key_flight_id)
    canceled_log = [event for event in log if event.action == 'canceled']
    log = [event for event in log if event.action != 'canceled']

    list = []
    length = len(log)
    i = -1
    lastid = -1
    while (i < length - 1):
        i += 1
        id = log[i].flight_id
        if(id == lastid): continue
        lastid = id
        landing_event = get_landing_event(log, id)
        take_off_event = get_takeoff_event(log, id)
        landing_event_in_plan = get_landing_event(plan, id)
        take_off_event_in_plan = get_takeoff_event(plan, id)
        tmp = []
        tmp.append(log[i].flight_id)  # id
        tmp.append(take_off_event_in_plan.time) #take off time in plan
        tmp.append(take_off_event.time) #take off time
        tmp.append(landing_event_in_plan.time) #landing_time_in_plan
        tmp.append(landing_event.time) #landing time
        tmp.append(take_off_event.airport)  # take off airport
        tmp.append(landing_event.airport)  # landing airport
        tmp.append(get_plane_type(take_off_event_in_plan.plane, aircrafts_df))
#plane type in plan
```

```
            tmp.append(get_plane_type(take_off_event.plane, aircrafts_df)) # plane
type
            tmp.append(take_off_event_in_plan.plane) # plane id in plan
            tmp.append(take_off_event.plane) # plane id
            tmp.append(take_off_event.loss)
            tmp.append(str(from_unix_stamp(take_off_event.time)))
            tmp.append(str(from_unix_stamp(landing_event.time)))
            tmp.append(take_off_event.annotation)
            list.append(tmp)

        length = len(canceled_log)
        i = 0
        while (i < length):
            id = canceled_log[i].flight_id
            landing_event_in_plan = get_landing_event(plan, id)
            take_off_event_in_plan = get_takeoff_event(plan, id)
            tmp = []
            tmp.append(canceled_log[i].flight_id)  # id
            tmp.append(take_off_event_in_plan.time)  # take off time in plan
            tmp.append('canceled')  # take off time
            tmp.append(landing_event_in_plan.time)  # landing_time_in_plan
            tmp.append('canceled')  # landing time
            tmp.append(take_off_event_in_plan.airport)  # take off airport
            tmp.append(landing_event_in_plan.airport)  # landing airport
            tmp.append(get_plane_type(take_off_event_in_plan.plane, aircrafts_df))
# plane type in plan
            tmp.append('canceled')  # plane type
            tmp.append(take_off_event_in_plan.plane)  # plane id in plan
            tmp.append('canceled')  # plane id
            tmp.append(canceled_log[i].loss)
            tmp.append('canceled')
            tmp.append('canceled')
            tmp.append(canceled_log[i].annotation)
            list.append(tmp)
            i += 1

        result_df = pd.DataFrame(list)
        result_df.columns = [u'航班唯一编号',u'起飞时间',u'新起飞时间',u'到达时间
',u'新到达时间',u'起飞机场',u'到达机场',\
                             u'飞机型号',u'新飞机型号',u'飞机尾号',u'新飞机尾号
',u'延误时间',u'起飞时间（一般表述）',u'到达时间（一般表述）',u'注释']
        result_df.to_csv(filename, encoding='GBK', index=False)

  #初始化
```

```
BLIZZARD_START = 1461348000
BLIZZARD_END = 1461358800

NONE_COST_FLIGHT = noncost_df['a'].values.tolist()

tmp = schedule_df.iloc[:, 3].values.tolist()
tmp.extend(schedule_df.iloc[:, 4].values.tolist())
AIRPORTS = list(set(tmp))

#初始飞机位置
airplane_dic = {}
events = []
for airport in AIRPORTS:
    airplane_dic[airport] = []

for index, row in aircrafts_df.iterrows():
    init_airport = row[4]
    plane_id = row[0]
    airplane_dic[init_airport].append(plane_id)

for index, row in schedule_df.iterrows():
    take_off_event = Event(row[0], row[1], 'takeoff', str(row[6]), row[3])
    landing_event = Event(row[0], row[2], 'landing', str(row[6]), row[4])
    events.append(take_off_event)
    events.append(landing_event)

plan_events = copy.deepcopy(events)

for event in events:
    if(event.time > BLIZZARD_START and event.time < BLIZZARD_END \
           and event.airport == 'OVS' and event.action == 'takeoff'):
        landing_event = get_landing_event(events, event.flight_id)
        delay = BLIZZARD_END - event.time
        event.time += delay
        event.annotation = 'delayed'
        landing_event.time += delay
        landing_event.annotation = 'delayed'
    if (event.time > BLIZZARD_START and event.time < BLIZZARD_END \
            and event.airport == 'OVS' and event.action == 'landing'):
        takeoff_event = get_takeoff_event(events, event.flight_id)
        delay = BLIZZARD_END - event.time
        event.time += delay
        event.annotation = 'delayed'
        takeoff_event.time += delay
```

