中国研究生创新实践系列大赛

“华为杯”第十八届中国研究生

数学建模竞赛

学　　校	上海交通大学
参赛队号	**21102480018**
队员姓名	**1.范晓宇**
	2.孔郑娇
	3.焦光磊

中国研究生创新实践系列大赛

“华为杯”第十八届中国研究生

数学建模竞赛

题 目　　F　航空公司机组优化排班问题研究

摘　　要：

航空公司机组优化排班问题，往往需要构造特定时间段的机组日程安排，包括每个机组人员在何时何地及哪个航班执行何种任务，这对于保证飞航安全和旅客服务质量至关重要。与此同时，排班问题具有规则复杂难刻画、目标繁多且精细的特点。基于以上背景，本文将复杂的航班机组人员优化排班问题拆解为多个子问题，由浅入深，通过建立数学规划模型并设计基于规则的启发式算法逐步进行求解，最终取得全局较优解。

针对问题一，建立 0-1 整数线性规划模型，满足一系列基础的约束条件，将航班分配机组人员，在尽可能多的航班满足机组配置时减少总体乘机次数，通过 Gurobi 求解器求得数据集 A 的最优解。同时基于规则建立启发式算法，将航段按照日期为单位进行不同层级的组合：将从基地出发并最终返回基地的航班组成回路；不同回路之间根据连接时间长短进行编组。通过航班-回路-编组的多层级概念，在考虑飞行员资格的基础上完成航班分配，最终求得**数据集 A 的满足机组配置航班数为 206（不满足数 0），机组人员总体乘机次数为 8，与 Gurobi 求解器求得的最优解相同。求得数据集 B 的满足机组配置航班数为 13911（不满足数 43），机组人员总体乘机次数为 221。**

针对问题二，本文首先结合执勤概念，在问题一中的线性规划模型基础上加入与执勤相关的约束条件和目标函数；紧接着对启发式算法规则进行优化设计，具体可划分为成对绑定及双向操作航班选择、排班规划和人员资格分配三个阶段，对大小规模数据分别求解，计算得出**数据集 A 的满足机组配置航班数为 206（不满足数 0），机组人员总体乘机次数为 20，总执勤成本为 53.59 万元。求得数据集 B 的满足机组配置航班数为 13649（不满足数 305），总执勤成本为 3969.86 万元。**

针对问题三，在问题二的基础上引入任务环的概念，针对整数规划模型非线性部分，利用辅助变量将 0-1 变量相乘线性化。结合需要拆分任务环的问题特点，以每日执勤数量上下限确立较大的范围区间。执勤的决策的强可调整性，使算法在一定程度上摆脱了贪心算法的局部性，能够得到一个相对于全局的较优解。最后通过虚拟机组人员配置完成任务环决策，并通过执勤成本最小化的贪心策略分配具体机组人员。**最终求得数据集 A 的满足机组配置航班数为 206（不满足数 0），总体执勤成本为 56.03 万元；数据集 B 的满足机组配置航班数为 12411（不满足数 1543），总体执勤成本为 3837.63 万元。**

最后，本文对算法的有效性、稳定性和复杂度进行了评估分析，同时针对连接时间上限阈值ε的灵敏度进行分析，对比总执勤成本和未满足配置航班数得到结论：ε的取值会影响模型求解结果，且ε取 45 时求解结果最优。

关键字：机组人员排班；多目标优化；整数规划；基于规则的启发式算法

目录

一、问题重述

1.1 问题背景

本题目是一个航空公司机组优化排班的问题。所谓机组排班问题，就是构造特定时间段的机组日程安排，包括每个机组人员在何时何地及哪个航班执行何种任务。由于航空公司的运营管理非常复杂，很多过程需要经过长期-中期-短期等多层次的往复循环，因此机组排班问题常常具有规则复杂难刻画、目标繁多且精细的特点[1]。一个高质量的机组航班任务计划，不仅能给航空公司的运营节约成本，还能合理地考虑劳逸平衡，机组偏好，组员同行，培训，时近性和休假等众多因素。

机组优化排班问题围绕下面几个关键概念展开：

1、时间：航空公司的运营是跨时空的，但所有时间均按单一指定时区定义，并由年月日时分进行表达。

2、机场：作为航班和机组人员出发到达的节点，不同机场有特定标识。

3、机组人员：问题简化为考虑具备**两种资格**的飞行员，正机长和副机长，其中部分正机长具备成为副机长的**替补资格**，其他均以本职有且仅有**主要资格**，以上两种资格人员均执行**飞行任务**；此外还有部分机组人员在满足飞机容量约束下执行**乘机任务**。

4、航班：指飞机的一次起飞和降落，相关参数有起飞、降落时间和对应机场，最低机组配置。

5、执勤：由起飞时间必须在同一天的一连串航段（飞行或乘机）和间隔连接时间组成，同时需要满足前后航段机场一致性和相邻航段最短连接时间下限约束。

6、任务环：由一连串的执勤和休息时间组成，且任务环的起点和终点必须都是自己的基地，同时需要满足前后执勤机场一致性和相邻执勤之间休息时间的下限约束。

7、排班计划及排班周期：一个排班周期对应一个排班计划，计划由一系列任务环和休假组成，休假天数需满足下限约束。

为了更直观地理解航段、执勤、任务环和排班计划的关系，综合上述概念简单作图如下所示。

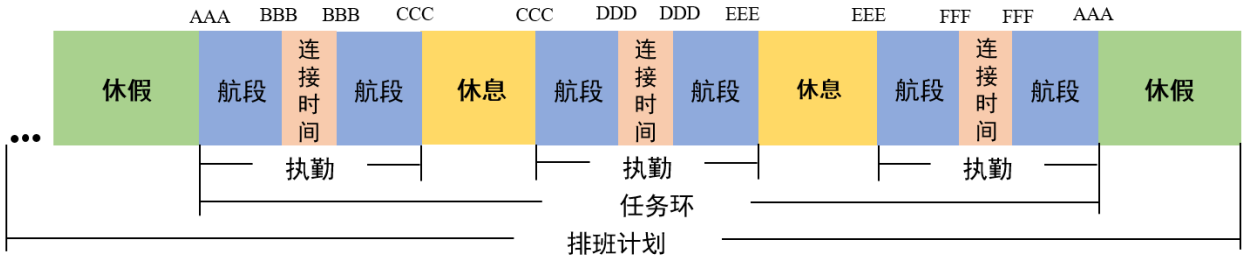

图 1 航段、执勤、任务环和排班计划间关系表达

1.2 问题提出

根据对机组优化排班问题由浅入深分析的过程，题目提出了以下 3 个子问题，其中每个子问题都基于前一个子问题并与之相容：

问题一：只考虑航班机组人员的分配问题，建立线性规划模型求解并给出满足配置的航班数、机组人员总体乘机次数、替补资格使用次数和程序运行时间等数据结果。

需要考虑的目标函数如下，且需要依编号次序满足：

1）最大化满足基本配置的飞机数量；
4）最小化总体乘机次数；
7）最小化使用替补资格的人员数。
需要考虑的约束条件如下：
1.每个机组人员初始从基地出发并最终回到基地；
2. 每个机组人员的下一航段的起飞机场必须和上一航段的到达机场一致；
3. 每个机组人员相邻两个航段之间的连接时间不小于 MinCT 分钟。

问题二：引进执勤概念。在问题一基础上给定每个机组人员的单位小时执勤成本。本问题需要考虑的目标函数如下，且需要依编号次序满足：
1）最大化满足基本配置的飞机数量；
2）机组人员的总体执勤成本最低；
4）最小化总体乘机次数；
5）机组人员之间的执勤时长尽可能平衡。
7）最小化使用替补资格的人员数。
同时在满足子问题一约束的基础上进一步要求：
1. 每个机组人员每天至多只能执行一个执勤；
2. 每次执勤的飞行时间最多不超过 MaxBlk 分钟；
3. 每次执勤的时长最多不超过 MaxDP 分钟；
4. 每个机组人员下一执勤的起始机场必须和上一执勤的结束机场一致；
5. 每个机组人员的相邻两个执勤之间的休息时间不小于 MinRest 分钟。

问题三：编制排班计划。在问题一、2 基础上再给定每个机组人员的单位小时任务环成本。本问题需要考虑的目标函数如下，且需要依编号次序满足：
1）最大化满足基本配置的飞机数量；
2）机组人员的总体执勤成本最低；
3）机组人员的总体任务环成本最低；
4）最小化总体乘机次数；
5）机组人员之间的执勤时长尽可能平衡。
6）机组人员之间的任务环时长尽可能平衡。
7）最小化使用替补资格的人员数。
同时在满足子问题一和二的约束的基础上进一步要求：
1. 每个机组人员每个排班周期的任务环总时长不超过 MaxTAFB 分钟；
2. 每个机组人员相邻两个任务环之间至少有 MinVacDay 天休息；
3. 每个机组人员连续执勤天数不超过 MaxSuccOn 天。

二、问题分析

2.1 问题一分析

问题一需要建立线性规划模型并求解给出**航班机组人员分配方案**，本问题作为整个建模题目的基本题，将机组优化排班问题中的一些复杂条件加以简化，如暂不考虑人员的排班和休息问题，以及机组人员薪酬带来的成本问题等。

从目标函数和约束条件方面看，该子问题需要首先满足一系列基础的约束条件，如计划周期内的起点和终点都必须是机组人员自身对应的基地，因此需对该人员的始末航段轨迹进行跟踪；需保证相连航段机场的一致性以及相邻航段间必需的连接时间的最短要求，因此需对两航班相邻性进行判别定义；该子问题的优化目标主要体现在对航班的利用率和机组人员执行任务时职务的要求上。

该问题的**核心和难点**主要集中在**对机组人员执行任务时职务的分类和判别**，即针对一名机组人员，首先讨论其是否具备替换资格，若具备替换资格，判别其担任了正机长还是副机长；若不具备，则其只能一直处于正机长或副机长其中一种身份。当清晰机组人员的身份后，便可以由此判断各身份人数是否满足**机组资格配置**以及**替补资格使用**情况，同时，基于全部身份的总人数之和与机组基本配置要求，求差可得**乘机人数**。

2.2 问题二分析

问题二是在问题一（考虑机组人员分配）基础上引入了**执勤**的概念，与问题一不同的是，问题二考虑了机组人员的休息时间，执勤成本及机组人员间执勤时长平衡的额外因素，从而实现对机组排班问题的复杂化。同时，执勤的概念与天数相对应，即对于某个机组人员来说，一次执勤意味着当天一定有至少一个分配的航班会起飞，而这次执勤中所有航班的起飞时间也必须都在这一天，两者一一对应，因此需要在问题二中引入关于**天数的变量**来反映执勤情况。

从约束条件方面看，该子问题的约束条件包括每个机组人员每天至多只能执行一个执勤，保证了执勤次数与天数的对应性，意味着不存在一次执勤将一天分割的情况；同时对一次执勤的总时长和飞行时间分别有上限要求，以及对相邻执勤的休息时间有下限要求，因此该问题的**核心和难点**在于引入天数的变量后，还需要**明确一次执勤（一天）中的首尾航班**及其对应的起飞和降落时间，并**判别执勤中的具体任务类型**（飞行任务、乘机任务还是连接时间）；前后执勤的机场一致性约束意味着机组人员休息期间不能改变其自身所在的机场，该条约束实质上和问题一中的前后航班机场一致性约束相同，因此在本问题中无需再额外考虑。

目标函数添加了总体执勤成本最小化的约束，其优先级仅次于最大化满足配置的航班数，因此在人员分配时应在满足航班配置基础上尽可能**缩短执勤时间**，增加其休息时间。此外，机组人员间的执勤时长尽可能平衡意味着安排每个机组人员适当平均的执勤与休息时间，减少个别极端的执勤或休息时长。因此该优化目标的关键点在于对**任务平衡性评价方式的选择**上。

2.3 问题三分析

问题三在问题一、二基础上引入了任务环的概念，因此需要进一步考虑任务环之间机组人员休假问题以及任务环成本问题，至此机组优化排班问题已大体完整，需考虑的因素繁多，本题最终需编制计划周期内完整的排班计划。

从目标函数和约束条件来看，本题添加的两个目标形式上与问题二相似，即关于薪酬成本和人员的任务平衡问题。但综合前两问目标函数并按序号形成优先级，可以得出应在尽可能满足机组基本配置的基础上首先追求执勤成本和任务环成本的最小化，之后再考虑乘机人数、工作强度平衡以及替补资格问题，因此在设计算法的过程中，也要根据优先级制定相应的**启发式规则**。本问题需要引入**与休假状态相关变量**来表示或间接表示约束条件中对任务环总时长、休假时间和连续执勤天数的要求，同时该问题也包含了计划周期内每轮任务环需回到基地的**隐含约束**，而该约束是**分割任务环**的关键点。

本问题的**核心和难点**在于如何综合问题一、二多方面条件限制对任务环进行生成并对人员进行分配，具体可体现为需要根据优先级设定优先规则，有效处理最优化各个目标实现过程中可能出现的矛盾情况。

三、模型假设

• 假设 1：本题假定每个航班都是唯一的，不考虑存在航班号、日期、起飞降落时间和地点、机组资格配置都完全相同的两趟航班。
• 假设 2：本题假定只有一种机型一种配置，即不考虑不同机型间机组人员的专用性，同时每趟航班执行飞行任务的正副机长人数固定不变。
• 假设 3：本题假定所有机组人员的初始位置和排班周期结束时的终了位置都在其基地，不考虑机组人员跨排班周期执行任务环的情况。
• 假设 4：本题假定机组人员之间可以任意组合，不考虑人员间具有特殊匹配性问题。
• 假设 5：本题假定允许存在因为无法满足最低机组资格配置而不能起飞的航班。
• 假设 6：本题假定不满足最低机组资格配置的航班不能配置任何机组人员，即当因不满足资格配置而不能起飞时，该航班的机组人员数设为零。
• 假设 7：本题假定机组人员可以乘机摆渡，即实际机组配置可以超过最低配置要求，乘机机组人员的航段时间计入执勤时间，但不计入飞行时间。
• 假设 8：本题假定飞机的起飞降落时间严格固定，不考虑飞机延误情况。
• 假设 9：本题假定机组人员工作过程中不会有突发情况，不考虑缺勤、临时请假等不确定事件的发生。

四、符号说明

本文中所用符号的说明如下表所示。

指标：

符号	含义
i	可用机组人员编号的指标，$i \in I_f \cup I_a$
j	航班编号的指标，$j \in J$
d	计划周期内的天数编号的指标，$d \in D$

输入参数：

符号	含义
D	计划周期天数集合
$Toal\_T$	计划周期总天数
J	计划周期内所有航班集合
JD_p	从机场p起飞的航班集合
JA_p	在机场p降落的航班集合
J_d	起飞时间在第d天的航班集合
I_f	不具备替补资格的机组人员集合
I_a	具备替补资格的机组人员集合
Day_j	航班j的起飞日期
O_i	机组人员i的自身基地
EP_{ij}	航班j的降落机场是机组人员i的自身基地时为 1，否则为 0
φ_j	能够满足与航班j相连接要求的紧后航班集合
DT_j	航班j的起飞时间
AT_j	航班j的降落时间
K_i	不具备替补资格的机组人员i是副机长时为 1，是正机长时为 0，$i \in I_f$
SF	设置虚拟的初始航班
EF	设置虚拟的最终航班
$Comp\_c$	航班最低机组资格配置中正机长数

$Comp\_f$	航班最低机组资格配置中副机长数
$MinCT$	航段之间最小连接时间
$MaxBlk$	一次执勤飞行时长上限
$MaxDP$	执勤时长上限
$MinRest$	相邻执勤间休息时间下限
$MaxDH$	每趟航班最多乘机人数上限
$MaxTAFB$	排班周期单个机组人员任务环总时长上限
$MaxSuccOn$	连续执勤天数上限
$MinVacDay$	相邻两个任务环间休假时间下限
$DutyCostPerHr_i$	机组人员i的每小时执勤工资
$ParingCostPerHr_i$	机组人员i的每小时任务环工资
M	一个足够大的数

决策变量：

符号	含义
x_{ij}	机组人员i被分配到航班j时为 1，否则为 0，$i \in I_f \cup I_a$
x_{ij1}	具备替补资格并担任副机长的机组人员i被分配到航班j时为 1，否则为 0，$i \in I_a$
x_{ij0}	具备替补资格仍担任正机长的机组人员i被分配到航班j时为 1，否则为 0，$i \in I_a$
$z_{ijj'}$	对于机组人员i，航班j'是航班j的紧后航班时为 1，否则为 0
s_j	航班j满足最低机组资格配置时为 1，否则为 0
e_{ij}	航班j是机组人员i当天的最后一班时为 1，否则为 0
l_{ij}	航班j是机组人员i当天的第一班时为 1，否则为 0
v_{id}	机组人员i在第d天执勤时为 1，否则为 0
y_{ij}	机组人员i在航班j上且乘机取 1，否则为 0
c_{id}	机组人员i在第d天处于休息状态则为 1，否则为 0
r_{id}	机组人员i在第d天处于休假状态则为 1，否则为 0

rs_{id}	机组人员i在第d天开始进行休假则为 1，否则为 0
re_{id}	机组人员i在第d天结束休假（此天仍在休假）则为 1，否则为 0

辅助变量：

符号	含义
u_j	问题一中副机长人数满足最低资格配置时取 1，否则为 0，$j \in J$
w_j	问题一中正机长人数满足最低资格配置时取 1，否则为 0，$j \in J$
max_duty_time	所有机组人员中最长的执勤时间
min_duty_time	所有机组人员中最短的执勤时间
max_work_time	所有机组人员中最长的任务环时间
min_work_time	所有机组人员中最短的任务环时间

五、问题一：模型建立与求解

5.1 模型预处理

在建立模型前，我们首先根据题目特征和模型所需参数对现有数据进行分类筛选，依据建模需要将整个数据集划分为多个所需特定集合，并结合问题特征适当添加新的参数，增加数据所能体现的信息量。对数据进行预处理有利于挖掘已知数据的隐含信息，避免模型中大量多余约束，同时有利于降低决策变量维度，一定程度上实现模型的简化。具体预处理工作如下所示：

• 时间表达：将计划周期内的日期全都转化为序数的概念，具体表示为与第一天 0 点 0 分的间隔时间，以“2021-8-11”为计划周期第一天，则“2021-8-15”为计划周期第五天，当天 0 点 0 分时刻表示为 1440*（5-1）=5760 分钟。

• 集合划分：1. 问题中关于相邻航班机场连贯性的相关约束，可以预先对航班起飞降落地点进行归纳整理，生成每个机场所对应的起飞和降落航班集合，同时筛选出对每个航班而言，连接时间符合最小连接时间标准的**紧后航班的集合**。通过模型预处理，直接形成集合的映射，减少了模型中关于航班是否能够连接的约束。2. 问题中关于机组人员职务的判别，首先依据数据对人员进行划分，划分依据为是否具备替换资格，在模型中针对特定**人员类型集合**讨论，省去模型中决策变量关于人员类型的维度。3. 针对问题中关于执勤与天数关系的约束，依据起飞的具体日期对航班进行划分，针对每一天生成**在当天起飞的航班集合**。

• 参数引入：问题中关于航班最终需要返还基地的约束，通过对每趟航班的降落机场以及该机场是否为机组人员对应基地的双重筛选，作为参数给出每趟航班是否返回基地的判断结果可直接应用于模型，因此避免了大量关于航班必须返回基地的约束，使模型得到进一步简化。

5.2 线性规划模型建立

根据题目要求及模型预处理建立整数规划模型如下：

$$1) \quad \max \sum_{j \in J} s_j \tag{1}$$

$$4) \quad \min \sum_{i \in I_f} \sum_{j \in J} x_{ij} + \sum_{i \in I_a} \sum_{j \in J} (x_{ij1} + x_{ij0}) - \sum_{j \in J} (Comp_c + Comp_f) \tag{2}$$

$$7) \quad \min \sum_{i \in I_a} \sum_{j \in J} x_{ij1} \tag{3}$$

s.t.