```
            takeoff_event.annotation = 'delayed'

    for event in events:
        if(event.action == 'landing'):
            event.time += 45 * 60

    events.sort(key=key_time)
    result = []
    in_travel_passenger = []
    #旅客的上一版航班
    passenger_flight_dic = {}

    print('start')
    index = -1
    for event in events:
        index += 1
        if(index % 100 == 0):
            print(str(index) + '/' + str(len(events)))
        id = event.flight_id
        time = event.time
        airport = event.airport
        plane_in_plan = event.plane
        action = event.action
        duration = get_duration(events, id)
        plane_type = aircrafts_df[aircrafts_df['飞机尾号'] == plane_in_plan].iloc[0,
1]
        earlist = aircrafts_df[aircrafts_df['飞机尾号'] == plane_in_plan].iloc[0,
2]
        latest = aircrafts_df[aircrafts_df['飞机尾号'] == plane_in_plan].iloc[0, 3]
        num_total = get_num_total(paxinfo_df, id)

        if(event.flight_id in NONE_COST_FLIGHT):
            if(event.action == 'takeoff'):
                result.append(Event(id, event.time, 'canceled', event.plane, ''))
            continue

        if(event.action == 'landing' and if_flight_take_off(result, id)):
            landing_passenger = get_landing_passenger(in_travel_passenger, id)
            if ('-' + plane_in_plan) in airplane_dic[airport]:
                airplane_dic[airport].remove('-' + plane_in_plan)
            else:
                airplane_dic[airport].append(plane_in_plan)
            for p in landing_passenger:
                in_travel_passenger.remove(p)
            continue

        takeoff_passengers = get_takeoff_passengers(paxinfo_df, id)
        #pre_flights = get_pre_flights_for_passenger(passenger_flight_dic, 
takeoff_passenger)
        delayed_passengers = get_delayed_passengers(in_travel_passenger, 
takeoff_passengers)