$$\sum_{j \in JD_{O_i} \cup \{EF\}} z_{i(SF)j} = 1 \qquad \forall i \in I_f \cup I_a \tag{4}$$

$$\sum_{j \in JA_{O_i} \cup \{SF\}} z_{ij(EF)} = 1 \qquad \forall i \in I_f \cup I_a \qquad (5)$$

$$x_{ij} \leq \sum_{j' \in \varphi_j \cup \{EF\}} z_{ijj'} \qquad \forall i \in I_f, \forall j \in J \cup \{SF\} \qquad (6)$$

$$x_{ij0} \leq \sum_{j' \in \varphi_j \cup \{EF\}} z_{ijj'} \qquad \forall i \in I_a, \forall j \in J \cup \{SF\} \qquad (7)$$

$$x_{ij1} \leq \sum_{j' \in \varphi_j \cup \{EF\}} z_{ijj'} \qquad \forall i \in I_a, \forall j \in J \cup \{SF\} \qquad (8)$$

$$x_{ij} \leq \sum_{j' \in J \cup \{SF\}} z_{ij'j} \qquad \forall i \in I_f, \forall j \in \varphi_j \cup \{EF\} \qquad (9)$$

$$x_{ij0} \leq \sum_{j' \in J \cup \{SF\}} z_{ij'j} \qquad \forall i \in I_a, \forall j \in \varphi_j \cup \{EF\} \qquad (10)$$

$$x_{ij1} \leq \sum_{j' \in J \cup \{SF\}} z_{ij'j} \qquad \forall i \in I_a, \forall j \in \varphi_j \cup \{EF\} \qquad (11)$$

$$x_{ij0} + x_{ij1} \leq 1 \qquad \forall i \in I_a, \forall j \in J \qquad (12)$$

$$\sum_{j' \in \varphi_j} z_{ijj'} \leq 1 \qquad \forall i \in I_f \cup I_a, \forall j \in J \qquad (13)$$

$$\sum_{j' \in J} z_{ij'j} \leq 1 \qquad \forall i \in I_f \cup I_a, \forall j \in \varphi_j \qquad (14)$$

$$\sum_{i \in I_f} K_i \, x_{ij} + \sum_{i \in I_a} x_{ij1} - Comp\_f - u_j M < 0 \qquad \forall j \in J \qquad (15)$$

$$\sum_{i \in I_f} K_i \, x_{ij} + \sum_{i \in I_a} x_{ij1} - Comp\_f + (1 - u_j) M \geq 0 \qquad \forall j \in J \qquad (16)$$

$$\sum_{i \in I_f} (1 - K_i) \, x_{ij} + \sum_{i \in I_a} x_{ij0} - Comp\_c - w_j M < 0 \qquad \forall j \in J \qquad (17)$$

$$\sum_{i \in I_f} (1 - K_i) \, x_{ij} + \sum_{i \in I_a} x_{ij0} - Comp\_c + (1 - w_j) M \geq 0 \qquad \forall j \in J \qquad (18)$$

$$s_j \leq u_j \qquad \forall j \in J \qquad (19)$$

$$s_j \le w_j \qquad \forall j \in J \qquad (20)$$

$$s_j \ge u_j + w_j - 1 \qquad \forall j \in J \qquad (21)$$

$$\sum_{i \in I_f} x_{ij} + \sum_{i \in I_a} (x_{ij0} + x_{ij1}) \le s_j M \qquad \forall j \in J \qquad (22)$$

$$x_{ij} \in \{0,1\} \qquad \forall i \in I_f, \forall j \in J \qquad (23)$$

$$x_{ij1}, x_{ij0} \in \{0,1\} \qquad \forall i \in I_a, \forall j \in J \qquad (24)$$

$$z_{ijj'}, s_j, u_j, w_j \in \{0,1\} \qquad \forall i \in I_f \cup I_a, \forall j \in J, \forall j' \in \varphi_j \qquad (25)$$

目标函数（1）为首要优化目标，即最大化满足机组配置的航班数。目标函数（2）为次要考虑的目标，即总体乘机次数最小化，总乘机次数具体可表示为对于所有航班来说，除最低机组资格配置人数外，被分配到这些航班的人数总和。目标函数（3）为最后需要考虑的优化目标，最小化使用替补资格的人数。约束（4）-（11）中SF和EF分别为计划周期内设置的虚拟初始航班和虚拟结束航班，通过建立其与起飞或降落机场为该机组人员地的连接约束，以及两个虚拟航班必须是起始和最终航班的位置约束，从而实现对每个机组人员初始从基地出发并最终回到基地约束的满足。约束（12）表示针对一名具备替换资格的机组人员，在一趟航段中只能选择一种职务，即正机长或是副机长。约束（13）和（14）保证了对于一名机组人员，其所在航班紧前航班和紧后航班的唯一性。约束（15）-（21）表明了针对一趟航班，其分别拥有的正、副机长人数是否都能满足航班的最低机组配置要求与该趟航班最终是否能够顺利起飞的关系，该约束条件组利用了处理逻辑约束的经典线性化方法，只有正、副机长人数的两个约束都能满足航班才算满足配置。约束（22）表示不满足最低机组资格配置的航班不能配置任何机组人员。约束（23）-（25）表明该模型决策变量都为 0-1 变量。

5.3 模型特点分析

本问题模型复杂度为$O(|J| \times |J| \times |I_f \cup I_a|)$，以小规模数据为例，模型约束 40000 条左右，变量个数 30000 个左右，模型规模中等，求解器能够以较短的求解时间求出最优解。

与题目给出的参考文献二[2]中的模型相比，本问题通过对模型的前期预处理，将航班分成不同集合并增加参数作为模型的输入，使变量的维度和个数明显减少，在特定集合下分别进行求解大大减少了约束数量，简化求解过程，有效降低了模型的复杂度。同时，将决策变量全部设置为 0-1 整数变量，一定程度上也使得模型求解更加容易。

5.4 模型求解

5.4.1 Gurobi 商用求解器

针对问题一中的 A 套小规模数据，结合模型规模和结构，由于前期对现有数据的预处

理大大减少了变量维度和数量，因此有效降低了模型复杂度。因此本节利用商用求解器Gurobi9.1.1，运用 python 语言对其进行编程。导入预处理后的数据，输入目标函数和约束条件，最终求得最优解。

5.4.2 基于规则的启发式算法

本节的算法核心为：基于题目中目标和约束指定规则。启发式算法可以获得局部最优解，但无法保证一定能得到模型的全局最优解，因此关键是对规则的制定。

本节采取的主要规则是先对所有航班进行回路绑定，再分组绑定，然后尽可能实现小组间的连接形成执勤绑定，同时针对性处理单独的未编组航班，最后基于人员资格进行有优先级的人员分配。

算法实现的关键步骤如下：

表 1 基于规则的启发式算法主要思路

基于规则的启发式算法
（1）**回路绑定：**如果从基地出发的若干航班能够两两连接（两航班的降落起飞机场一致且后续航班的起飞时间比前序航班降落时间晚且超过$MinCT$分钟，但同时不超过ε分钟，ε为设定的上限阈值）并最终能够返回基地形成回路，则绑定这若干个航班为回路航班群。 （2）**分组绑定：**从当天起飞时间最早的航班开始依次搜索，若存在一个回路航班群降落后在基地有另一个回路航班群即将起飞，且两回路航班群间隔时间超过$MinCT$分钟但同时不超过ε分钟，则绑定这两个回路航班群为一个小组航班。 （3）**执勤绑定：**在此步骤提前引入执勤概念，工作一天对应一次执勤，因此一天内的所有航班可以被分为几个小组和部分单独未编组航班。判断这些组间（包含未编组）是否能够继续连接成为一个大组，即成为一次执勤并分配给一对机组配置人员。 （4）**单独未编组航班处理：**针对未编组情况，考虑最小化乘机次数目标，设置三种自上而下依次优先的处理方式。 • 在当天所有未编组航班中搜索是否存在一趟航班其起飞和降落机场恰好与未编组航班相反，若是则结成一组。 • 在当天之前的所有天数中查找是否存在未编组且起飞和降落机场恰好相反的航班，若是则结成一组。 • 若已分组航班中存在降落机场与单独未编组航班的起飞机场一致的情况，判断连接时间是否大于$MinCT$分钟，由此安排机组人员在该趟航班上乘机到达未编组航班的起飞机场。 （4）**按职务分配机组人员：**根据尽可能少使用替补资格的目标，且本问未考虑人员休息情况，因此一天内优先安排不具备替补资格的正机长和副机长执行任务，同时考虑满足机组配置为首要目标，因此若当天所需机组配置人员大于不具备替补资格人数，再安排机组人员使用替补资格。 （5）满足所有约束条件和规则后，输出机组人员分配方案，以及满足机组配置的航班数、总体乘机次数和替补资格使用次数。

伪代码如表 2 所示：

表 2 基于规则的启发式算法伪代码

基于规则的启发式算法伪代码
Input：机组人员信息、航班信息
Output：机组排班信息、可行航班数、乘机数、替补数

```
Initialize：读取数据
01.for  d=1:NumD
02.  for  i 为 d 天的航班
03.    for  j 为 d 天其他航班
04.      if  i 与 j 构成回路 and 45>=连接时间>=40
05.        把 i 和 j 连成一个回路
06.      end
07.    end
08.end
09.for  d=1:NumD
10.  for  i 为 d 天的回路
11.    for  j 为 d 天其他回路
12.      if  45>=i 与 j 连接时间时间>=40
13.        把 i 和 j 连成一个小组
14.        继续接着 j 寻找连接，知道无可连接
15.      end
16.    end
17.end
18.把未编组回路自己当做一个小组
19.把任意小组和单独未编组航班一起相互组合，如果可连接则连接
20.for  i 是某个未编组航班
21.  if 今日不能加一个乘机给 i 编组
22.    从之前的天数找一个可以乘机去 i 出发机场的航班，编组
23.  elseif  今天能加一个乘机给 i 编组
24.    从今天找一个可以乘机去 i 出发机场的航班，编组
25.  end
26.end
27.分配人员，按机组人员不使用替补到必须使用替补，依次分配
28.整理排班方案，输出结果
```

5.4.3 考虑多基地拆分策略的算法优化

前文提到的基于规则的启发式算法仅限于起飞和降落机场一致且唯一的情况，即只考虑所有机组人员来自相同的一个基地，尚未考虑 B 组大数据规模下机组人员来自多基地的情况。当基地数由一个变为多个，在计划周期开始时不同的机组人员要从不同机场出发，并在计划周期结束时回到不同的机场，因此单基地的航线安排不再适用，需重新进行航线安排及人员分配[3]。

对于多基地问题，本题借鉴了多车场路径规划问题中的订单拆分策略，将所有航班和机组人员根据基地数量进行拆分，拆分后形成若干个航班集合和机组人员集合，每个基地及其对应的航班和机组人员集合再作为一个子问题输入到上述单基地基于规则的启发式算法中进行求解。具体拆分策略如下表所示。

表 3 多基地拆分策略说明

多基地拆分策略
（1）**机组人员拆分：**将所有机组人员根据其自身对应基地划分为若干个集合。 （2）**基于起飞降落机场的航班拆分：**将所有航班按照基地的种类进行划分。以 B 组数据为例，共考虑两个基地 a 和 b，则首先将所有航班中起飞和降落机场是 a 的航班划分为一个集合 A；起飞和降落机场是 b 的航班划分为另一个集合 B。 （3）**基于连接时间的航班拆分：**步骤（2）中的划分尚未包括既包含 a 又包含 b 的航班和起飞降落都不在 a 或 b 的航班。针对这类航班，依据连接时间进行划分，同时结合已知数据特点，考虑到 a 基地人数远大于 b 基地且 a 基地副机长人数远大于正机长，意味着使用替补资格的可能性低，因此 a 拥有对未划分航班的优先选择权，选择满足最短连接时间且连接时间尽可能短的航班划分到 A 集合中，其余航班划分到 B 集合。 （4）**输出划分集合：**将划分完成的机组人员集合和航班集合按照基地种类一一对应。至此，每个基地分别对应一部分航班和机组人员，并作为子问题沿用单基地基于规则的启发式算法进行航线安排和人员分配。

5.5 求解结果与分析

5.5.1 A 组小规模数据结果展示

针对问题一的 A 组小规模数据，利用商用软件 Gurobi 及设计的基于规则的启发式算法分别进行求解并对求解结果进行比较。结果显示，两种方法在小规模问题中都能求到最优解且结果一致，但求解时间上基于规则的启发式算法表现明显优于 Gurobi，具体输出结果如表 4 所示。

表 4 问题一小规模数据求解结果

结果指标	子问题 1-数据集 A
不满足机组配置航班数	0
满足机组配置航班数	206
机组人员总体乘机次数	8
替补资格使用次数	0
程序运行分钟数（Gurobi v.s.启发式算法）	14.33 v.s. 0.01

由表可知，问题一的机组人员分配基本问题由于考虑因素较少，计划周期短，因此计划内所有航班都能够满足机组配置要求。同时，总体乘机次数也较少且机组人员没有使用替补资格。

综上分析，模型能够很好地解决问题一，且算法实现具备较高的准确率，这也为后续问题的求解打下了基础。

5.5.2 B 组大规模数据结果展示

针对 B 组大规模数据，利用基于规则的启发式算法同时考虑多基地拆分策略对该问题进行求解，具体输出结果如表 5 所示。

表 5 问题一大规模数据求解结果

结果指标	子问题 1-数据集 B
不满足机组配置航班数	43
满足机组配置航班数	13911
机组人员总体乘机次数	221
替补资格使用次数	70
程序运行分钟数	0.03

由表可知，当数据规模变大，机组人员所属基地不再全部相同时，问题一的排班优化问题变得复杂化，因此出现了部分航班无法满足机组配置的情况，这也是符合常识的。同时，基地数增加会使航班的起飞和降落机场不再集中于一个机场，因此通过乘机实现机组人员的运输的情况会明显增加。此外，航班数量大规模增加以及人员排班复杂化使得某些航班不得不使用机组人员的替补资格来解决原本机组人员人数不够或主副机长人数差距悬殊的问题。程序运行时间表明算法的求解效率较高，能够在短时间求出较优解。

六、问题二：模型建立与求解

6.1 线性规划模型建立

根据题目要求及模型预处理，在问题一的基础上引入与执勤的相关决策变量和时间、成本上的约束，建立整数规划模型如下：

$$1) \quad \max \sum_{j \in J} s_j \tag{1}$$

$$2) \quad \min \sum_{d \in D} \sum_{i \in I_f \cup I_a} \sum_{j \in J_d} DutyCostPerHr_i \times (AT_j e_{ij} - DT_j l_{ij}) \tag{26}$$

$$4) \quad \min \sum_{i \in I_f} \sum_{j \in J} x_{ij} + \sum_{i \in I_a} \sum_{j \in J} (x_{ij1} + x_{ij0}) - \sum_{j \in J} (Comp_c + Comp_f) \tag{2}$$

$$5) \quad \min(max\_duty\_time - min\_duty\_time) \tag{27}$$

$$7) \quad \min \sum_{i \in I_a} \sum_{j \in J} x_{ij1} \tag{3}$$

s.t 约束(4) - (25)

$$x_{ij} = x_{ij0} + x_{ij1} \qquad \forall i \in I_a, \forall j \in J \tag{28}$$

$$\sum_{j \in J_d} (x_{ij} + l_{ij} + e_{ij}) \leq M\, v_{id} \qquad \forall i \in I_f \cup I_a, \forall d \in D \tag{29}$$

$$\sum_{j \in J_d} l_{ij} = v_{id} \qquad \forall i \in I_f \cup I_a, \forall d \in D \tag{30}$$

$$\sum_{j \in J_d} e_{ij} = v_{id} \qquad \forall i \in I_f \cup I_a, \forall d \in D \tag{31}$$

$$l_{ij} \leq x_{ij} \qquad \forall i \in I_f \cup I_a, \forall j \in J \tag{32}$$

$$e_{ij} \leq x_{ij} \qquad \forall i \in I_f \cup I_a, \forall j \in J \tag{33}$$

$$l_{ij'} \leq 1 - x_{ij} \qquad \forall i \in I_f \cup I_a, \forall j \in J, \forall j' \in J_{Day_j} \cap \varphi_j \tag{34}$$

$$e_{ij} \leq 1 - x_{ij'} \qquad \forall i \in I_f \cup I_a, \forall j \in J, \forall j' \in J_{Day_j} \cap \varphi_j \tag{35}$$

$$\sum_{j\in J_{d+1}} DT_j l_{ij} - \sum_{j\in J_d} AT_j e_{ij} \geq MinRest \qquad \forall i \in I_f \cup I_a \qquad (36)$$

$$\sum_{j\in J_d} AT_j e_{ij} - \sum_{j\in J_d} DT_j l_{ij} \leq MaxDP \qquad \forall i \in I_f \cup I_a \qquad (37)$$

$$y_{ij} \leq x_{ij} \qquad \forall i \in I_f \cup I_a, \forall j \in J \qquad (38)$$

$$\sum_{i\in I_f} K_i(x_{ij} - y_{ij}) + \sum_{i\in I_a}(x_{ij1} - y_{ij}) \leq Comp\_f \qquad \forall j \in J \qquad (39)$$

$$\sum_{i\in I_f}(1 - K_i)(x_{ij} - y_{ij}) + \sum_{i\in I_a}(x_{ij0} - y_{ij}) \leq Comp\_c \qquad \forall j \in J \qquad (40)$$

$$\sum_{j\in J_d}(x_{ij} - y_{ij})(AT_j - DT_j) \leq MaxBlk \qquad \forall i \in I_f \cup I_a \qquad (41)$$

$$max\_duty\_time \geq \sum_{d\in D}\sum_{j\in J_d}(AT_j e_{ij} - DT_j l_{ij}) \qquad \forall i \in I_f \cup I_a \qquad (42)$$

$$min\_duty\_time \leq \sum_{d\in D}\sum_{j\in J_d}(AT_j e_{ij} - DT_j l_{ij}) \qquad \forall i \in I_f \cup I_a \qquad (43)$$

$$max\_duty\_time, min\_duty\_time \text{ integer} \qquad (44)$$

$$e_{ij}, l_{ij}, y_{ij} \in \{0,1\} \qquad \forall i \in I_f \cup I_a, \forall j \in J \qquad (45)$$

$$v_{id} \in \{0,1\} \qquad \forall i \in I_f \cup I_a, d \in D \qquad (46)$$

目标函数（1）、（2）、（3）分别对应问题一中的优化目标，而（26）和（27）为问题二需要新加入的优化目标，分别表示了最小化机组人员的总体执勤成本以及机组人员之间的执勤时长尽可能平衡。根据 Bhaskar and Srinivasan[4]的研究，最小化与平均工作负载偏差的平方和可以近似为最小化最大工作负载和最小工作负载之间的差异。因此为了保持模型的线性化，本题采用了极差法对执勤时长的平衡问题进行衡量，将最长执勤时长与最短作差，极差越小表示人员间执勤时长越均衡。所有目标函数前的编号代表其考虑的优先次序。由于问题一与问题二的连贯性，问题二仍可沿用问题一的所有约束。此外，约束（28）对问题一中的不同类型机组人员进行整合，即后续约束条件将不再考虑人员是否具备替补资格，从而利于变量的统一表达。约束（29）-（33）表示只有当机组人员i执勤时才会被分配到航班上并一定会产生当天的首末航班。约束（34）和（35）对一天内的首末航班进行定义，即当天第一趟航班一定没有前序航班，最后一趟一定没有后序航班。约束（36）表示每个机组人员的相邻两个执勤间的休息时间不小于$MinRest$分钟。约束（37）表示每次执勤的时长最多不超过$MaxDP$分钟。约束（38）表明只有机组人员i被分配到航班j上，才能执行乘机任务。约束（39）和（40）表明在航班j上的机组人员，执行飞行任务的人数

不能超过机组资格配置。约束（41）表明每次执勤的飞行时间最多不超过$MaxBlk$分钟。约束（42）和（43）为找到执勤时间最长和最短的线性化公式。约束（44）表示执勤时长的最大最小值为整数。约束（45）和（46）表示该模型中决策变量都为 0-1 变量。

6.2 模型求解

本问题中引入执勤概念，包括执勤时长、执勤中的飞行时长以及相邻执勤间休息时长的约束，目标函数增加了对执勤成本和执勤时长平衡问题的评价。根据目标函数优先级，该问题首要考虑目标为最大化满足机组配置的航班数，其次是尽可能追求执勤成本的最小化，第三个目标是尽量减少机组人员的乘机次数。问题一中所提出的基于规则的启发式算法对第一优先级的目标进行了充分的考虑，因此问题二的**算法优化重点在原算法基础上加入对执勤成本最小化的考虑**，同时在不违背该目标最优的前提下尽可能优化后续目标。

执勤成本最小化，即最小化执勤时长，而执勤时长=飞行时长+乘机时长+连接时间时长，飞行时长是固定不变的，因此本题目标应在满足飞机满足配置基础上尽可能减少机组人员的乘机时长和航班间的连接时长，因此对该目标的优化一定程度上也有利于减少乘机总次数。本节**算法核心为设计排班规则使得乘机时长和连接时长尽可能小**。