        tmp = get_future_landing_time_and_airport(result, plane_in_plan)
        landing_soon = get_landing_soon(result, event.time, airport)
        same_type_substitute_list = []
        diff_type_substitute_list = []
        if(tmp == None):
            for p in landing_soon:
                if get_plane_type(p.plane, aircrafts_df) == plane_type:
                    same_type_substitute_list.append('*' + p.plane)
                else:
                    diff_type_substitute_list.append('*' + p.plane)
        elif(tmp[1] != airport):
            for p in landing_soon:
                if get_plane_type(p.plane, aircrafts_df) == plane_type:
                    same_type_substitute_list.append('*' + p.plane)
                else:
                    diff_type_substitute_list.append('*' + p.plane)
        else:
            future_landing_time = tmp[0]
            for p in landing_soon:
                if get_plane_type(p.plane, aircrafts_df) == plane_type and p.time < 
future_landing_time:
                    same_type_substitute_list.append('*' + p.plane)
                if get_plane_type(p.plane, aircrafts_df) != plane_type and p.time + 
30 < future_landing_time:
                    diff_type_substitute_list.append('*' + p.plane)
        for plane in airplane_dic[airport]:
            if(plane[0] == '-'): continue
            _earlist = aircrafts_df[aircrafts_df['飞机尾号'] == plane].iloc[0, 2]
            _latest = aircrafts_df[aircrafts_df['飞机尾号'] == plane].iloc[0, 3]
            _plane_type = aircrafts_df[aircrafts_df['飞机尾号'] == plane].iloc[0, 
1]
            if (time >= _earlist and time + duration <= _latest):
                if(_plane_type == plane_type):
                    same_type_substitute_list.append(plane)
                else:
                    diff_type_substitute_list.append(plane)
        if(len(same_type_substitute_list) > 0):
            substitute_list = same_type_substitute_list
        else:
            substitute_list = diff_type_substitute_list
        if(num_total == 0):
            result.append(Event(id, event.time, 'canceled', event.plane, ''))
            continue
        #按计划起飞
        if plane_in_plan in airplane_dic[airport]:
            if(len(delayed_passengers) > 0):
                print()
            best_delay_time, loss = best_delay(delayed_passengers, num_total, time,
time)
            if(best_delay_time == 0):
                event.add_annotation('as plan')
                landing_event = get_landing_event(events, id)
                landing_event.add_annotation('as plan')
                tmp = copy.deepcopy(event)
                tmp.loss = loss
                result.append(tmp)
                tmp = copy.deepcopy(landing_event)
                result.append(tmp)
                airplane_dic[airport].remove(plane_in_plan)
            else:
                event.add_annotation('delayed')
                landing_event = get_landing_event(events, id)
                landing_event.add_annotation('delayed')
                tmp = copy.deepcopy(event)
                tmp.loss = loss
                tmp.time += best_delay_time
                result.append(tmp)
                tmp = copy.deepcopy(landing_event)
                tmp.time += best_delay_time
                result.append(tmp)
                airplane_dic[airport].remove(plane_in_plan)
        #
        elif len(substitute_list) > 0:
            substitute = substitute_list[0]
            flag = 1 if substitute[0] == '*' else 0
            substitute = substitute[1:] if flag == 1 else substitute
            best_delay_time, loss= best_delay(delayed_passengers, num_total, time,
time)
            if(best_delay_time == 0):
                tmp = copy.deepcopy(event)
                tmp.loss = loss
                tmp.plane = substitute
                tmp.annotation = 'substitute'
                result.append(tmp)
                landing_event = get_landing_event(events, id)
                tmp = copy.deepcopy(landing_event)
                tmp.plane = substitute
                tmp.annotation = 'substitute'
                result.append(tmp)
                if flag == 0:
                    airplane_dic[airport].remove(substitute)
                else:
                    airplane_dic[airport].append('-' + substitute)
                tmp = sorted(events[index + 1:], key=key_time)
                events[index + 1:] = tmp
            else:
                tmp = copy.deepcopy(event)
                tmp.loss = loss
                tmp.plane = substitute
                tmp.annotation = 'delayed'
                tmp.time += best_delay_time
                result.append(tmp)
                landing_event = get_landing_event(events, id)
                tmp = copy.deepcopy(landing_event)
                tmp.plane = substitute
                tmp.annotation = 'delayed'
                tmp.time += best_delay_time
                result.append(tmp)
                if flag == 0:
                    airplane_dic[airport].remove(substitute)
                else:
                    airplane_dic[airport].append('-' + substitute[1:])
                tmp = sorted(events[index + 1:], key=key_time)
                events[index + 1:] = tmp
        #延误或取消
        else:
            tmp = get_future_landing_time_and_airport(result, plane_in_plan)
            if(tmp != None):
                landing_time, landing_airport = tmp
                destination_landing_time = landing_time + duration
                best_delay_time, loss = best_delay(delayed_passengers, num_total,
landing_time, time)
                landing_time += best_delay_time
                if (landing_time - time <= 60 * 60 * 5 and landing_airport == airport
and not (\
                    landing_time > BLIZZARD_START and landing_time < BLIZZARD_END)
\
                    and destination_landing_time - 45 * 60 < latest and landing_time >
earlist):
                    tmp = copy.deepcopy(event)
                    tmp.loss = loss
                    tmp.time = landing_time
                    tmp.annotation = 'delayed'
                    result.append(tmp)
                    landing_event = get_landing_event(events, id)
                    tmp = copy.deepcopy(landing_event)
                    tmp.time += landing_time - time
                    tmp.annotation = 'delayed'
                    result.append(tmp)
                    airplane_dic[airport].append('-' + plane_in_plan)
                else:
                    tmp = Event(id, event.time, 'canceled', event.plane, '')
                    tmp.loss = num_total * (24 * 60 * 60)
                    result.append(tmp)
            else:
                result.append(Event(id, event.time, 'canceled', event.plane, ''))

    for event in result:
        if(event.action == 'landing'):
            event.time -= 45 * 60

    result.sort(key=key_time)

    #处理跑道冲突
    while(not adjust_plan(result)):
        pass

    result.sort(key=key_time)

    #输出
    for event in result:
        print('flight id ' + str(event.flight_id) + ' ' + str(event.action) + ' at
' + str(from_unix_stamp(event.time)) + '(unix stamp:' \
              + str(event.time) + ')' + ' in ' + str(event.airport) + '. plane: ' +
str(event.plane) + ', ' + event.annotation)

    to_csv(result, plan_events, './result4.csv')
```