6.2.1 成对绑定及双向操作航班选择

该小节属于考虑执勤因素的基于规则的启发式算法的前期预处理阶段，该阶段主要任务是绑定成对航班，并确定一天内能够实现双向操作航班的数量。具体算法在该阶段主要思路如下所示。

该阶段算法实现的关键步骤如下：

表 6 成对绑定及双向操作航班选择主要思路

成对绑定及双向操作航班选择
（1）**成对绑定：**如果存在航班从基地出发，降落后在满足连接时间（超过$MinCT$分钟，但同时不超过ε分钟）的情况下有后序航班返回基地，则将上述两趟航班进行成对绑定。从第一天开始记录每天成对航班的对数，查找重复出现的成对航班并记录小组信息、开始和结束重复的时间以及重复天数。 （2）**双向选择航班：“双向选择航班”定义：**当存在多趟成对航班时，为了保证更多的机组满足资格配置，因此在满足连接时间要求条件下，考虑将多趟成对航班合并成为一组航班，但同时会导致成对航班间产生大量连接时间。因此提出“双向操作航班”策略，即在开始重复的当天最早从基地出发的航班上多配置一组机组人员，使其执行乘机任务，到降落机场后原本执行飞行任务的人员在机场开始休息，由前序航班中乘机的一组机组人员执行飞行任务，以此类推，每次航班降落都有一组人员就地休息，直到结束重复的当天，返回基地的航班需要将在起飞机场休息的人员一起带回基地，因此除了重复天数内开始和结束增加的乘机时长，其余时间段内的连接时间均可节省，具体过程间图 2 所示。**“双向选择航班”的确定：**“双向选择航班”能够存在的对数=一天内最多配置人员组数-一天内存在的成对航班对数，因此选择最能节省时间的“双向选择航班”。

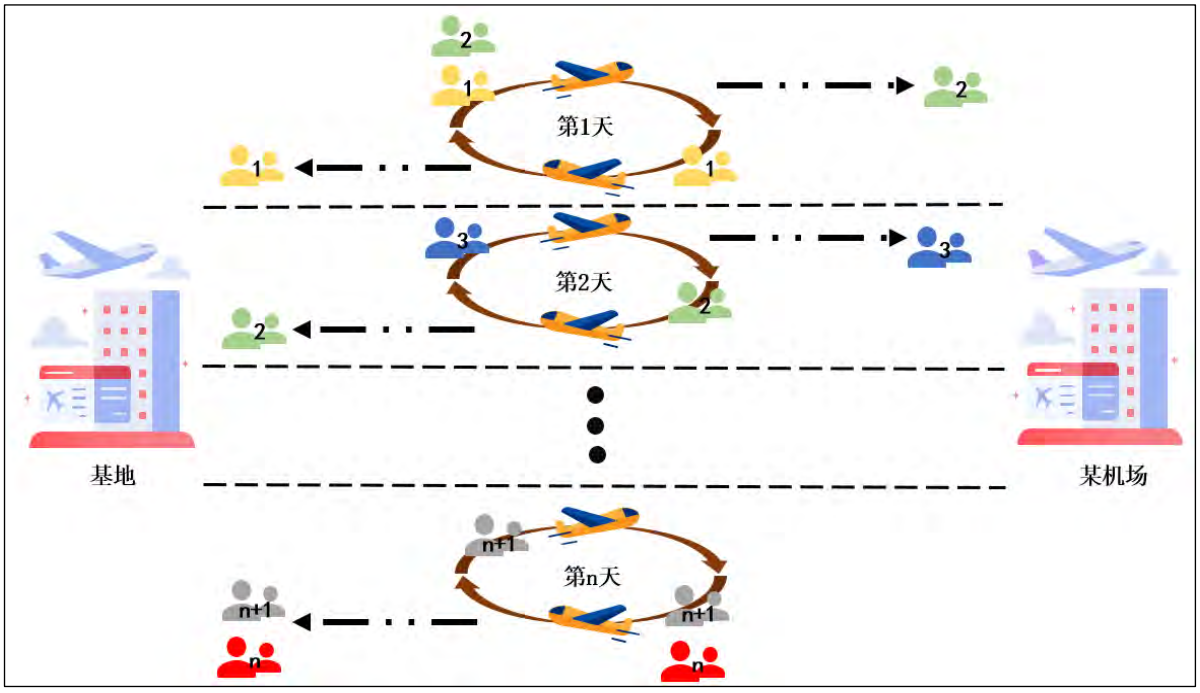

图 2 双向选择航班具体过程

6.2.2 排班规划

排班规划阶段主要任务是基于上个阶段的双向选择航班为剩余单独未编航班安排人数，伪代码如下表 7 所示。

表 7 排班规划阶段伪代码

排班规划阶段伪代码
Input：机组人员信息、航班信息
Output：机组排班信息、可行航班数、执勤成本、飞行成本
Initialize：读取数据，昨天剩余人数=0，昨天机场人员分布=（1×NumStn）int
01.for d=1:NumD
02. 该天有 team（d）个组
03. if 该天组里有双向组
04. 判断双向今天状态是开始，持续，还是结束
05. end
06. 今日双向组数为 dualteam（d）
07. 使用机组人员数=2*（team（d）+dualteam（d））
08. 今天剩余人数=总人数-使用机组人员数
09. if 该天还剩余有单体航班
10. 分类：可以今天乘机的 and 不能今天乘机的
11. end
12. for 不能今天乘机的单体航班

```
    记录其起飞机场为 X
    if 昨天剩余人数>=2 且昨天有前往 X 机场的航班
        选择前往 X 机场的航班用时最短的航班，2 人乘机
        昨天剩余人数=昨天剩余人数-2
    elseif 昨天剩余人数<2 且昨天有前往 X 机场的航班
        寻找昨天间隔期为 40min 的两个小组连接起来
        选择前往 X 机场的航班用时最短的航班，2 人乘机
    end
end
for 可以今天乘机的单体航班
    计算昨日乘机成本 YTCost
    计算今日乘机成本 TCost
    选择 TCost 与 YTCost 中成本小的，应用方案
end
确认昨日方案
更新昨天机场人员分布，更新昨天剩余人数
end
```

6.2.3 人员资格分配

该算法最后是人员资格分配阶段，在上一阶段确定完每日所需人数后，首先检查连续执勤间的休息时长约束是否能够满足。若满足时，考虑尽可能减少替补资格使用次数的目标，按照**“副机长+具备替补资格的主机长”优先分配策略**进行人员分配，当所需机组人员不够时再分配“副机长+不具备替补资格的主机长”的组合，最后考虑对“选择使用替补资格+不具备替补资格的主机长”组合人员的排班。

6.3 求解结果与分析

6.3.1 A 组小规模数据结果展示

针对问题二的 A 组小规模数据，利用考虑执勤因素的基于规则的启发式算法对其进行求解，具体输出结果如表 8 所示。

表 8 问题二小规模数据求解结果

结果指标	子问题 2-数据集 A
不满足机组配置航班数	0
满足机组配置航班数	206
机组人员总体乘机次数	20
替补资格使用次数	0

机组总体利用率	85.81%
最小/平均/最大 一次执勤飞行时长	1.25/3.51/6.92
最小/平均/最大 一次执勤时长	1.25/4.17/8.92
最小/平均/最大 机组人员执勤天数	14/14.19/15
总体执勤成本（万元）	53.59
程序运行分钟数	0.01

问题二引入执勤概念并加入了对总执勤成本以及机组人员间执勤时长平衡问题的考虑，将问题二的结果与问题一进行比较发现，在小规模数据下，问题二依旧能够使所有航班都能够满足机组资格配置要求同时没有使用替补资格，但不同的是，机组人员的总体乘机次数较原来产生了增加，这和问题二优先考虑执勤成本及算法中所选择的“双向选择航班”策略有关，该策略意味着在成对航班出现多次时，牺牲乘机次数来换取连接时间的大大缩短，从而总体上使执勤成本得到降低。此外，机组总体利用率较高，最大与最小执勤飞行时长及执勤时长的差距略大，机组人员间执勤天数差距较小，可以归结于算法中将其作为优先级较前的目标进行考虑。最后，总体执勤成本为 53.59 万元，程序运行时间为 0.01 分钟。

6.3.1 B 组大规模数据结果展示

针对 B 组大规模数据，利用考虑执勤因素的基于规则的启发式算法同时考虑多基地拆分策略对该问题进行求解，具体输出结果如表 9 所示。

表 9 问题二大规模数据求解结果

结果指标	子问题 2-数据集 B
不满足机组配置航班数	305
满足机组配置航班数	13649
机组人员总体乘机次数	1474
替补资格使用次数	22
机组总体利用率	0.6931
最小/平均/最大 一次执勤飞行时长	1.83/3.17/8.83
最小/平均/最大 一次执勤时长	1.83/4.34/12
最小/平均/最大 机组人员执勤天数	13/27.10/31
总体执勤成本（万元）	3969.8620
程序运行分钟数	0.07

在大规模数据下，与小规模类似，问题二不满足机组配置的航班数比问题一更多，且由于对执勤成本的考虑，总体乘机次数大幅度增加，机组总体利用率也因此降低，但替补资格使用次数受此影响较小。同时对于执勤飞行时长、执勤时长以及组员间执勤平衡等指

标，由于问题的复杂化，对均衡问题的考量难度较大，因此其最大最小时长差异较大。最后，计算得出总体执勤成本为 3969.8620 万元。

七、问题三：模型建立与求解

7.1 数学规划模型建立

根据题目要求及模型预处理，在问题一、二的基础上引入与任务环的相关决策变量和时间、成本上的约束，建立数学规划模型如下：

$$1)\quad \max \sum_{j\in J} s_j \tag{1}$$

$$2)\quad \min \sum_{d\in D}\sum_{i\in I_f\cup I_a}\sum_{j\in J_d} DutyCostPerHr_i \times (AT_j e_{ij} - DT_j l_{ij}) \tag{26}$$

$$\begin{aligned} 3)\quad \min \sum_{i\in I_f\cup I_a} ParingCostPerHr_i \times \{Toal\_T - \{\sum_{d\in D} 1440 r_{id} \\ + rs_{id}\left[1440(d-1) - \sum_{j\in J_{d-1}} AT_j e_{ij}\right] \\ + re_{id}\left(\sum_{j\in J_{d+1}} DT_j l_{ij} - 1440d\right)\}\} \end{aligned} \tag{47}$$

$$4)\quad \min \sum_{i\in I_f}\sum_{j\in J} x_{ij} + \sum_{i\in I_a}\sum_{j\in J}(x_{ij1} + x_{ij0}) - \sum_{j\in J}(Comp_c + Comp_f) \tag{2}$$

$$5)\quad \min(max\_duty\_time - min\_duty\_time) \tag{27}$$

$$6)\quad \min(max\_work\_time - min\_work\_time) \tag{48}$$

$$7)\quad \min \sum_{i\in I_a}\sum_{j\in J} x_{ij1} \tag{3}$$

s.t 约束(4) - (25)，(28) - (46)

$$c_{id} + v_{id} + r_{id} = 1 \qquad \forall i \in I_f \cup I_a, \forall d \in D \tag{47}$$

$$r_{id+1} = r_{id} + rs_{id+1} - re_{id} \qquad \forall i \in I_f \cup I_a, \forall d \in D \tag{48}$$

$$rs_{id}\sum_{j\in J_{d-1}} e_{ij}\times EP_{ij}=rs_{id} \qquad \forall i\in I_f\cup I_a,\forall d\in D \tag{49}$$

$$\sum_{d}^{d+MaxSuccOn} v_{id}\le MaxSuccOn \qquad \forall i\in I_f\cup I_a \tag{50}$$

$$rs_{id}-r_{id+u}\le 0 \qquad \forall i\in I_f\cup I_a,\forall d\in D,\ u=0,1,\dots,MinVacDay-1 \tag{51}$$

$$Toal\_T-\{\sum_{d\in D}1440r_{id}+rs_{id}\left[1440(d-1)-\sum_{j\in J_{d-1}}AT_je_{ij}\right]+re_{id}\left(\sum_{j\in J_{d+1}}DT_jl_{ij}-1440d\right)\}\le MaxTAFB \qquad \forall i\in I_f\cup I_a \tag{52}$$

$$\max\_work\_time\ge Toal\_T$$

$$-\{\sum_{d\in D}1440r_{id}+rs_{id}\left[1440(d-1)-\sum_{j\in J_{d-1}}AT_je_{ij}\right]+re_{id}\left(\sum_{j\in J_{d+1}}DT_jl_{ij}-1440d\right)\} \qquad \forall i\in I_f\cup I_a \tag{53}$$

$$\min\_work\_time\le Toal\_T$$

$$-\{\sum_{d\in D}1440r_{id}+rs_{id}\left[1440(d-1)-\sum_{j\in J_{d-1}}AT_je_{ij}\right]+re_{id}\left(\sum_{j\in J_{d+1}}DT_jl_{ij}-1440d\right)\} \qquad \forall i\in I_f\cup I_a \tag{54}$$

$$c_{id},r_{id},rs_{id},re_{id}\in\{0,1\} \qquad \forall i\in I_f\cup I_a,d\in D \tag{55}$$

目标函数（1）、（2）、（3）、（26）和（27）分别对应问题一和二中的优化目标，而（47）和（48）为问题三需要新加入的优化目标，与问题二目标函数形式相似，分别表示了最小化机组人员的总体任务环成本以及尽可能平衡机组人员之间的任务环时长。与问题二同理，为了有效避免目标函数非线性，利用极差对机组人员间任务环时长的平衡问题进行评价，极差越小表示人员间任务环时长越均衡。由于与问题一和问题二存在连贯性，问题三沿用了问题一和二的所有约束。此外，约束（47）表示对于机组人员i在一天内只能处于执勤、休息、休假三种状态之一。约束（48）表示相邻两天休假状态与开始休假时间和结束休假时间之间的关系。约束（49）表示若机组人员i当天开始休假，则前一天最后一趟航班的降落机场一定为机组人员i自身的基地。约束（50）表示每个机组人员连续执勤天数不能超过$MaxSuccOn$天。约束（51）表示每个机组人员相邻两个任务环之间至少有$MinVacDay$天休息。约束（53）和（54）表示为找到任务环时间最长和最短的线性化公式。约束（55）表示该模型中决策变量都为 0-1 变量。

7.2 模型线性化处理

由 7.1 节数学规划模型可知，约束（49）、约束（52）-（54）是涉及决策变量相乘的非线性约束。考虑决策变量类型都为 0-1 变量，因此对其进行线性化处理，处理方法如下：

对于约束（49）引入二元辅助变量$a_{ijd} = rs_{id} \times e_{ij}$并加入以下约束：

$$a_{ijd} \leq rs_{id};$$

$$a_{ijd} \leq e_{ij};$$

$$a_{ijd} \geq rs_{id} + e_{ij} - 1$$

则约束（49）变为

$$\sum_{j \in J_{d-1}} a_{ijd} \times EP_{ij} = rs_{id} \qquad \forall i \in I_f \cup I_a, \forall d \in D$$

$$a_{ijd} \in \{0,1\} \qquad \forall i \in I_f \cup I_a, \forall j \in J, \forall d \in D$$

由于约束（52）-（54）中涉及与上述相同的决策变量的乘积，因此线性化过程同理。

7.3 模型求解

本问题中引入任务环概念，包括任务环时长约束、任务环内连续执勤天数约束，任务环之间休假时长约束，目标函数增加了对任务环成本和任务环时长平衡问题的评价。根据目标函数的优先级，该问题首先考虑目标为最大化满足机组配置的航班数，第二目标依旧是尽可能追求执勤成本的最小化，新加入的第三目标是任务环成本的最小化。

针对第一目标，算法在前两问求得回路绑定、分组绑定和执勤绑定的基础上，**基于满足执勤时长约束下一个执勤长度尽可能长而一天内执勤次数尽可能少的思想，获得了每日执勤数量下限和其对应的执勤表；同时基于一天内执勤次数尽可能多的思想，以每个绑定回路为一个执勤，获得了每日执勤数量上限和其对应的执勤表**。将执勤组合成为任务环时，在满足任务环约束情况下依次增加单日执勤数量上限，只有单日可添加的执勤数量小于该日执勤数量下限时，才产生不满足机组配置的航班数，这样设计可以尽可能地优先最大化可行航班数。

针对第二目标和第三目标，两个目标都是最小化成本，而考虑到最小化执勤成本的优先级更高，**算法的优化方向为尽可能增加执勤次数以减少连接时间，并可以为此而增加休息时间。**同时算法与第二问相同，为了尽可能减少执勤时间而优先安排前一日乘机来满足本日的单体未编组航班。

在将执勤组合成为任务环时，**算法中根据任务环和执勤状态的约束设计了环路迭代规则表**，在满足任务环约束情况下依次增加单日执勤数量，直到达到单日执勤数量上限；如果最多可分配执勤数达到了上限与下限中间的某个值，则根据这个数值重新由回路表生成新的执勤表，这个新的执勤表即为实际执勤表。最后通过一个执勤成本最低的贪心算法将人员分配到对应的任务环。表 10 展示了该问题算法的基本思路。算法流程图如图 3 所示。

表 10 考虑任务环因素的基于规则的启发式算法主要思路

考虑任务环因素的基于规则的启发式算法
（1）**回路绑定：**与问题一中的“回路绑定”步骤相同。

（2）**分组绑定：**与问题一中的“分组绑定”步骤相同。
（3）**执勤绑定：**与问题一中的“执勤绑定”步骤相同。
（4）**单独未编组航班处理：**针对未编组情况，考虑最小化执勤成本并满足最多航班数，设置三种自上而下依次优先的处理方式。
•对于本日无法编入执勤的单独未编组航班，从之前天数中找到一个成本最小的乘机方案，与本日单独未编组航班组成跨日执勤。
• 对于本日可以编入执勤的单独未编组航班，分别计算本日执勤与跨日执勤两种方案，本日执勤是从本日找到一个成本最小乘机方案，跨日执勤从之前天数中找到一个成本最小的乘机方案。
（5）**确认日执勤上下限：**以回路表和未编组处理表合并作为日执勤上限，以预执勤表与未编组处理表合并作为执勤下限，预处理表中的根据上下限倾向选择是跨日执勤还是本日执勤。在此环节得到每日可存在的可行执勤数的一个上下限。

（6）**制定环路迭代规则表：**制定环路迭代规则表，表中记录了一个四维向量（a,b,c,d）为一个环路状态，a 代表当前已连续执勤天数，b 代表当前任务环存已在天数，c 代表当前已休息或休假天数，d 表示当前可进行的状态（0 为不可执勤，1 为可继续在任务环中执勤，2 为可开启任务环），根据任务环与执勤相关约束，对每一种状态生成分别选择执勤或休息的下一状态，归纳所有迭代规则得到一个环路迭代规则表。

（7）**预组合任务环：**确定最多机组人员对数 P，生成 P 个初始环路状态（0,0,2,2），根据每日执勤上限，依据优先度规则：【正在执行的环路（连续执勤天数从多到少）休息 1 天还可以继续的环路→休假完成的回路】，依次给本日预分配机组人员数量，最高直到达到本日执勤上限。

（8）**确定真实执勤表：**如果预分配数量在执勤上下限之间，则根据环路表和预执勤表进行组合与拆解，生成真实执勤表；否则如果预分配数小于执勤下限，则根据预执勤表生成含有不可行航班的真实执勤表。

（9）**分配机组人员：**把虚拟人员负责的任务环表按照执勤数量从高到低排列，依次根据真实执勤表按照用时从高到底向任务环表中插入执勤方案，直到完成所有任务环的执勤安排。最后将所有机组人员组合成单位成本从低到高的人员配置，将单位成本从低到高的人员配置依次插入执勤数量从高到低的任务环表，整个过程需要满足执勤信息时间的约束。
（10 **输出结果：**满足所有约束条件和规则后，输出机组人员分配方案，以及满足机组配置的航班数、总体乘机次数和替补资格使用次数。

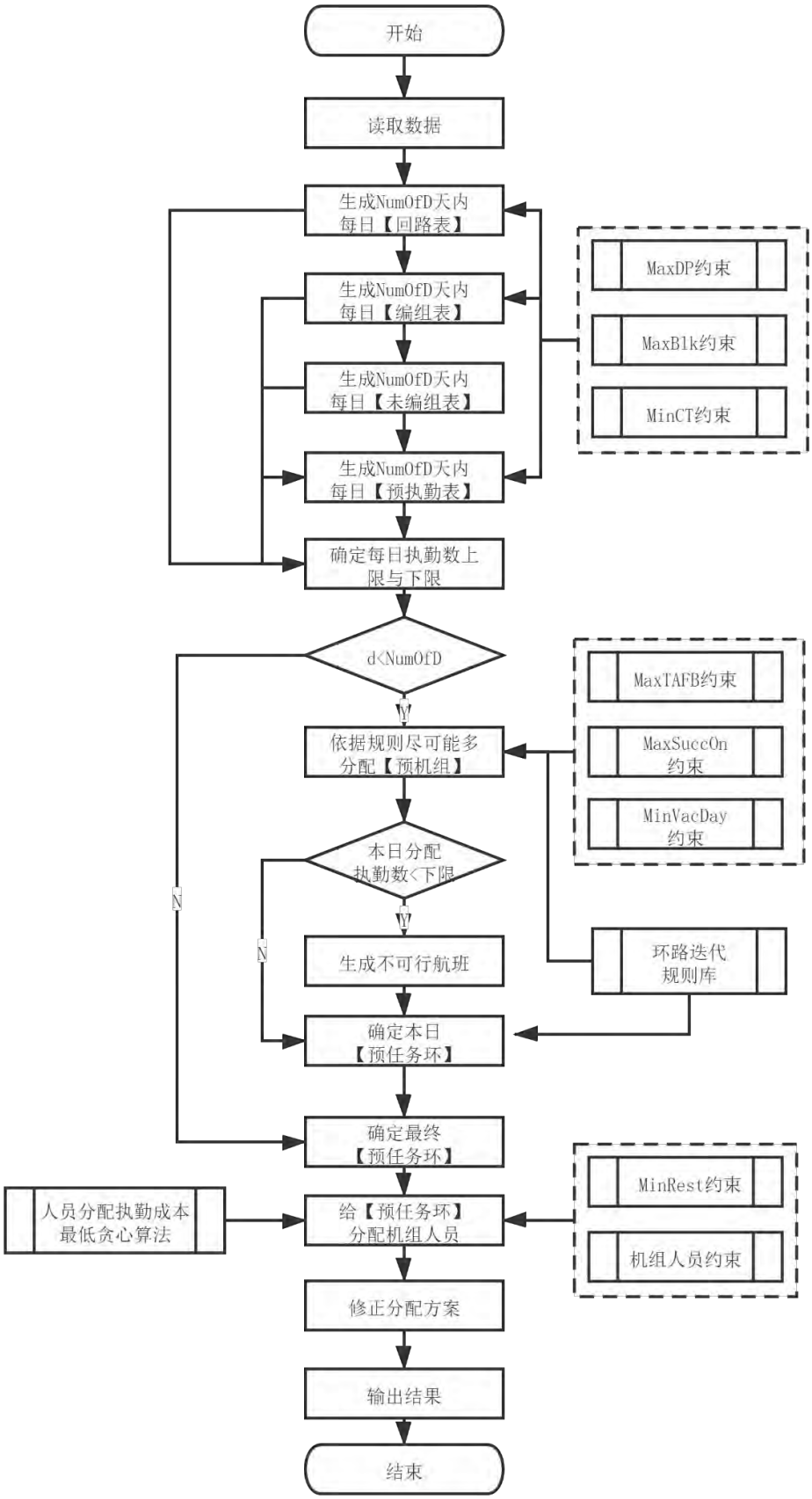

图 3 考虑任务环的基于规则的启发式算法流程图

7.4 求解结果与分析

7.4.1 A 组小规模数据结果展示

针对问题三的 A 组小规模数据，利用考虑任务环因素的基于规则的启发式算法对其进行求解，具体输出结果如表 11 所示。

表 11 问题三小规模数据求解结果

结果指标	子问题 3-数据集 A
不满足机组配置航班数	0
满足机组配置航班数	206
机组人员总体乘机次数	8
替补资格使用次数	0
机组总体利用率	82.07%
最小/平均/最大 一次执勤飞行时长	1.25/3.57/6.92
最小/平均/最大 一次执勤时长	1.25/4.26/8.92
最小/平均/最大 机组人员执勤天数	8/10/12
一/二/三/四天 任务环数量（连续执勤）分布	0/14/2/44
总体执勤成本（万元）	56.03
总体任务环成本（万元）	2.18
程序运行分钟数	0.03

问题三引入任务环概念并加入了对总任务环成本以及机组人员间任务环时长平衡问题的考虑，将问题三的结果与问题一、二进行比较发现，在小规模数据下，问题三依旧能够使所有航班都能够满足机组资格配置要求同时没有使用替补资格，与问题二不同的是，机组人员的总体乘机次数更少且等于问题一的次数，因为问题三和问题一都类似地只考虑了单独未编航班的乘机可能，因此机组总体利用率也较问题二有所提高。由于任务环中对机组人员休假的考虑，因此总体执勤成本较问题一、二来说会有所增加。总任务环成本作为执勤成本外的出差补贴，具有较小的数额。

（注：由于任务环分布不仅仅限于一到四天的分布，因此仅标注四天以内的任务环分布不能完整体现任务环的全部状态，同时考虑到连续执勤的分布是在一、二、三、四个指标间分布的，因此认为输出连续执勤的分布更能真实全面地体现解的质量。）

7.4.2 B 组大规模数据结果展示

针对 B 组大规模数据，利用考虑任务环因素的基于规则的启发式算法同时考虑多基地拆分策略对该问题进行求解，具体输出结果如表 12 所示。

表 12 问题三大规模数据求解结果

结果指标	子问题 3-数据集 B
不满足机组配置航班数	1543
满足机组配置航班数	12411
机组人员总体乘机次数	1587
替补资格使用次数	49
机组总体利用率	0.65
最小/平均/最大 一次执勤飞行时长	1.83/4.13/8.33
最小/平均/最大 一次执勤执勤时长	1.83/5.84/12
最小/平均/最大 机组人员执勤天数	13/21.09/22
一/二/三/四天 任务环数量（连续执勤）分布	61/894/207/1833
总体执勤成本（万元）	3857.6350
总体任务环成本（万元）	163.0671
程序运行分钟数	0.21

由表可知，在大规模数据下，与问题一、二相比，由于任务环的复杂性及启发式算法的局限性，各评价指标均有下降，同时程序运行时间也明显增加。

八、模型评价与改进

8.1 算法的有效性与复杂度分析

本文采用基于规则的启发式算法寻找满足约束条件与优化目标的解，算法中融入了贪心算法的思想[5]，将每日的航段进行不同层级的组合。第一个层级是回路，首先根据航段往往存在连续往返的特性，将从基地出发并最终返回基地，将这些航段组合为一个回路，这些回路在绝大部分情况下就是最优解。

第二个层级是小组，当我们有必要寻找到一个连续的长时间执勤时，以最短连接时间（40min）或者近似最短连接时间（45min）连接在一起的一些回路，往往具有最优性，因此我们提前找到这些可能的小组并存储入表中，这有助于我们需要的时候进行调用。

第三个层级是执勤，在第二问与第三问将会用到这个概念。这意味着一个机组人员配置一天执行的所有任务序列，它既是小组、回路与航班组合而成的最终集合，也是后续组成任务环的基本单位，起到承前启后的作用，决定着解的优劣程度，因此对于执勤的确定在算法中是十分谨慎的，往往在不同阶段根据成本比较与可行性判断而组合成为不同的执勤，称之为预执勤。

执勤的决定是灵活且易于调整的，它不会因过早地确定而降低了搜索空间从而导致解的不优，而是提前在一个较大的范围区间内波动以适应前后约束的变化。执勤的这种强可调整性，有助于我们在求到更多可行航班与减少成本两个目标之间适时调整，这使得算法在一定程度上摆脱了贪心算法的局部性，能够得到一个相对于全局的较优解。

第四个层级是任务环，其中第三问用到了此概念。在算法中同样是逐步收敛任务环的求解区间的，我们首先确定了一群达到最高上限的虚拟机组人员配置，同时用这些虚拟机组人员配置来满足每天的可变数量的执勤安排。这使得我们从一个最优解的下界出发，在满足约束的情况下不断逼近一个可行全局较优解，这使得模型算法具有了一定的优势。

对于子问题一的空间复杂度为$O(nd + np)$，时间复杂度约为$O(dp \cdot n!)$。

对于子问题二的空间复杂度为$O(nd + np)$，，时间复杂度约为$O(dp \cdot n!)$。

对于子问题三的空间复杂度为$O(nd + np)$，，时间复杂度约为$O(dp \cdot n^2)$。

8.2 模型稳定性与灵敏度分析

本文所设计的算法为基于规则的启发式算法，其不具有随机性，所以多次重复运行可得到相同的结果。本文算法在两种数据集上均取得优良的结果，可见算法具有很强的稳定性。

本文所涉及的超参数仅有连接时间阈值上限ε，故对此参数进行灵敏度分析如图 4 所示。这里以问题 2 在数据 B 上的实验结果为例。

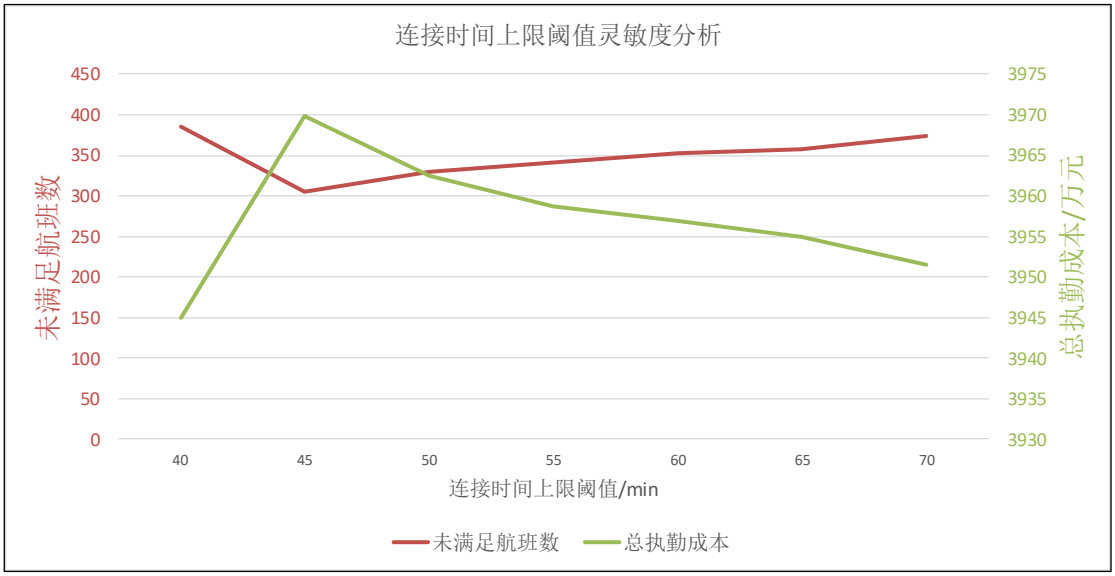

图 4 连接时间阈值上限ε灵敏度分析

图中红色折线与绿色折线分别代表了未满足航班数与总执勤成本随连接时间上限阈值的变化情况。当该阈值从 40min 增加到 45min 时，未满足航班数减少，原因是较小的阈值阻碍了一些可能的连接，从而形成更多的分组，来分配给更多的机组人员，所以造成机组人员不足，部分航班无法满足。而较大的阈值可以形成更多的连接，减少机组人员的需求数，从而满足更多航班。与此同时，满足航班更多时，执勤成本也会随之提高。

当阈值从 45min 增加时，未满足航班数随之增加，这是由于较大的阈值引起了很多无效的连接，如将相邻过远的两个航班相连，造成它们中间的一些航班无法被满足。并且执勤成本由于机组人员去到了更少的航班也出现减少。

由于问题 2 的主要目标为优化未满足航班数，所以选择效果最好的 45min 作为最优参数。

8.3 优点分析

(1) 本文在建立各问题的 0-1 整数线性规划模型时，提前对数据进行整理，利用得到的EP_{ij}与φ_j等参数与集合，缩减了变量的维度、约束的数目，从而极大加速了模型的求解。

(2) 本文结合问题特征针对各问题分别提出了有效的启发式规则，可以在短时间内得到高质量的结果，通过将航段组合为回路、小组、执勤等单位来简化问题的求解过程，同时保留执勤的灵活性，以适用于不同的目标函数。

8.4 缺点分析

(1) 本文建立的问题 3 的线性规划模型，需要对 01 变量相乘进行线性化处理，从而增大了模型的维度。

(2) 本文所设计的算法为启发式规则，无法保证最优性。

8.5 模型改进

(1) 在 9.2 节的灵敏度分析中，本文分析了未满足航班数与总执勤成本两个目标受连接时间阈值上限参数的影响，根据目标优先级确定了最优的设定值为 45min。后续的优化可以将不同等级的单位（回路、小组等）连接时的阈值上限设定为不同的值，从而更精细地

控制航段间的组合。

(2) 本文建立的数学规划模型可看作网络流模型，即$z_{ijj'}$代表了组员i在航段组成的图上，是否从j走到j'，显然各个航段间并不是全连接的，所以在建立模型时，可以提前对航段的组合进行筛选以减少变量。另外，保证航段间机场相连的约束很难直接写成线性模型，故通过根据数据生成的EP_{ij}参数，写成线性约束，使得模型得以求解。

参考文献

[1] 向杜兵. 航空机组排班计划一体化优化研究[D].北京交通大学,2020.
[2] Saeed Saemi, Alireza Rashidi Komijan, Reza Tavakkoli-Moghaddam and Mohammad Fallah, A new mathematical model to cover crew pairing and rostering problems simultaneously, Journal of Engg. Research Vol. 9 No. (2) June 2021 pp. 218-233
[3] 张米. 航空公司机组排班模型研究[D].清华大学,2014.
[4] K.Bhaskar, G.Srinivasan. Static and dynamic operator allocation problems in cellular manufacturing systems. International Journal of Production Research, 35(1997), pp.3467-3481
[5] 司守奎, 孙玺箐. 数学建模算法与应用 [M].北京:国防工业出版社,2011:306- 307.

附录（代码及注释）

Matlab 代码

```
clc;
clear all;
%% 读取数据
filename = '机组排班 Data A-Flight.csv'; %文件

[~,DptrDD] = xlsread(filename,1,'B2:B207');%读取
DptrT = xlsread(filename,1,'C2:C207');%读取
[~,DptrStnn] = xlsread(filename,1,'D2:D207');%读取
[~,ArrvDD] = xlsread(filename,1,'E2:E207');%读取
ArrvT = xlsread(filename,1,'F2:F207');%读取
[~,ArrvStnn] = xlsread(filename,1,'G2:G207');%读取

NumOfFli=length(DptrT); %flight 数量
DptrD=zeros(NumOfFli,1);
ArrvD=zeros(NumOfFli,1);
FlightT=zeros(NumOfFli,1);
StnTableA={"NKX" "CTH" "PDK" "PGX" "PLM" "PXB" "XGS"};
FlightStn=zeros(NumOfFli,2);
NumOfStn=length(StnTableA);

for i=1:NumOfFli
    Temp=strsplit(DptrDD{i},'/');
    DptrD(i)=str2double(Temp{2});
    Temp=strsplit(ArrvDD{i},'/');
    ArrvD(i)=str2double(Temp{2});
end
ArrvD=ArrvD-min(DptrD)+1;
DptrD=DptrD-min(DptrD)+1;
NumOfD=max(DptrD);

for i=1:NumOfFli
    if DptrD(i)==ArrvD(i)
        FlightT(i)=round((ArrvT(i)-DptrT(i))*24*60);
    elseif DptrD(i)+1==ArrvD(i)
        FlightT(i)=round((ArrvT(i)-DptrT(i)+1)*24*60);
    else
        print("错误 1");
    end
```

```
39  end
40
41  for i=1:NumOfFli
42      if DptrT(i)<ArrvT(i)
43          ArrvT(i)=round(ArrvT(i)*24*60);
44          DptrT(i)=round(DptrT(i)*24*60);
45      elseif DptrT(i)>ArrvT(i)
46          ArrvT(i)=round((ArrvT(i)+1)*24*60);
47          DptrT(i)=round(DptrT(i)*24*60);
48      end
49  end
50
51  for i=1:NumOfFli
52      for i2= 1:NumOfStn
53          if DptrStnn{i}==StnTableA{i2}
54              FlightStn(i,1)=i2;
55          end
56          if ArrvStnn{i}==StnTableA{i2}
57              FlightStn(i,2)=i2;
58          end
59      end
60  end
61
62  %A 数据
63  StnPop=zeros(NumOfStn,3); %各个机场现有人员数量,1 列为正，2 列为皆可，3
    列为副
64  StnPop(1,:)=[5,6,10];
65
66  MaxBlk=660;
67  MaxDP=720;
68
69
70
71  %% 初始化
72  group=cell(NumOfD,1);
73  Duty=cell(NumOfD,1);
74  DutySingle=cell(NumOfD,1);
75  DutyPlus=cell(NumOfD,1);
76  NumOfTheDay=zeros(NumOfD,1);
77  NumOfGroup=zeros(NumOfD,1);
```

```
NumOfDuty=zeros(NumOfD,1);
NumOfDutySingle=zeros(NumOfD,1);
NumOfDutyPlus=zeros(NumOfD,1);
TheDayTable=cell(NumOfD,1);
TDT=cell(NumOfD,1);
chengjiNum=zeros(NumOfFli,1);

for d=1:NumOfD
    %生成该天要做的航段任务表
    findex=find(DptrD==d);
    NumOfTheDay(d)=length(findex);
    TheDayTable{d}=zeros(NumOfTheDay(d),5);
    TheDayTable{d}(:,1)=findex;
    TheDayTable{d}(:,[2,3])=FlightStn(findex,:);
    TheDayTable{d}(:,4)=DptrT(findex,:);
    TheDayTable{d}(:,5)=ArrvT(findex,:);
    TheDayTable{d}(:,6)=FlightT(findex,:);

    TDT{d}=cell(NumOfStn,1);
    for i=1: NumOfStn
        TDT{d}{i}=TheDayTable{d}(TheDayTable{d}(:,2)==i,:);
    end
end

%% 生成组 group

usedid=cell(NumOfD,1);
for d=1:NumOfD
    for i= 1:size(TDT{d}{1},1)
        targetstn=TDT{d}{1}(i,3);
        findgroup=0;
        findgroup2=0;
        for i2= 1:size(TDT{d}{targetstn},1)
            if TDT{d}{targetstn}(i2,3)~=1
                continue
            end
            jiangeT=TDT{d}{targetstn}(i2,4)-TDT{d}{1}(i,5);
            if jiangeT==40
                NumOfGroup(d)=NumOfGroup(d)+1;

```

```
                group{d}{NumOfGroup(d),1}=[TDT{d}{1}(i,:);TDT{d}{targetstn}(i2,:)];
118             group{d}{1,2}(NumOfGroup(d),:)=[TDT{d}{1}(i,1),TDT{d}{targetstn}(i2,1),...
119             TDT{d}{1}(i,4),TDT{d}{targetstn}(i2,5),jiangeT];
120             usedid{d}=[usedid{d},group{d}{1,2}(NumOfGroup(d),[1,2])];
121                 findgroup=findgroup+1;
122                 break;
123             elseif jiangeT>=40&&jiangeT<=45
124             tempgroup1=[TDT{d}{1}(i,:);TDT{d}{targetstn}(i2,:)];
125             tempgroup2=[TDT{d}{1}(i,1),TDT{d}{targetstn}(i2,1),...
126             TDT{d}{1}(i,4),TDT{d}{targetstn}(i2,5),jiangeT];
127                 findgroup2=findgroup2+1;
128             end
129         end
130         if findgroup==0&&findgroup2>=1
131             NumOfGroup(d)=NumOfGroup(d)+1;
132             group{d}{NumOfGroup(d),1}=tempgroup1;
133             group{d}{1,2}(NumOfGroup(d),:)=tempgroup2;
134             usedid{d}=[usedid{d},group{d}{1,2}(NumOfGroup(d),[1,2])];
135         end
136     end
137 end
138
139
140 %% 生成 duty
141 for d=1:NumOfD
142     Duty{d}=cell(1,1);
143     [~,gweizhi]=sort(group{d}{1,2}(:,3));
144     usedgroup=zeros(1,NumOfGroup(d))+1; %1 是还没用，可用
145     for i=1:NumOfGroup(d)
146         if usedgroup(gweizhi(i))~=0
147             nowi=i;
```

```
148             NumOfDuty(d)=NumOfDuty(d)+1;
149             usedgroup(gweizhi(i))=0;
150             Duty{d}{NumOfDuty(d),1}=[];
151 
Duty{d}{NumOfDuty(d),1}=[Duty{d}{NumOfDuty(d),1};group{d}{gweiz
hi(i),1}];
152 
Duty{d}{NumOfDuty(d),2}(1,1)=group{d}{gweizhi(i),1}(1,4);
153 
Duty{d}{NumOfDuty(d),2}(1,2)=group{d}{gweizhi(i),1}(2,5);
154 
Duty{d}{NumOfDuty(d),2}(1,3)=group{d}{gweizhi(i),1}(2,4)-...
155                 group{d}{gweizhi(i),1}(1,5);
156 
Duty{d}{NumOfDuty(d),2}(1,4)=Duty{d}{NumOfDuty(d),2}(1,2)-...
157 
Duty{d}{NumOfDuty(d),2}(1,1)-Duty{d}{NumOfDuty(d),2}(1,3);
158             for i2=i+1:NumOfGroup(d)
159                 if
(group{d}{1,2}(gweizhi(nowi),4)-group{d}{1,2}(gweizhi(i2),3)==-
40 ...
160 
||group{d}{1,2}(gweizhi(nowi),4)-group{d}{1,2}(gweizhi(i2),3)==
-40)...
161 
&&group{d}{gweizhi(i2),1}(2,5)-Duty{d}{NumOfDuty(d),2}(1)<=720
162                     usedgroup(gweizhi(i2))=0;
163 
Duty{d}{NumOfDuty(d),1}=[Duty{d}{NumOfDuty(d),1};group{d}{gweiz
hi(i2),1}];
164 
Duty{d}{NumOfDuty(d),2}(1,2)=group{d}{gweizhi(i2),1}(2,5);
165 
Duty{d}{NumOfDuty(d),2}(1,3)=Duty{d}{NumOfDuty(d),2}(1,3)+...
166 
group{d}{gweizhi(i2),1}(2,4)-group{d}{gweizhi(i2),1}(1,5)...
167 
+group{d}{gweizhi(i2),1}(1,4)-group{d}{gweizhi(i),1}(2,5);
168 
Duty{d}{NumOfDuty(d),2}(1,4)=Duty{d}{NumOfDuty(d),2}(1,2)-...
```

```
Duty{d}{NumOfDuty(d),2}(1,1)-Duty{d}{NumOfDuty(d),2}(1,3);
                    nowi=i2;
                end
            end
        end
    end
end

%% dutysigle&&dutyplus
for d=1:NumOfD
    usedid{d}=sort(usedid{d});
    for i=1:NumOfTheDay(d)
        if sum(TheDayTable{d}(i,1)==usedid{d})==0
            NumOfDutySingle(d)=NumOfDutySingle(d)+1;
DutySingle{d}(NumOfDutySingle(d),:)=TheDayTable{d}(i,:);
        end
    end
end

bixushangci=cell(NumOfD,1);
for d=1:NumOfD
    if NumOfDutySingle(d)>0
        bixushangci{d}=zeros(1,NumOfDutySingle(d))+1;
        for i=1:NumOfDutySingle(d) %找必须上次
            X=DutySingle{d}(i,2);
            toX=find(TheDayTable{d}(:,3)==X);
            toXNum=length(toX);
            for i2=1:toXNum
              if DutySingle{d}(i,4)-TheDayTable{d}(toX(i2),5)>=40
                  bixushangci{d}(i)=0;
              end
            end
        end

        for i=1:NumOfDutySingle(d)
            if bixushangci{d}(i)==1 %先做上次必须
               X=DutySingle{d}(i,2);
```

```
207                 YtoX=find(TheDayTable{d-1}(:,3)==X);
208                 YtoXNum=length(YtoX);
209                 over=0;
210                 [~,YtoXID]=sort(TheDayTable{d-1}(YtoX,6));
211                 for i2=1:YtoXNum
212                     if chengjiNum(TheDayTable{d-1}(YtoX(YtoXID(i2)),1))<=3
213                 chengjiNum(TheDayTable{d-1}(YtoX(YtoXID(i2)),1))=chengjiNum(TheDayTable{d-1}(YtoX(YtoXID(i2)),1))+2;
214                         NumOfDutyPlus(d-1)=NumOfDutyPlus(d-1)+1;
215                 DutyPlus{d-1}(NumOfDutyPlus(d-1),:)=zeros(1,8);
216                 DutyPlus{d-1}(NumOfDutyPlus(d-1),[1:6])=TheDayTable{d-1}(YtoX(YtoXID(i2)),:);
217                         DutyPlus{d-1}(NumOfDutyPlus(d-1),7)=1;
218                         DutyPlus{d-1}(NumOfDutyPlus(d-1),8)=1;
219                         NumOfDutyPlus(d)=NumOfDutyPlus(d)+1;
220                         DutyPlus{d}(NumOfDutyPlus(d),:)=zeros(1,8);
221                 DutyPlus{d}(NumOfDutyPlus(d),[1:6])=DutySingle{d}(i,:);
222                         DutyPlus{d}(NumOfDutyPlus(d),7)=2;
223                         DutyPlus{d}(NumOfDutyPlus(d),8)=1;
224                         over=1;
225                         break;
226                     end
227                 end
228                 if over==0
229                     print("error2")
230                 end
231             end
232         end
233
234         for i=1:NumOfDutySingle(d)
235             if bixushangci{d}(i)==0 %后坐无所谓的
236                 X=DutySingle{d}(i,2);
237                 YtoX=find(TheDayTable{d-1}(:,3)==X);
238                 YtoXNum=length(YtoX);
239                 over=0;
```

```
                [~,YtoXID]=sort(TheDayTable{d-1}(YtoX,6));
                for i2=1:YtoXNum
                    if chengjiNum(TheDayTable{d-1}(YtoX(YtoXID(i2)),1))<=3
chengjiNum(TheDayTable{d-1}(YtoX(YtoXID(i2)),1))=chengjiNum(TheDayTable{d-1}(YtoX(YtoXID(i2)),1))+2;
                        NumOfDutyPlus(d-1)=NumOfDutyPlus(d-1)+1;
DutyPlus{d-1}(NumOfDutyPlus(d-1),:)=zeros(1,8);
DutyPlus{d-1}(NumOfDutyPlus(d-1),[1:6])=TheDayTable{d-1}(YtoX(YtoXID(i2)),:);
                        DutyPlus{d-1}(NumOfDutyPlus(d-1),7)=1;
                        DutyPlus{d-1}(NumOfDutyPlus(d-1),8)=0;
                        NumOfDutyPlus(d)=NumOfDutyPlus(d)+1;
                        DutyPlus{d}(NumOfDutyPlus(d),:)=zeros(1,8);
DutyPlus{d}(NumOfDutyPlus(d),[1:6])=DutySingle{d}(i,:);
                        DutyPlus{d}(NumOfDutyPlus(d),7)=2;
                        DutyPlus{d}(NumOfDutyPlus(d),8)=1;
                        over=1;
                        break;
                    end
                end
                if over==0
                    print("error2")
                end
            end
        end


    end
end


%% 第三问
P=10;
global pstatus
pstatus=zeros(P,NumOfD);
```

```
global Pstatus
Pstatus=zeros(4,P); %已执勤几天，已在环几天，已休息的第几天,是否可用(0休息，1环中还可以工作，2休息好未工作)
Pstatus(3,:)=zeros(1,P)+2;
Pstatus(4,:)=zeros(1,P)+2;
pable=sum(Pstatus(4,:)>0);
global pold
global poldnum
pold=zeros(5,P); %01,p是否已工作x天
poldnum=zeros(4,1);

abletoplus=NumOfGroup-NumOfDuty;
realplus=zeros(NumOfD,1);

for i=1:3
    pold(i,:)=(Pstatus(4,:)==1).*(Pstatus(1,:)==i);
    poldnum(i)=sum(pold(i,:));
end
pold(4,:)=Pstatus(4,:)==2;
poldnum(4)=sum(pold(4,:));
pold(5,:)=(Pstatus(4,:)==1).*(Pstatus(1,:)==0);
poldnum(5)=sum(pold(5,:));

fenpei=zeros(d,P);
for d=1:NumOfD
    if NumOfDutyPlus(d)>0
        knum=sum(DutyPlus{d}(:,7)==1);
        tiqiannum=0;
        index=find(pold(2,:)==1);
        for i=1:poldnum(2)
            if fenpei(d,index(i))~=1
                setp(index(i),d)
                fenpei(d,index(i))=1;
                setp(index(i),d+1)
                fenpei(d+1,index(i))=1;
                tiqiannum=tiqiannum+1;
                if tiqiannum==knum
                    break;
                end
            end
```

```
312         end
313         if tiqiannum<knum
314             index=find(pold(1,:)==1);
315             for i=1:poldnum(1)
316                 if fenpei(d,index(i))~=1
317                     setp(index(i),d)
318                     fenpei(d,index(i))=1;
319                     setp(index(i),d+1)
320                     fenpei(d+1,index(i))=1;
321                     tiqiannum=tiqiannum+1;
322                     if tiqiannum==knum
323                         break;
324                     end
325                 end
326             end
327         end
328         if tiqiannum<knum
329             index=find(pold(4,:)==1);
330             for i=1:poldnum(4)
331                 if fenpei(d,index(i))~=1
332                     setp(index(i),d)
333                     fenpei(d,index(i))=1;
334                     setp(index(i),d+1)
335                     fenpei(d+1,index(i))=1;
336                     tiqiannum=tiqiannum+1;
337                     if tiqiannum==knum
338                         break;
339                     end
340                 end
341             end
342         end
343         if tiqiannum<knum
344             disp("error5")
345         end
346
347     end
348
349     % 剩下的全是单体
350     donum=0;
351     if donum<NumOfGroup(d)
```

```
    index=find(pold(1,:)==1);
    for i=1:poldnum(1)
        if fenpei(d,index(i))~=1
            setp(index(i),d)
            fenpei(d,index(i))=1;
            donum=donum+1;
            if donum==NumOfGroup(d)
                break;
            end
        end
    end
    end

    if donum<NumOfGroup(d)
    index=find(pold(2,:)==1);
    for i=1:poldnum(2)
        if fenpei(d,index(i))~=1
            setp(index(i),d)
            fenpei(d,index(i))=1;
            donum=donum+1;
            if donum==NumOfGroup(d)
                break;
            end
        end
    end
    end

    if donum<NumOfGroup(d)
    index=find(pold(3,:)==1);
    for i=1:poldnum(3)
        if fenpei(d,index(i))~=1
            setp(index(i),d)
            fenpei(d,index(i))=1;
            donum=donum+1;
            if donum==NumOfGroup(d)
                break;
            end
        end
    end
    end
```

```
392
393     if donum<NumOfGroup(d)
394     index=find(pold(5,:)==1);
395     for i=1:poldnum(5)
396         if fenpei(d,index(i))~=1
397             setp(index(i),d)
398             fenpei(d,index(i))=1;
399             donum=donum+1;
400             if donum==NumOfGroup(d)
401                 break;
402             end
403         end
404     end
405     end
406
407     if donum<NumOfGroup(d)
408     index=find(pold(4,:)==1);
409     for i=1:poldnum(4)
410         if fenpei(d,index(i))~=1
411             setp(index(i),d)
412             fenpei(d,index(i))=1;
413             donum=donum+1;
414             if donum==NumOfGroup(d)
415                 break;
416             end
417         end
418     end
419     end
420
421     if donum<NumOfDuty(d)
422         disp("error6")
423     else
424         realplus(d)=donum-NumOfDuty(d);
425     end
426
427     %其他休息
428     for p=1:P
429         if fenpei(d,p)==0
430             restp(p,d)
431             fenpei(d,p)=1;
```

```
        end
    end

end


%% 算时间
chengjitime=0;
for d=1:NumOfD
    if NumOfDutyPlus(d)>0
        for i=1:NumOfDutyPlus(d)
            if DutyPlus{d}(i,7)==1
                chengjitime=chengjitime+DutyPlus{d}(i,6);
            end
        end
    end
end
lianjietime=0;
for d=1:NumOfD
    if NumOfDuty(d)>0
        for i=1:NumOfDuty(d)
            lianjietime=lianjietime+Duty{d}{i,2}(3);
        end
    end
end
lianjietime=lianjietime-sum(realplus)*40;
ftime=sum(FlightT);
liyonglv=ftime/(ftime+lianjietime+chengjitime);
dutycost=((ftime+lianjietime+chengjitime)/60)*(640+600);

%% 真实duty

Duty=cell(NumOfD,1);
NumOfDuty=zeros(NumOfD,1);
for d=1:NumOfD
    finalplus=0;
    Duty{d}=cell(1,1);
    [~,gweizhi]=sort(group{d}{1,2}(:,3));
    usedgroup=zeros(1,NumOfGroup(d))+1; %1是还没用，可用
    for i=1:NumOfGroup(d)
```

```
472         if usedgroup(gweizhi(i))~=0
473             nowi=i;
474             NumOfDuty(d)=NumOfDuty(d)+1;
475             usedgroup(gweizhi(i))=0;
476             Duty{d}{NumOfDuty(d),1}=[];
477 Duty{d}{NumOfDuty(d),1}=[Duty{d}{NumOfDuty(d),1};group{d}{gweiz
    hi(i),1}];
478 Duty{d}{NumOfDuty(d),2}(1,1)=group{d}{gweizhi(i),1}(1,4);
479 Duty{d}{NumOfDuty(d),2}(1,2)=group{d}{gweizhi(i),1}(2,5);
480 Duty{d}{NumOfDuty(d),2}(1,3)=group{d}{gweizhi(i),1}(2,4)-...
481                 group{d}{gweizhi(i),1}(1,5);
482 Duty{d}{NumOfDuty(d),2}(1,4)=Duty{d}{NumOfDuty(d),2}(1,2)-...
483 Duty{d}{NumOfDuty(d),2}(1,1)-Duty{d}{NumOfDuty(d),2}(1,3);
484             for i2=i+1:NumOfGroup(d)
485                 if
    (group{d}{1,2}(gweizhi(nowi),4)-group{d}{1,2}(gweizhi(i2),3)==-
    40 ...
486 ||group{d}{1,2}(gweizhi(nowi),4)-group{d}{1,2}(gweizhi(i2),3)==
    -40)...
487 &&group{d}{gweizhi(i2),1}(2,5)-Duty{d}{NumOfDuty(d),2}(1)<=720
    ...
488                     && finalplus<(abletoplus(d)-realplus(d))
489                     usedgroup(gweizhi(i2))=0;
490 Duty{d}{NumOfDuty(d),1}=[Duty{d}{NumOfDuty(d),1};group{d}{gweiz
    hi(i2),1}];
491 Duty{d}{NumOfDuty(d),2}(1,2)=group{d}{gweizhi(i2),1}(2,5);
492 Duty{d}{NumOfDuty(d),2}(1,3)=Duty{d}{NumOfDuty(d),2}(1,3)+...
493 group{d}{gweizhi(i2),1}(2,4)-group{d}{gweizhi(i2),1}(1,5)...
```

```
+group{d}{gweizhi(i2),1}(1,4)-group{d}{gweizhi(i),1}(2,5);

Duty{d}{NumOfDuty(d),2}(1,4)=Duty{d}{NumOfDuty(d),2}(1,2)-...

Duty{d}{NumOfDuty(d),2}(1,1)-Duty{d}{NumOfDuty(d),2}(1,3);
                    nowi=i2;
                    finalplus=finalplus+1;
                end
            end
        end
    end
end

%% jisuan2
allnum=sum(NumOfDuty)+sum(NumOfDutyPlus);
alldutytime=zeros(allnum,1);
allflighttime=zeros(allnum,1);
k=0;
for d=1:NumOfD
    for i=1:NumOfDuty(d)
        k=k+1;
        allflighttime(k)=Duty{d}{i,2}(4);
        alldutytime(k)=Duty{d}{i,2}(4)+Duty{d}{i,2}(3);
    end
    for i=1:NumOfDutyPlus(d)
        k=k+1;
        allflighttime(k)=DutyPlus{d}(i,6);
        alldutytime(k)=DutyPlus{d}(i,6);
    end
end

answer6=[min(allflighttime),mean(allflighttime),max(allflighttime)];
answer7=[min(alldutytime),mean(alldutytime),max(alldutytime)];
answer6=answer6/60;
answer7=answer7/60;

dutytianshu=sum(pstatus,2);
answer8=[min(dutytianshu),sum(dutytianshu)*2/21,max(dutytianshu
```

```
)];

%% 计算环数量
huanlx=zeros(P,4);
for p=1:P
    do=0;
    for d=1:NumOfD-1
        if pstatus(p,d)==1
            do=do+1;
        elseif pstatus(p,d)==0 && pstatus(p,max(d-1,1))==1
            huanlx(p,do)=huanlx(p,do)+1;
            do=0;
        end
    end
    d= NumOfD;
    if pstatus(p,d)==1
        do=do+1;
        huanlx(p,do)=huanlx(p,do)+1;
    elseif pstatus(p,d)==0 && pstatus(p,max(d-1,1))==1
        huanlx(p,do)=huanlx(p,do)+1;
        do=0;
    end
end
answer9=sum(huanlx);

%% huan
scofp=cell(P,1);
huannumofp=zeros(P,1);
d=1;
for p=1:P
    if pstatus(p,d)==1
        scofp{p}=zeros(1,3);
        scofp{p}(1,1)=1;
        huannumofp(p)=1;
    end
end
for p=1:P
    do=huannumofp(p);
    if p==4
        1;
```

```
    end
    for d=2:NumOfD-1
        if pstatus(p,d)==1&&pstatus(p,d-1)==0&&pstatus(p,max(d-2,1))==0
            do=1;
            huannumofp(p)=huannumofp(p)+1;
            scofp{p}(huannumofp(p),:)=zeros(1,3);
            scofp{p}(huannumofp(p),1)=d;
        elseif pstatus(p,d)==0 && pstatus(p,d-1)==0&&huannumofp(p)~=0&&scofp{p}(huannumofp(p),3)==0
            if d~=2
                scofp{p}(huannumofp(p),2)=d-2;
                scofp{p}(huannumofp(p),3)=do;
                do=0;
            end
        elseif pstatus(p,d)==1&&pstatus(p,d-1)==0&&pstatus(p,max(d-2,1))==1
            do=do+2;
        elseif pstatus(p,d)==1
            do=do+1;
        elseif pstatus(p,d)==0
            do=do;
        else
            disp("error8");
        end
    end

    d=NumOfD;
    if pstatus(p,d)==1&&pstatus(p,d-1)==0&&pstatus(p,max(d-2,1))==0
        do=1;
        huannumofp(p)=huannumofp(p)+1;
        scofp{p}(huannumofp(p),:)=zeros(1,3);
        scofp{p}(huannumofp(p),1)=d;
        scofp{p}(huannumofp(p),2)=d;
        scofp{p}(huannumofp(p),3)=1;
    elseif pstatus(p,d)==0 && pstatus(p,d-1)==0&&huannumofp(p)~=0&&scofp{p}(huannumofp(p),3)==0
```

```
602         if d~=2
603             scofp{p}(huannumofp(p),2)=d-2;
604             scofp{p}(huannumofp(p),3)=do;
605             do=0;
606         end
607     elseif pstatus(p,d)==1&&pstatus(p,d-1)==0&&pstatus(p,max(d-2,1))==1
608         do=do+2;
609         scofp{p}(huannumofp(p),2)=d;
610         scofp{p}(huannumofp(p),3)=do;
611     elseif pstatus(p,d)==1
612         do=do+1;
613         scofp{p}(huannumofp(p),2)=d;
614         scofp{p}(huannumofp(p),3)=do;
615     elseif pstatus(p,d)==0
616         do=do;
617         scofp{p}(huannumofp(p),2)=d-1;
618         scofp{p}(huannumofp(p),3)=do;
619     else
620         disp("error9");
621     end
622 end
623
624 startday=zeros(NumOfD,1);
625 endday=zeros(NumOfD,1);
626 huantime=0;
627 for p=1:P
628
    huantime=huantime+(sum(scofp{p}(:,3))-length(scofp{p}(:,3)))*12*60;
629     for i=1:length(scofp{p}(:,3))
630         startday(scofp{p}(i,1))=startday(scofp{p}(i,1))+1;
631         endday(scofp{p}(i,2))=endday(scofp{p}(i,2))+1;
632     end
633 end
634
635 %找结束时间的最早值减去开始时间的最晚值
636 daystarttime=cell(NumOfD,1);
637 dayendtime=cell(NumOfD,1);
638 daydutytime=cell(NumOfD,1);
```

```
for d=1:NumOfD
    l1=0;
    l2=0;
    l=0;
    for i=1:NumOfDuty(d)
        l1=l1+1;
        l2=l2+1;
        l=l+1;
        daydutytime{d}(l,1)=Duty{d}{i,2}(2)-Duty{d}{i,2}(1);
        daystarttime{d}(l1,1)=Duty{d}{i,2}(1);
        dayendtime{d}(l2,1)=Duty{d}{i,2}(2);
    end
    for i=1:NumOfDutyPlus(d)
        l=l+1;
        daydutytime{d}(l,1)=DutyPlus{d}(i,6);
        if DutyPlus{d}(i,7)==1
            l1=l1+1;
            daystarttime{d}(l1,1)=DutyPlus{d}(i,4);
        elseif DutyPlus{d}(i,7)==2
            l2=l2+1;
            dayendtime{d}(l2,1)=DutyPlus{d}(i,5);
        end
    end
end
for d=1:NumOfD
    daystarttime{d}=sort(daystarttime{d},'descend');
    dayendtime{d}=sort(dayendtime{d});
    daydutytime{d}=sort(daydutytime{d});
    if startday(d)>0

huantime=huantime-sum(daystarttime{d}([1:startday(d)]));
    end
    if endday(d)>0
        huantime=huantime+sum(dayendtime{d}([1:endday(d)]));
    end
end

huancost=huantime/60*20;

dutycostplusD=sum(pstatus([7:10],:));
```

```
for d=1:NumOfD
dutycost=dutycost+sum(daydutytime{d}([1:dutycostplusD(d)]))/60*40;
end

shuchu=cell();



function setp(p,d)
global pstatus
global Pstatus
global pold
global poldnum
    pstatus(p,d)=1;
    if sum(Pstatus(:,p)==[0;0;2;2])==4
        Pstatus(:,p)=[1;1;0;1];
    %工作中
    elseif sum(Pstatus([3,4],p)==[0;1])==2&&Pstatus(1,p)~=3&&Pstatus(2,p)==9
        Pstatus(:,p)=[Pstatus(1,p)+1;Pstatus(2,p)+1;0;0];
    elseif sum(Pstatus([3,4],p)==[0;1])==2&&Pstatus(1,p)~=3
        Pstatus(:,p)=[Pstatus(1,p)+1;Pstatus(2,p)+1;0;1];
    elseif sum(Pstatus([3,4],p)==[0;1])==2&&Pstatus(1,p)==3
        Pstatus(:,p)=[Pstatus(1,p)+1;Pstatus(2,p)+1;0;0];
    %休息一天
    elseif sum(Pstatus(:,p)==[0;9;1;1])==4
        Pstatus(:,p)=[1;10;0;0];
    elseif sum(Pstatus([3,4],p)==[1;1])==2
        Pstatus(:,p)=[1;Pstatus(2,p)+1;0;1];

    else
        disp("error3")
    end

    for i=1:3
        pold(i,:)=(Pstatus(4,:)==1).*(Pstatus(1,:)==i);
        poldnum(i)=sum(pold(i,:));
```

```
714     end
715     pold(4,:)=Pstatus(4,:)==2;
716     poldnum(4)=sum(pold(4,:));
717     pold(5,:)=(Pstatus(4,:)==1).*(Pstatus(1,:)==0);
718     poldnum(5)=sum(pold(5,:));
719 end
720
721
722 function restp(p,d)
723 global pstatus
724 global Pstatus
725 global pold
726 global poldnum
727     pstatus(p,d)=0;
728     if sum(Pstatus(:,p)== [0;0;2;2])==4
729         Pstatus(:,p)=[0;0;2;2];
730
731     elseif Pstatus(3,p)==1
732         Pstatus(:,p)=[0;0;2;2];
733     elseif Pstatus(2,p)==10
734         Pstatus(:,p)=[0;0;1;0];
735
736     elseif Pstatus(3,p)==0&&Pstatus(2,p)==9
737         Pstatus(:,p)=[0;0;1;0];
738     elseif Pstatus(3,p)==0
739         Pstatus(:,p)=[0;Pstatus(2,p)+1;1;1];
740     else
741         print("error4")
742     end
743
744     for i=1:3
745         pold(i,:)=(Pstatus(4,:)==1).*(Pstatus(1,:)==i);
746         poldnum(i)=sum(pold(i,:));
747     end
748     pold(4,:)=Pstatus(4,:)==2;
749     poldnum(4)=sum(pold(4,:));
750     pold(5,:)=(Pstatus(4,:)==1).*(Pstatus(1,:)==0);
751     poldnum(5)=sum(pold(5,:));
752
753 end
```

Python 代码

```
# DATA_A
##############################################
NUM_D = 15
NUM_Fli = 206
NUM_P = 7
NUM_BOTH = 6  # 全都能干
NUM_CAP = 5  # 只能当几机长
NUM_FIRST = 10  # 只能当助手
LowerLimit_Combine = 40
UpperLimit_Combine = 1e5
JIDI = [1]
MaxBlk=600
MinRest =660
MaxDP =720
##############################################

# DATA_B
##############################################
# NUM_D = 31
# NUM_Fli = 13954
# NUM_P = 39
# NUM_BOTH = 124  # 全都能干
# NUM_CAP = 87  # 只能当几机长
# NUM_FIRST = 254  # 只能当助手
# LowerLimit_Combine = 40
# UpperLimit_Combine = 1e5
# JIDI = [1,2]
# MaxBlk=600
# MinRest =660
# MaxDP =720
##############################################


from Parameters import *


class Flight:
    def __init__(self, id_in, t_d_in, t_a_in, p_d_in, p_a_in, d_d_in):
```

```
792             self.id = id_in
793             self.cap = 1
794             self.fir = 1
795             self.t_d = t_d_in
796             self.t_a = t_a_in
797             self.p_d = p_d_in
798             self.p_a = p_a_in
799             self.d_d = d_d_in
800
801     def __lt__(self, other):
802         return self.t_d < other.t_d
803
804     def __eq__(self, other):
805         return self.p_d == other.p_d and self.p_a == self.p_a
806
807
808 class Group_Flight:
809     def __init__(self, f: Flight):
810         self.flights = [f]
811         self.t_d = f.t_d
812         self.t_a = f.t_a
813         self.p_d = f.p_d
814         self.p_a = f.p_a
815         self.cap = 1
816         self.first = 1
817         self.d_d = f.d_d
818         self.type = -1    # 1 主机长  3  副机长  2  乘机
819         self.time_fly = f.t_a - f.t_d
820
821     def add_flight(self, f: Flight):
822         self.flights.append(f)
823         self.t_a = f.t_a
824         self.p_a = f.p_a
825         self.time_fly = self.time_fly + f.t_a - f.t_d
826
827     def add_group(self, g):
828         for f in g.flights:
829             self.add_flight(f)
830
831     def add_group_first(self, g):
```

```
832                 for f in reversed(g.flights):
833                     self.add_flight_first(f)
834
835         def add_flight_first(self, f: Flight):
836             self.flights.insert(0,f)
837             self.t_d = f.t_d
838             self.p_d = f.p_d
839             self.time_fly = self.time_fly + f.t_a - f.t_d
840
841         def set_type(self, t):
842             self.type = t
843
844         def __lt__(self, other):
845             return self.t_d < other.t_d
846
847         def __eq__(self, other):
848             return self.p_d == other.p_d and self.p_a == self.p_a
849
850
851 class Pilot:
852         def __init__(self, EmpNo, DutyCostPerHour, ParingCostPerHour, TYPE, jidi):
853             self.EmpNo = EmpNo
854             self.DutyCostPerHour = DutyCostPerHour
855             self.ParingCostPerHour = ParingCostPerHour
856             self.TYPE = TYPE
857             self.jidi = jidi
858             self.flights = []
859             self.group = None
860             self.groups = {}
861             self.duty = [False] * (NUM_D + 1)
862             self.p_a_d = [jidi] * (NUM_D + 1)
863             self.p_d_d = [jidi] * (NUM_D + 1)
864             self.t_a_d = [-1] * (NUM_D + 1)
865             self.t_d_d = [-1] * (NUM_D + 1)
866             self.zhiqinChengben = [0] * (NUM_D + 1)
867             self.time_fly = [0] * (NUM_D + 1)
868             self.time_duty = [0] * (NUM_D + 1)
869
870             for d in range(1, NUM_D + 1):
871                 self.groups[d] = []
```

```
872
873         def assign_group(self, g: Group_Flight):
874             self.group = g
875             self.groups[g.d_d].append(g)
876             self.duty[g.d_d] = True
877             self.p_a_d[g.d_d] = g.p_a
878             self.t_a_d[g.d_d] = g.t_a
879             self.p_d_d[g.d_d] = g.p_d
880             self.t_d_d[g.d_d] = g.t_d
881             self.zhiqinChengben[g.d_d] = self.DutyCostPerHour * (self.t_a_d[g.d_d] - self.t_d_d[g.d_d])
882             self.time_duty[g.d_d] = (self.t_a_d[g.d_d] - self.t_d_d[g.d_d])
883             self.time_fly[g.d_d] = g.time_fly
884
885         def add_group(self, g: Group_Flight):
886             if len(self.groups[g.d_d]) == 0:
887                 self.assign_group(g)
888                 return
889             self.groups[g.d_d].append(g)
890             self.p_a_d[g.d_d] = g.p_a
891             self.t_a_d[g.d_d] = g.t_a
892             self.time_duty[g.d_d] = (self.t_a_d[g.d_d] - self.t_d_d[g.d_d])
893             self.zhiqinChengben[g.d_d] = self.DutyCostPerHour * self.time_duty[g.d_d]
894             self.time_fly[g.d_d] = self.time_fly[g.d_d] + g.time_fly
895
896         def add_group_first(self, g: Group_Flight):
897             if len(self.groups[g.d_d]) == 0:
898                 self.assign_group(g)
899                 return
900             self.groups[g.d_d].insert(0, g)
901             self.p_d_d[g.d_d] = g.p_d
902             self.t_d_d[g.d_d] = g.t_d
903             self.time_duty[g.d_d] = (self.t_a_d[g.d_d] - self.t_d_d[g.d_d])
904             self.zhiqinChengben[g.d_d] = self.DutyCostPerHour * self.time_duty[g.d_d]
905             self.time_fly[g.d_d] = self.time_fly[g.d_d] + g.time_fly
906
907 import pandas as pd
908 import numpy as np
909 from Flight import *
910 from Parameters import *
911 import copy
```

```
912 import time
913
914
915 class Greedy:
916     def __init__(self):
917         self.data = None
918         self.map_p2f = {}  # 机场到航班（起飞相同）的映射
919         for i in range(1, NUM_P + 1):
920             self.map_p2f[i] = set()
921         self.map_t2f = {}
922         self.pilots = {}
923         self.pilots[1] = []
924         self.pilots[2] = []
925         self.pilots[3] = []
926         self.chengji = 0
927         self.zhiqinChengben = 0
928         self.read()
929         self.map_t2f[0] = []
930         for i in range(1, NUM_D + 1):
931             self.map_t2f[i] = []
932         for i, r in self.data.iterrows():
933             self.map_t2f[r["DptrD"]].append(
934                 Flight(i, r["DptrT_min"], r["ArrvT_min"], r["DptrP"], r["ArrvP"], r["DptrD"]))
935
936         self.map_t2g = None
937
938     def read(self):
939         self.data = pd.read_csv("Data_A_Flight.csv", dtype={"ArrvD": int, "ArrvT": int,
940                                                             "DptrD": int, "DptrT": int,
"ArrvP": int, "DptrP": int,
941                                                             "FlightT": int, "DptrT_min": int,
"ArrvT_min": int})
942         for i, r in self.data.iterrows():
943             self.map_p2f[r["DptrP"]].add(i)
944
945         self.data_pilot = pd.read_csv("Data_A_Pilot.csv", dtype={"DutyCostPerHour": int,
"ParingCostPerHour": int,
946                                                                  "TYPE": int, "jidi": int,
"EmpNo": str})
947         for i, r in self.data_pilot.iterrows():
```

```
948             self.pilots[r["TYPE"]].append(Pilot(r["EmpNo"], r["DutyCostPerHour"],
r["ParingCostPerHour"],
949                                                 r["TYPE"], r["jidi"]))
950
951         # print(self.data)
952
953     def run(self):
954         ordered_map_p2f = {}
955         for i in range(1, NUM_P + 1):
956             ordered_map_p2f[i] = list()
957         for i, r in self.data.iterrows():
958             self.map_p2f[r["DptrP"]].add(Flight(i, r["DptrT_min"]))
959         for i in range(1, NUM_P + 1):
960             ordered_map_p2f[i].sort()
961         num_allocated_per = 0
962         while num_allocated_per < NUM_Fli:
963             pass
964
965     def make_group(self):
966         map_t2f_temp = copy.deepcopy(self.map_t2f)
967         map_t2g = {}
968         map_t2g[0] = []
969         for d in range(1, NUM_D + 1):
970             map_t2g[d] = []
971             flag_group = True
972             map_t2f_temp[d].sort()
973             while flag_group:
974                 flag_group = False
975                 temp_f1 = None
976                 temp_f2 = None
977                 for f1 in range(len(map_t2f_temp[d])):
978                     if map_t2f_temp[d][f1].p_d not in JIDI:
979                         continue
980                     for f2 in range(f1 + 1, len(map_t2f_temp[d])):
981                         if (map_t2f_temp[d][f1].p_d == map_t2f_temp[d][f2].p_a
982                                 and map_t2f_temp[d][f1].p_a == map_t2f_temp[d][f2].p_d
983                                 and (map_t2f_temp[d][f2].t_d - map_t2f_temp[d][f1].t_a) >
UpperLimit_Combine):
984                             print("检查 2")
985                         if (map_t2f_temp[d][f1].p_d == map_t2f_temp[d][f2].p_a
```

```
                                    and map_t2f_temp[d][f1].p_a == map_t2f_temp[d][f2].p_d
                                    and (map_t2f_temp[d][f2].t_d - map_t2f_temp[d][f1].t_a) >= LowerLimit_Combine
                                    and (map_t2f_temp[d][f2].t_d - map_t2f_temp[d][f1].t_a) <= UpperLimit_Combine):
                                g = Group_Flight(map_t2f_temp[d][f1])
                                g.add_flight(map_t2f_temp[d][f2])
                                map_t2g[d].append(g)
                                if map_t2f_temp[d][f1].p_d not in JIDI:
                                    print("检查 1")
                                flag_group = True
                                temp_f1 = f1
                                temp_f2 = f2
                                break
                        if flag_group:
                            break
                    if flag_group:
                        del map_t2f_temp[d][temp_f2]
                        del map_t2f_temp[d][temp_f1]

        for d in range(1, NUM_D + 1):
            for f1 in range(len(map_t2f_temp[d])):
                map_t2g[d].append(Group_Flight(map_t2f_temp[d][f1]))

        self.map_t2g = map_t2g
        return map_t2g

    def coombine_group(self):
        for d in range(1, NUM_D + 1):
            flag_com = True  # 是否还有可以合并的
            self.map_t2g[d].sort()
            while flag_com:
                flag_com = False
                temp_g1 = None
                temp_g2 = None
                for g1 in range(len(self.map_t2g[d])):
                    for g2 in range(g1 + 1, len(self.map_t2g[d])):
                        if (self.map_t2g[d][g1].p_d == self.map_t2g[d][g2].p_a
                                    and self.map_t2g[d][g1].p_a == self.map_t2g[d][g2].p_d
                                    and (self.map_t2g[d][g2].t_d - self.map_t2g[d][g1].t_a) >
```

```
                UpperLimit_Combine):
1024                                # 由于连接时间过长而没有连接的 group
1025                                print("检查 3")
1026                            if (self.map_t2g[d][g1].p_d == self.map_t2g[d][g2].p_a
1027                                    and self.map_t2g[d][g1].p_a == self.map_t2g[d][g2].p_d
1028                                    and (self.map_t2g[d][g2].t_d - self.map_t2g[d][g1].t_a) >=
                LowerLimit_Combine
1029                                    and (self.map_t2g[d][g2].t_d - self.map_t2g[d][g1].t_a) <=
                UpperLimit_Combine):
1030                                self.map_t2g[d][g1].add_group(self.map_t2g[d][g2])
1031                                temp_g2 = g2
1032                                flag_com = True
1033                                break
1034                        if flag_com:
1035                            break
1036                    if flag_com:
1037                        del self.map_t2g[d][temp_g2]
1038
1039                for g in range(len(self.map_t2g[d])):
1040                    if self.map_t2g[d][g].p_a not in JIDI:
1041                        print("检查 4")
1042
1043        def allocate_pilot_zhiqin(self):
1044            self.chengji = 0
1045            for d in range(1, NUM_D + 1):
1046                for g in self.map_t2g[d]:
1047                    if (g.p_d in JIDI):
1048                        flag_1 = False  # 正机长是否分配成功
1049                        flag_3 = False  # 副机长是否分配成功
1050                        for p in self.pilots[1]:
1051                            if (not p.duty[d]) and (not p.duty[d - 1]):
1052                                p.assign_group(g)
1053                                flag_1 = True
1054                        for p in self.pilots[3]:
1055                            if (not p.duty[d]) and (not p.duty[d - 1]):
1056                                p.assign_group(g)
1057                                flag_3 = True
1058                        if not flag_1:
1059                            for p in self.pilots[2]:
1060                                if (not p.duty[d]) and (not p.duty[d - 1]):
```

```
                                p.assign_group(g)
                                flag_1 = True
                    if not flag_3:
                        for p in self.pilots[2]:
                            if (not p.duty[d]) and (not p.duty[d - 1]):
                                p.assign_group(g)
                                flag_3 = True
                    if not flag_1 or not flag_3:
                        print("分配失败")
                else:
                    for g2 in self.map_t2g[d]:
                        if (g2.p_d in JIDI and g.t_d - g2.t_a >= LowerLimit_Combine):
                            flag_1 = False  # 正机长是否分配成功
                            flag_3 = False  # 副机长是否分配成功
                            for p in self.pilots[1]:
                                if (not p.duty[d]) and (not p.duty[d - 1]):
                                    p.assign_group(g2)
                                    p.add_group(g)
                                    flag_1 = True
                                    self.chengji = self.chengji + 1
                            for p in self.pilots[3]:
                                if (not p.duty[d]) and (not p.duty[d - 1]):
                                    p.assign_group(g2)
                                    p.add_group(g)
                                    flag_3 = True
                                    self.chengji = self.chengji + 1

                            if not flag_1:
                                for p in self.pilots[2]:
                                    if (not p.duty[d]) and (not p.duty[d - 1]):
                                        p.assign_group(g2)
                                        p.add_group(g)
                                        flag_1 = True
                                        self.chengji = self.chengji + 1
                            if not flag_3:
                                for p in self.pilots[2]:
                                    if (not p.duty[d]) and (not p.duty[d - 1]):
                                        p.assign_group(g2)
                                        p.add_group(g)
                                        flag_3 = True
```

```
                                    self.chengji = self.chengji + 1
                            if not flag_1 or not flag_3:
                                print("分配失败")
                        if flag_1 and flag_3:
                            break

    def allocate_pilot(self):
        self.chengji = 0
        for d in range(1, NUM_D + 1):
            for g in self.map_t2g[d]:
                if (g.p_d in JIDI and g.p_a in JIDI):
                    flag_1 = False  # 正机长是否分配成功
                    flag_3 = False  # 副机长是否分配成功
                    for p in self.pilots[1]:
                        if not p.duty[d] and p.p_a_d[d - 1] == g.p_d:
                            p.assign_group(g)
                            p.groups[d][-1].set_type(1)
                            flag_1 = True
                            break
                    for p in self.pilots[3]:
                        if not p.duty[d] and p.p_a_d[d - 1] == g.p_d:
                            p.assign_group(g)
                            p.groups[d][-1].set_type(3)
                            flag_3 = True
                            break
                    if not flag_1:
                        for p in self.pilots[2]:
                            if not p.duty[d] and p.p_a_d[d - 1] == g.p_d:
                                p.assign_group(g)
                                p.groups[d][-1].set_type(1)
                                flag_1 = True
                                break
                    if not flag_3:
                        for p in self.pilots[2]:
                            if not p.duty[d] and p.p_a_d[d - 1] == g.p_d:
                                p.assign_group(g)
                                p.groups[d][-1].set_type(3)
                                flag_3 = True
                                break
                    if (not flag_1) or (not flag_3):
```

```
                        print(f"{d}天分配失败 3")
                    else:
                        print("分配成功 1")
                        continue
                elif g.p_d not in JIDI:
                    flag_1_wai = False
                    flag_3_wai = False
                    for p in self.pilots[1]:
                        if p.p_a_d[d - 1] == g.p_d and (not p.duty[d]):
                            p.assign_group(g)
                            p.groups[d][-1].set_type(1)
                            flag_1_wai = True
                            break
                    for p in self.pilots[3]:
                        if p.p_a_d[d - 1] == g.p_d and (not p.duty[d]):
                            p.assign_group(g)
                            p.groups[d][-1].set_type(3)
                            flag_1_wai = True
                            break
                    if not flag_1_wai:
                        for p in self.pilots[2]:
                            if p.p_a_d[d - 1] == g.p_d and (not p.duty[d]):
                                p.assign_group(g)
                                p.groups[d][-1].set_type(1)
                                flag_1_wai = True
                                break
                    if not flag_3_wai:
                        for p in self.pilots[2]:
                            if p.p_a_d[d - 1] == g.p_d and (not p.duty[d]):
                                p.assign_group(g)
                                p.groups[d][-1].set_type(3)
                                flag_1_wai = True
                                break
                    if flag_1_wai and flag_3_wai:
                        continue

                    for g2 in self.map_t2g[d]:
                        flag_1 = flag_1_wai  # 正机长是否分配成功
                        flag_3 = flag_3_wai  # 副机长是否分配成功
                        if (g2.p_d in JIDI and g.t_d - g2.t_a >= LowerLimit_Combine and g2.p_a
```

```
                == g.p_d):
1181                    if not flag_1:
1182                        for p in self.pilots[1]:
1183                            if (not p.duty[d]) and p.p_a_d[d - 1] == g2.p_d:
1184                                p.assign_group(g2)
1185                                p.groups[d][-1].set_type(2)
1186                                p.add_group(g)
1187                                p.groups[d][-1].set_type(1)
1188                                flag_1 = True
1189                                self.chengji = self.chengji + 1
1190                                break
1191                    if not flag_3:
1192                        for p in self.pilots[3]:
1193                            if (not p.duty[d]) and p.p_a_d[d - 1] == g2.p_d:
1194                                p.assign_group(g2)
1195                                p.groups[d][-1].set_type(2)
1196                                p.add_group(g)
1197                                p.groups[d][-1].set_type(3)
1198                                flag_3 = True
1199                                self.chengji = self.chengji + 1
1200                                break
1201
1202                    if not flag_1:
1203                        for p in self.pilots[2]:
1204                            if (not p.duty[d]) and p.p_a_d[d - 1] == g2.p_d:
1205                                p.assign_group(g2)
1206                                p.groups[d][-1].set_type(2)
1207                                p.add_group(g)
1208                                p.groups[d][-1].set_type(1)
1209                                flag_1 = True
1210                                self.chengji = self.chengji + 1
1211                                break
1212                    if not flag_3:
1213                        for p in self.pilots[2]:
1214                            if (not p.duty[d]) and p.p_a_d[d - 1] == g2.p_d:
1215                                p.assign_group(g2)
1216                                p.groups[d][-1].set_type(2)
1217                                p.add_group(g)
1218                                p.groups[d][-1].set_type(3)
1219                                flag_3 = True
```

```
                                    self.chengji = self.chengji + 1
                                    break
                            if (not flag_1) or (not flag_3):
                                print("分配失败 2")
                            else:
                                print("分配成功 2")
                        if flag_1 and flag_3:
                            flag_1_wai = True
                            flag_3_wai = True
                            break

                if not (flag_1_wai and flag_3_wai):
                    for g2 in self.map_t2g[d - 1]:
                        flag_1 = flag_1_wai  # 正机长是否分配成功
                        flag_3 = flag_3_wai  # 副机长是否分配成功
                        if (g2.p_d in JIDI and g.t_d - g2.t_a >= LowerLimit_Combine):
                            if not flag_1:
                                for p in self.pilots[1]:
                                    if (not p.duty[d]) and (not p.duty[d - 1]) and p.p_a_d[d - 2] == g2.p_d:
                                        p.add_group(g2)
                                        p.groups[d - 1][-1].set_type(2)
                                        p.assign_group(g)
                                        p.groups[d][-1].set_type(1)
                                        flag_1 = True
                                        self.chengji = self.chengji + 1
                                        break
                            if not flag_3:
                                for p in self.pilots[3]:
                                    if (not p.duty[d] and (not p.duty[d - 1])) and p.p_a_d[d - 2] == g2.p_d:
                                        p.add_group(g2)
                                        p.groups[d - 1][-1].set_type(2)
                                        p.assign_group(g)
                                        p.groups[d][-1].set_type(3)
                                        flag_3 = True
                                        self.chengji = self.chengji + 1
                                        break

                        if not flag_1:
```

```
                                    for p in self.pilots[2]:
                                        if (not p.duty[d]) and (not p.duty[d - 1]) and p.p_a_d[d - 1] == g2.p_d:
                                            p.add_group(g2)
                                            p.groups[d - 1][-1].set_type(2)
                                            p.assign_group(g)
                                            p.groups[d][-1].set_type(1)
                                            flag_1 = True
                                            self.chengji = self.chengji + 1
                                            break
                                if not flag_3:
                                    for p in self.pilots[2]:
                                        if (not p.duty[d] and (not p.duty[d - 1])) and p.p_a_d[d - 1] == g2.p_d:
                                            p.add_group(g2)
                                            p.groups[d - 1][-1].set_type(2)
                                            p.assign_group(g)
                                            p.groups[d][-1].set_type(3)
                                            flag_3 = True
                                            self.chengji = self.chengji + 1
                                            break
                                if not flag_1 or not flag_3:
                                    print("分配失败 1")
                                else:
                                    print("分配成功 3")
                            if flag_1 and flag_3:
                                flag_1_wai = True
                                flag_3_wai = True
                                break
                        if not (flag_1_wai and flag_3_wai):  # 前面没有单独航班配对，只能找航班来配对
                            for g2 in self.map_t2g[d]:
                                self.dangtian_find_fly2_g(g, g2, flag_1_wai, flag_3_wai, d)

    def dangtian_find_fly2_g(self, g, g2, flag_1_wai, flag_3_wai, d):
        flag_1 = flag_1_wai  # 正机长是否分配成功
        flag_3 = flag_3_wai  # 副机长是否分配成功
        if (g2.p_d in JIDI and g.t_d - g2.t_a >= LowerLimit_Combine and g2.p_a == g.p_d):
            if not flag_1:
                for p in self.pilots[1]:
```

```
                        if (not p.duty[d]) and p.p_a_d[d - 1] == g2.p_d:
                            p.assign_group(g2)
                            p.groups[d][-1].set_type(2)
                            p.add_group(g)
                            p.groups[d][-1].set_type(1)
                            flag_1 = True
                            self.chengji = self.chengji + 1
                            break
                if not flag_3:
                    for p in self.pilots[3]:
                        if (not p.duty[d]) and p.p_a_d[d - 1] == g2.p_d:
                            p.assign_group(g2)
                            p.groups[d][-1].set_type(2)
                            p.add_group(g)
                            p.groups[d][-1].set_type(3)
                            flag_3 = True
                            self.chengji = self.chengji + 1
                            break

                if not flag_1:
                    for p in self.pilots[2]:
                        if (not p.duty[d]) and p.p_a_d[d - 1] == g2.p_d:
                            p.assign_group(g2)
                            p.groups[d][-1].set_type(2)
                            p.add_group(g)
                            p.groups[d][-1].set_type(1)
                            flag_1 = True
                            self.chengji = self.chengji + 1
                            break
                if not flag_3:
                    for p in self.pilots[2]:
                        if (not p.duty[d]) and p.p_a_d[d - 1] == g2.p_d:
                            p.assign_group(g2)
                            p.groups[d][-1].set_type(2)
                            p.add_group(g)
                            p.groups[d][-1].set_type(3)
                            flag_3 = True
                            self.chengji = self.chengji + 1
                            break
                if (not flag_1) or (not flag_3):
```

```
                print("分配失败 2")
            else:
                print("分配成功 2")

    def print_result_pl(self):
        print("开始统计")
        result = pd.DataFrame(columns=('EmpNo', 'Day', 'FlightNo', 'DptrT', 'DptrP', 'ArrvT', 'ArrvP', 'Type'))
        self.zhiqinChengben = []
        self.time_fly = []
        self.time_duty = []
        self.num_tibu = [0] * NUM_BOTH
        idx = -1
        for p in self.pilots[2]:
            idx = idx + 1
            for d in range(1, NUM_D + 1):
                if p.duty[d]:
                    for g in p.groups[d]:
                        if g.type == 3:
                            self.num_tibu[idx] = self.num_tibu[idx] + 1
        print(f"替补数量: {sum(self.num_tibu)}")
        # hangshu=0
        # for p in self.pilots[1] + self.pilots[2] + self.pilots[3]:
        #     self.time_fly.append(sum(p.time_fly))
        #     self.zhiqinChengben.append(sum(p.zhiqinChengben))
        #     self.time_duty.append(sum(p.time_duty))
        #     for d in range(1, NUM_D + 1):
        #         print(d)
        #         if p.duty[d]:
        #             for g in p.groups[d]:
        #                 for f in g.flights:
        #                     hangshu=hangshu+1
        #         else:
        #             hangshu = hangshu + 1
        # print(hangshu)
        hangshu = 0
        table = []
        for p in self.pilots[1] + self.pilots[2] + self.pilots[3]:
            self.time_fly.append(sum(p.time_fly))
            self.zhiqinChengben.append(sum(p.zhiqinChengben)/60.0)
```

```
            self.time_duty.append(sum(p.time_duty))

            for d in range(1, NUM_D + 1):
                if p.duty[d]:
                    for g in p.groups[d]:
                        for f in g.flights:
                            # table.append([p.EmpNo,  d,  f.id,  f.t_d,f.p_d,  f.t_a,  f.p_a, g.type])
                            # print([p.EmpNo,  d,  f.id,  f.t_d,f.p_d,  f.t_a,  f.p_a, g.type], hangshu)
                            hangshu = hangshu + 1
                            print(hangshu)
                            result = result.append({'EmpNo': p.EmpNo, 'Day': d, 'FlightNo': f.id, 'DptrT': f.t_d,
                                                    'DptrP': f.p_d, 'ArrvT': f.t_a, 'ArrvP': f.p_a, 'Type': g.type},
                                                   ignore_index=True)
                else:
                    hangshu = hangshu + 1
                    print(hangshu)
                    result = result.append({'EmpNo': p.EmpNo, 'Day': d, 'Type': 0}, ignore_index=True)
        # result=pd.concat([result, pd.DataFrame(table, columns=['EmpNo', 'Day', 'FlightNo', 'DptrT',
        #                                                        'DptrP', 'ArrvT', 'ArrvP', 'Type'])])

        print(f"执勤成本总和: {sum(self.zhiqinChengben)}")
        print(f"执勤成本最小: {min(self.zhiqinChengben) }")
        print(f"执勤成本最大: {max(self.zhiqinChengben) }")
        print(f"飞行时间总和: {sum(self.time_fly) / 60}")
        print(f"飞行时间最小: {min(self.time_fly) / 60}")
        print(f"飞行时间最大: {max(self.time_fly) / 60}")
        print(f"执勤时间总和: {sum(self.time_duty) / 60}")
        print(f"执勤时间最小: {min(self.time_duty) / 60}")
        print(f"执勤时间最大: {max(self.time_duty) / 60}")

        result.to_csv("CrewRosters_A_p1.csv", index=False)

    def print_result_p2(self):
        print("开始统计")
```

```
        result = pd.DataFrame(columns=('EmpNo', 'Day', 'FlightNo', 'DptrT', 'DptrP', 'ArrvT', 'ArrvP', 'Type'))
        self.zhiqinChengben = []
        self.time_fly = []
        self.time_duty = []
        self.zhiqintianshu=[]
        self.ziqinshichang_perduty=[]
        self.time_fly_perduty=[]
        self.num_tibu = [0] * NUM_BOTH
        idx = -1
        for p in self.pilots[2]:
            idx = idx + 1
            for d in range(1, NUM_D + 1):
                if p.duty[d]:
                    for g in p.groups[d]:
                        if g.type == 3:
                            self.num_tibu[idx] = self.num_tibu[idx] + 1
        print(f"替补数量: {sum(self.num_tibu)}")
        hangshu = 0
        for p in self.pilots[1] + self.pilots[2] + self.pilots[3]:
            self.zhiqintianshu.append(sum(p.duty))
            self.time_fly.append(sum(p.time_fly))
            self.zhiqinChengben.append(sum(p.zhiqinChengben) / 60.0)
            self.time_duty.append(sum(p.time_duty))
            for d in range(1, NUM_D + 1):
                self.ziqinshichang_perduty.append(p.t_a_d[d]-p.t_d_d[d])
                if (p.t_a_d[d]-p.t_d_d[d])/60>24:
                    print("66666666666666666666")
                self.time_fly_perduty.append(p.time_fly[d])
                if p.duty[d]:
                    for g in p.groups[d]:
                        for f in g.flights:
                            hangshu = hangshu + 1
                            print(hangshu)
                            # result = result.append({'EmpNo': p.EmpNo, 'Day': d, 'FlightNo': f.id, 'DptrT': f.t_d,
                            #                         'DptrP': f.p_d, 'ArrvT': f.t_a, 'ArrvP': f.p_a, 'Type': g.type},
                            #                        ignore_index=True)
                else:
```

```
                        hangshu = hangshu + 1
                        print(hangshu)
                        # result = result.append({'EmpNo': p.EmpNo, 'Day': d, 'Type': 0},
ignore_index=True)
        print(f"执勤成本总和: {sum(self.zhiqinChengben)}")
        print(f"执勤成本最小: {min(self.zhiqinChengben) }")
        print(f"执勤成本最大: {max(self.zhiqinChengben) }")
        print(f"飞行时间总和: {sum(self.time_fly) / 60}")
        print(f"飞行时间最小: {min(self.time_fly) / 60}")
        print(f"飞行时间最大: {max(self.time_fly) / 60}")
        print(f"执勤时间总和: {sum(self.time_duty) / 60}")
        print(f"执勤时间最小: {min(self.time_duty) / 60}")
        print(f"执勤时间最大: {max(self.time_duty) / 60}")
        print(f"执勤天数平均: {np.mean(self.zhiqintianshu)}")
        print(f"执勤天数最小: {min(self.zhiqintianshu)}")
        print(f"执勤天数最大: {max(self.zhiqintianshu)}")
        print(f"一次执勤时长平均: {np.mean(self.ziqinshichang_perduty)/60}")
        print(f"一次执勤时长最小: {min(self.ziqinshichang_perduty)/60}")
        print(f"一次执勤时长最大: {max(self.ziqinshichang_perduty)/60}")
        print(f"一次执勤飞行时长平均: {np.mean(self.time_fly_perduty)/60}")
        print(f"一次执勤飞行时长最小: {min(self.time_fly_perduty)/60}")
        print(f"一次执勤飞行时长最大: {max(self.time_fly_perduty)/60}")
        self.total_zhiqin=sum(self.time_duty)
        result.to_csv("CrewRosters_A_p2.csv", index=False)

    def print_result_p3(self):
        print("开始统计")
        result = pd.DataFrame(columns=('EmpNo', 'Day', 'FlightNo', 'DptrT', 'DptrP', 'ArrvT',
'ArrvP', 'Type'))
        self.zhiqinChengben = []
        self.time_fly = []
        self.time_duty = []
        self.zhiqintianshu=[]
        self.ziqinshichang_perduty=[]
        self.time_fly_perduty=[]
        self.num_tibu = [0] * NUM_BOTH
        idx = -1
        for p in self.pilots[2]:
            idx = idx + 1
            for d in range(1, NUM_D + 1):
```

```
                if p.duty[d]:
                    for g in p.groups[d]:
                        if g.type == 3:
                            self.num_tibu[idx] = self.num_tibu[idx] + 1
        print(f"替补数量: {sum(self.num_tibu)}")
        hangshu = 0
        for p in self.pilots[1] + self.pilots[2] + self.pilots[3]:
            self.zhiqintianshu.append(sum(p.duty))
            self.time_fly.append(sum(p.time_fly))
            self.zhiqinChengben.append(sum(p.zhiqinChengben) / 60.0)
            self.time_duty.append(sum(p.time_duty))
            for d in range(1, NUM_D + 1):
                self.ziqinshichang_perduty.append(p.t_a_d[d]-p.t_d_d[d])
                if (p.t_a_d[d]-p.t_d_d[d])/60>24:
                    print("666666666666666666666")
                self.time_fly_perduty.append(p.time_fly[d])
                if p.duty[d]:
                    for g in p.groups[d]:
                        for f in g.flights:
                            hangshu = hangshu + 1
                            print(hangshu)
                            # result = result.append({'EmpNo': p.EmpNo, 'Day': d, 'FlightNo': f.id, 'DptrT': f.t_d,
                            #                         'DptrP': f.p_d, 'ArrvT': f.t_a, 'ArrvP': f.p_a, 'Type': g.type},
                            #                        ignore_index=True)
                else:
                    hangshu = hangshu + 1
                    print(hangshu)
                    # result = result.append({'EmpNo': p.EmpNo, 'Day': d, 'Type': 0},
ignore_index=True)
        print(f"执勤成本总和: {sum(self.zhiqinChengben)}")
        print(f"执勤成本最小: {min(self.zhiqinChengben) }")
        print(f"执勤成本最大: {max(self.zhiqinChengben) }")
        print(f"飞行时间总和: {sum(self.time_fly) / 60}")
        print(f"飞行时间最小: {min(self.time_fly) / 60}")
        print(f"飞行时间最大: {max(self.time_fly) / 60}")
        print(f"执勤时间总和: {sum(self.time_duty) / 60}")
        print(f"执勤时间最小: {min(self.time_duty) / 60}")
        print(f"执勤时间最大: {max(self.time_duty) / 60}")
```

```
        print(f"执勤天数求和: {np.sum(self.zhiqintianshu)}")
        print(f"执勤天数平均: {np.mean(self.zhiqintianshu)}")
        print(f"执勤天数最小: {min(self.zhiqintianshu)}")
        print(f"执勤天数最大: {max(self.zhiqintianshu)}")
        print(f"一次执勤时长平均: {np.mean(self.ziqinshichang_perduty)/60}")
        print(f"一次执勤时长最小: {min(self.ziqinshichang_perduty)/60}")
        print(f"一次执勤时长最大: {max(self.ziqinshichang_perduty)/60}")
        print(f"一次执勤飞行时长平均: {np.mean(self.time_fly_perduty)/60}")
        print(f"一次执勤飞行时长最小: {min(self.time_fly_perduty)/60}")
        print(f"一次执勤飞行时长最大: {max(self.time_fly_perduty)/60}")
        self.total_zhiqin=sum(self.time_duty)
        result.to_csv("CrewRosters_B_p3.csv", index=False)

    def judge_rep(self, g: Group_Flight):
        if (g.flights[0].p_d == 1 and g.flights[0].p_a == 3) or \
                (g.flights[0].p_d == 1 and g.flights[0].p_a == 6) or \
                (g.flights[0].p_d == 1 and g.flights[0].p_a == 5):
            return True
        else:
            return False

    def judge_rep_1(self, g: Group_Flight):
        if (g.flights[0].p_d == 1 and g.flights[0].p_a == 3) or \
                (g.flights[0].p_d == 1 and g.flights[0].p_a == 6):
            return True
        else:
            return False

    def judge_rep_2(self, g: Group_Flight):
        if (g.flights[0].p_d == 1 and g.flights[0].p_a == 5):
            return True
        else:
            return False

    def allocate_p2(self):
        ##分配第 0 天##
        for g in self.map_t2g[1]:
            if self.judge_rep(g):
                flag_1, flag_3 = self.assign_p_to_group(g, 1)
                if (not flag_1) or (not flag_3):
```

```
                    print(f"{1}天分配失败 3")
                else:
                    print("分配成功 1")
                flag_1, flag_3 = self.assign_p_to_group_chengji(g, 1)
                if (not flag_1) or (not flag_3):
                    print(f"{1}天分配失败 3")
                else:
                    print("分配成功 1")
                    continue
            elif len(g.flights) == 1:
                flag_1, flag_3 = self.assign_p_to_group_dan(g, g.d_d)
                if (not flag_1) or (not flag_3):
                    print(f"{1}天分配失败 3")
                else:
                    print("分配成功 1")
                    continue

            else:
                flag_1, flag_3 = self.assign_p_to_group(g, 1)
                if (not flag_1) or (not flag_3):
                    print(f"{1}天分配失败 3")
                else:
                    print("分配成功 1")
                    continue

        ##分配第二天##
        for g in self.map_t2g[2]:
            if self.judge_rep_2(g):
                flag_1, flag_3 = self.assign_p_to_group(g, 2)
                if (not flag_1) or (not flag_3):
                    print(f"{2}天分配失败 3")
                else:
                    print("分配成功 1")
                flag_1, flag_3 = self.assign_p_to_group_chengji(g, 2)
                if (not flag_1) or (not flag_3):
                    print(f"{2}天分配失败 3")
                else:
                    print("分配成功 1")
                    continue
            elif self.judge_rep_1(g):
```

```
                self.assign_p_to_group_zaiFeiShuangXiang(g, g.d_d)
            elif len(g.flights) == 1:
                if [g.flights[0].id == 1]:
                    print(1)
                flag_1, flag_3 = self.assign_p_to_group_dan(g, g.d_d)
                if (not flag_1) or (not flag_3):
                    print(f"{1}天分配失败 3")
                else:
                    print("分配成功 1")
                    continue
            else:
                flag_1, flag_3 = self.assign_p_to_group(g, 2)
                if (not flag_1) or (not flag_3):
                    print(f"{1}天分配失败 3")
                else:
                    print("分配成功 1")
                    continue
        ##分配第三到十四天##
        for d in range(3, NUM_D):
            for g in self.map_t2g[d]:
                if self.judge_rep(g):
                    self.assign_p_to_group_zaiFeiShuangXiang(g, g.d_d)
                elif len(g.flights) == 1:
                    flag_1, flag_3 = self.assign_p_to_group_dan(g, g.d_d)
                    if (not flag_1) or (not flag_3):
                        print(f"{1}天分配失败 3")
                    else:
                        print("分配成功 1")
                        continue
                else:
                    flag_1, flag_3 = self.assign_p_to_group(g, g.d_d)
                    if (not flag_1) or (not flag_3):
                        print(f"{1}天分配失败 3")
                    else:
                        print("分配成功 1")
                        continue
        ##分配最后一天##
        for g in self.map_t2g[15]:
            if self.judge_rep(g):
                flag_1, flag_3 = self.assign_p_to_group(g, 15)
```

```
                if (not flag_1) or (not flag_3):
                    print(f"{2}天分配失败 3")
                else:
                    print("分配成功 1")
                self.assign_p_to_group_zaiFeiShuangXiang_qian(g, 15)
            elif len(g.flights) == 1:
                flag_1, flag_3 = self.assign_p_to_group_dan(g, g.d_d)
                if (not flag_1) or (not flag_3):
                    print(f"{1}天分配失败 3")
                else:
                    print("分配成功 1")
                    continue
            else:
                flag_1, flag_3 = self.assign_p_to_group(g, g.d_d)
                if (not flag_1) or (not flag_3):
                    print(f"{1}天分配失败 3")
                else:
                    print("分配成功 1")
                    continue

    def assign_p_to_group(self, g, d):
        flag_1 = False  # 正机长是否分配成功
        flag_3 = False  # 副机长是否分配成功
        for p in self.pilots[1]:
            if not p.duty[d]:
                p.assign_group(g)
                p.groups[d][-1].set_type(1)
                flag_1 = True
                break
        for p in self.pilots[3]:
            if not p.duty[d]:
                p.assign_group(g)
                p.groups[d][-1].set_type(3)
                flag_3 = True
                break
        if not flag_1:
            for p in self.pilots[2]:
                if not p.duty[d]:
                    p.assign_group(g)
                    p.groups[d][-1].set_type(1)
```

```
                    flag_1 = True
                    break
        if not flag_3:
            for p in self.pilots[2]:
                if not p.duty[d]:
                    p.assign_group(g)
                    p.groups[d][-1].set_type(3)
                    flag_3 = True
                    break
        return flag_1, flag_3

    def assign_p_to_group_chengji(self, g, d):
        flag_1 = False  # 正机长是否分配成功
        flag_3 = False  # 副机长是否分配成功
        for p in self.pilots[1]:
            if not p.duty[d]:
                p.assign_group(Group_Flight(g.flights[0]))
                p.groups[d][-1].set_type(2)
                flag_1 = True
                break
        for p in self.pilots[3]:
            if not p.duty[d]:
                p.assign_group(Group_Flight(g.flights[0]))
                p.groups[d][-1].set_type(2)
                flag_3 = True
                break
        if not flag_1:
            for p in self.pilots[2]:
                if not p.duty[d]:
                    p.assign_group(Group_Flight(g.flights[0]))
                    p.groups[d][-1].set_type(2)
                    flag_1 = True
                    break
        if not flag_3:
            for p in self.pilots[2]:
                if not p.duty[d]:
                    p.assign_group(Group_Flight(g.flights[0]))
                    p.groups[d][-1].set_type(2)
                    flag_3 = True
                    break
```

```
        return flag_1, flag_3

    def assign_p_to_group_chengji_1(self, g, d, g_zhen):
        flag_1 = False  # 正机长是否分配成功
        for p in self.pilots[1]:
            if not p.duty[d]:
                p.assign_group(Group_Flight(g.flights[0]))
                p.groups[d][-1].set_type(2)
                flag_1 = True
                p.add_group(g_zhen)
                p.groups[g_zhen.d_d][-1].set_type(1)
                break

        if not flag_1:
            for p in self.pilots[2]:
                if not p.duty[d]:
                    p.assign_group(Group_Flight(g.flights[0]))
                    p.groups[d][-1].set_type(2)
                    flag_1 = True
                    p.add_group(g_zhen)
                    p.groups[g_zhen.d_d][-1].set_type(1)
                    break
        return flag_1

    def assign_p_to_group_chengji_3(self, g, d, g_zhen):

        flag_3 = False  # 副机长是否分配成功

        for p in self.pilots[3]:
            if not p.duty[d]:
                p.assign_group(Group_Flight(g.flights[0]))
                p.groups[d][-1].set_type(2)
                flag_3 = True
                p.add_group(g_zhen)
                p.groups[g_zhen.d_d][-1].set_type(3)
                break

        if not flag_3:
            for p in self.pilots[2]:
                if not p.duty[d]:
```

```
                    p.assign_group(Group_Flight(g.flights[0]))
                    p.groups[d][-1].set_type(2)
                    flag_3 = True
                    p.add_group(g_zhen)
                    p.groups[g_zhen.d_d][-1].set_type(3)
                    break
        return flag_3

    def assign_p_to_group_zaiFeiShuangXiang_1(self, g, d):
        flag_pre_1 = False  # 是否分配完前面来的人
        for p in self.pilots[1]:
            if (len(p.groups[d - 1]) > 1):
                continue
            for g_pre in p.groups[d - 1]:

                for f in g_pre.flights:
                    if g.flights[-1].p_d == g_pre.flights[-1].p_a:
                        p.add_group(Group_Flight(g.flights[-1]))
                        p.groups[g.flights[-1].d_d][-1].set_type(1)
                        flag_pre_1 = True
                        break
                if flag_pre_1:
                    break
            if flag_pre_1:
                break
        return flag_pre_1

    def assign_p_to_group_zaiFeiShuangXiang_3(self, g, d):
        flag_pre_1 = False  # 是否分配完前面来的人
        for p in self.pilots[3]:
            if (len(p.groups[d - 1]) > 1):
                continue
            for g_pre in p.groups[d - 1]:

                for f in g_pre.flights:
                    if g.flights[-1].p_d == g_pre.flights[-1].p_a:
                        p.add_group(Group_Flight(g.flights[-1]))
                        p.groups[g.flights[-1].d_d][-1].set_type(3)
                        flag_pre_1 = True
                        break
```

```
                    if flag_pre_1:
                        break
            if flag_pre_1:
                break
        return flag_pre_1

    def assign_p_to_group_zaiFeiShuangXiang_2(self, g, d, ty):
        flag_pre_1 = False  # 是否分配完前面来的人
        for p in self.pilots[2]:
            if (len(p.groups[d - 1]) > 1):
                continue
            for g_pre in p.groups[d - 1]:
                for f in g_pre.flights:
                    if g.flights[-1].p_d == g_pre.flights[-1].p_a:
                        p.add_group(Group_Flight(g.flights[-1]))
                        p.groups[g.flights[-1].d_d][-1].set_type(ty)
                        flag_pre_1 = True
                        break
                    if flag_pre_1:
                        break
            if flag_pre_1:
                break
        return flag_pre_1

    def assign_p_to_group_zaiFeiShuangXiang(self, g, d):
        flag_1 = self.assign_p_to_group_zaiFeiShuangXiang_1(g, d)
        flag_3 = self.assign_p_to_group_zaiFeiShuangXiang_3(g, d)
        if not flag_1:
            self.assign_p_to_group_zaiFeiShuangXiang_2(g, d, 1)
        if not flag_3:
            self.assign_p_to_group_zaiFeiShuangXiang_2(g, d, 3)
        self.assign_p_to_group(Group_Flight(g.flights[0]), g.flights[0].d_d)

    def assign_p_to_group_zaiFeiShuangXiang_qian_type(self, g, d, type):
        flag_pre = False  # 是否分配完前面来的人
        for p in self.pilots[type]:
            for g_pre in p.groups[d - 1]:
                if g.flights[-1] == g_pre.flights[0]:
                    p.add_group(Group_Flight(g.flights[-1]))
                    flag_pre = True
```

```
                    break
            if flag_pre:
                break
        return flag_pre

    def assign_p_to_group_zaiFeiShuangXiang_qian(self, g, d):
        flag_pre_1 = False  # 是否分配完前面来的人
        flag_pre_3 = False  # 是否分配完前面来的人
        flag_pre_1 = self.assign_p_to_group_zaiFeiShuangXiang_qian_type(g, d, 1)
        flag_pre_3 = self.assign_p_to_group_zaiFeiShuangXiang_qian_type(g, d, 3)
        if not flag_pre_1:
            flag_pre_1 = self.assign_p_to_group_zaiFeiShuangXiang_qian_type(g, d, 2)
        if not flag_pre_3:
            flag_pre_3 = self.assign_p_to_group_zaiFeiShuangXiang_qian_type(g, d, 2)
        return flag_pre_1, flag_pre_3

    def assign_p_to_group_dan(self, g, d):
        flag_qian_1 = False  # 是否前一天乘机过去了
        flag_qian_3 = False  # 是否前一天乘机过去了
        for f in self.map_t2f[d - 1]:
            if f.p_a == g.p_a:
                flag_qian_1 = self.assign_p_to_group_chengji_1(Group_Flight(f), d - 1, g)
                flag_qian_3 = self.assign_p_to_group_chengji_3(Group_Flight(f), d - 1, g)
                if flag_qian_1 and flag_qian_3:
                    break
        if not (flag_qian_1 and flag_qian_3):
            for f in self.map_t2f[d]:
                if f.p_a == g.p_a:
                    if not flag_qian_1:
                        flag_qian_1 = self.assign_p_to_group_chengji_1(Group_Flight(f), d, g)
                    if not flag_qian_3:
                        flag_qian_3 = self.assign_p_to_group_chengji_3(Group_Flight(f), d, g)
                    if flag_qian_1 and flag_qian_3:
                        break
        return flag_qian_1, flag_qian_3

    def run_jiao_p2(self):
        self.make_group()
        self.allocate_p2()
        self.print_result_p2()
```

```
1880
1881     def run_jiao_p1(self):
1882         self.make_group()
1883         self.coombine_group()
1884         self.allocate_pilot()
1885         self.print_result_p1()
1886
1887     def tanlan_p1_B(self):
1888         ordered_flight = copy.deepcopy(self.map_t2f)
1889         num_un = 0
1890         table_un = pd.DataFrame()
1891         unallocated_g = []
1892         self.chengji = 0
1893         for d in range(1, NUM_D + 1):
1894             print(d)
1895             ordered_flight[d].sort()
1896             alled_f = []
1897             while len(ordered_flight[d]) - len(alled_f) > 0:
1898                 print(len(ordered_flight[d]) - len(alled_f))
1899                 g = Group_Flight(ordered_flight[d][0])
1900                 f_fir = None
1901                 for f in range(0, len(ordered_flight[d])):
1902                     if f not in alled_f:
1903                         g = Group_Flight(ordered_flight[d][f])
1904                         alled_f.append(f)
1905                         f_fir = f
1906                         break
1907                 ind_del_f = [0]
1908                 if g.p_d in JIDI:
1909                     for f in range(f_fir, len(ordered_flight[d])):
1910                         if f in alled_f:
1911                             continue
1912                         if ordered_flight[d][f].p_d == g.p_a and ordered_flight[d][f].t_d -
g.t_a >= LowerLimit_Combine:
1913                             g.add_flight(ordered_flight[d][f])
1914                             alled_f.append(f)
1915                             ind_del_f.append(f)
1916                             if g.p_a in JIDI:
1917                                 break
1918
```

```
                # added_f = []
                # flag_xinhuilu = False
                # temp_g=copy.deepcopy(g)
                # for f in range(ind_del_f[-1], len(ordered_flight[d])):
                #     if f in alled_f:
                #         continue
                #     if ordered_flight[d][f].p_d == temp_g.p_a and ordered_flight[d][
                #         f].t_d - temp_g.t_a >= LowerLimit_Combine:
                #         added_f.append(f)
                #         temp_g.add_flight(ordered_flight[d][f])
                #         # alled_f.append(f)
                #         # ind_del_f.append(f)
                #         if temp_g.p_a == g.p_a:
                #             flag_xinhuilu=True
                #             break
                # if flag_xinhuilu:
                #     for f in added_f:
                #         g.add_flight(ordered_flight[d][f])
                #         alled_f.append(f)
                #         ind_del_f.append(f)

                # print(ordered_flight[d][f].id)
                # for f in reversed(ind_del_f):
                #     print(ordered_flight[d][f].id)
                #     del ordered_flight[d][f]
                flag_1 = False
                flag_3 = False
                for p in self.pilots[1]:
                    if p.p_a_d[d - 1] == g.p_d and g.t_d - p.t_a_d[d - 1] >= LowerLimit_Combine:
                        if p.p_a_d[d] == g.p_d and p.t_d_d[d] - g.t_a >= LowerLimit_Combine:
                            p.add_group_first(g)
                            p.groups[d][0].set_type(1)
                            flag_1 = True
                            break
                        if p.p_d_d[d] == g.p_a and g.t_d - p.t_a_d[d] >= LowerLimit_Combine:
                            p.add_group(g)
                            p.groups[d][-1].set_type(1)
                            flag_1 = True
                            break
```

```
                            if (not p.duty[d]):
                                p.add_group(g)
                                p.groups[d][-1].set_type(1)
                                flag_1 = True
                                break
                    for p in self.pilots[3]:
                        if p.p_a_d[d - 1] == g.p_d and g.t_d - p.t_a_d[d - 1] >= LowerLimit_Combine:
                            if p.p_a_d[d] == g.p_d and p.t_d_d[d] - g.t_a >= LowerLimit_Combine:
                                p.add_group_first(g)
                                p.groups[d][0].set_type(3)
                                flag_3 = True
                                break
                            if p.p_d_d[d] == g.p_a and g.t_d - p.t_a_d[d] >= LowerLimit_Combine:
                                p.add_group(g)
                                p.groups[d][-1].set_type(3)
                                flag_3 = True
                                break

                            if (not p.duty[d]):
                                p.add_group(g)
                                p.groups[d][-1].set_type(3)
                                flag_3 = True
                                break
                            # p.add_group(g)
                            # p.groups[d][-1].set_type(3)
                            # flag_3 = True
                            # break
                    if not flag_1:
                        for p in self.pilots[2]:
                            if p.p_a_d[d - 1] == g.p_d and g.t_d - p.t_a_d[d - 1] >=
LowerLimit_Combine:
                                if p.p_a_d[d] == g.p_d and p.t_d_d[d] - g.t_a >= LowerLimit_Combine:
                                    p.add_group_first(g)
                                    p.groups[d][0].set_type(1)
                                    flag_1 = True
                                    break
                                if p.p_d_d[d] == g.p_a and g.t_d - p.t_a_d[d] >= LowerLimit_Combine:
                                    p.add_group(g)
                                    p.groups[d][-1].set_type(1)
                                    flag_1 = True
```

```
                                    break
                                if (not p.duty[d]):
                                    p.add_group(g)
                                    p.groups[d][-1].set_type(1)
                                    flag_1 = True
                                    break
                                # p.add_group(g)
                                # p.groups[d][-1].set_type(1)
                                # flag_1 = True
                                # break
                    if not flag_3:
                        for p in self.pilots[2]:
                            if p.p_a_d[d - 1] == g.p_d and g.t_d - p.t_a_d[d - 1] >= LowerLimit_Combine:
                                if p.p_a_d[d] == g.p_d and p.t_d_d[d] - g.t_a >= LowerLimit_Combine:
                                    p.add_group_first(g)
                                    p.groups[d][0].set_type(3)
                                    flag_3 = True
                                    break
                                if p.p_d_d[d] == g.p_a and g.t_d - p.t_a_d[d] >= LowerLimit_Combine:
                                    p.add_group(g)
                                    p.groups[d][-1].set_type(3)
                                    flag_3 = True
                                    break

                                if (not p.duty[d]):
                                    p.add_group(g)
                                    p.groups[d][-1].set_type(3)
                                    flag_3 = True
                                    break
                                # p.add_group(g)
                                # p.groups[d][-1].set_type(3)
                                # flag_3 = True
                                # break

                    if not (flag_1 and flag_3):
                        for f in self.map_t2f[d - 1]:
                            if f.p_d in JIDI and g.p_d == f.p_a and g.t_d - f.t_a >= LowerLimit_Combine:
```

```
                        if not flag_1:
                            for p in self.pilots[1]:
                                if p.p_a_d[max(d - 2, 0)] == f.p_d and f.t_d - p.t_a_d[
                                    max(d - 2, 0)] >= LowerLimit_Combine and (not p.duty[d
- 1]) and (
                                not p.duty[d]):
                                    p.add_group(Group_Flight(f))
                                    p.groups[d - 1][-1].set_type(2)
                                    self.chengji = self.chengji + 1
                                    p.add_group(g)
                                    p.groups[d][-1].set_type(1)
                                    flag_1 = True
                                    break
                        if not flag_3:
                            for p in self.pilots[3]:
                                if p.p_a_d[max(d - 2, 0)] == f.p_d and f.t_d - p.t_a_d[
                                    max(d - 2, 0)] >= LowerLimit_Combine and (not p.duty[d
- 1]) and (
                                not p.duty[d]):
                                    p.add_group(Group_Flight(f))
                                    p.groups[d - 1][-1].set_type(2)
                                    self.chengji = self.chengji + 1
                                    p.add_group(g)
                                    p.groups[d][-1].set_type(3)
                                    flag_3 = True
                                    break
                        if not flag_1:
                            for p in self.pilots[2]:
                                if p.p_a_d[max(d - 2, 0)] == f.p_d and f.t_d - p.t_a_d[
                                    max(d - 2, 0)] >= LowerLimit_Combine and (not p.duty[d
- 1]) and (
                                not p.duty[d]):
                                    p.add_group(Group_Flight(f))
                                    p.groups[d - 1][-1].set_type(2)
                                    self.chengji = self.chengji + 1
                                    p.add_group(g)
                                    p.groups[d][-1].set_type(1)
                                    flag_1 = True
                                    break
                        if not flag_3:
```

```
                                for p in self.pilots[2]:
                                    if p.p_a_d[max(d - 2, 0)] == f.p_d and f.t_d - p.t_a_d[
                                        max(d - 2, 0)] >= LowerLimit_Combine and (not p.duty[d
- 1]) and (
                                    not p.duty[d]):
                                        p.add_group(Group_Flight(f))
                                        p.groups[d - 1][-1].set_type(2)
                                        self.chengji = self.chengji + 1
                                        p.add_group(g)
                                        p.groups[d][-1].set_type(3)
                                        flag_3 = True
                                        break
            temp_g = copy.deepcopy(g)
            if not (flag_1 and flag_3):
                flag_qian = False
                flag_hou = False
                for f in self.map_t2f[d]:
                    if f.p_d in JIDI and f.p_a == g.p_d and g.t_d - f.t_a >=
LowerLimit_Combine:
                        temp_g.add_flight_first(f)
                        flag_qian = True
                        break
                if flag_qian:
                    for f in self.map_t2f[d]:
                        if f.p_a in JIDI and f.p_d == g.p_a and f.t_d - g.t_a >=
LowerLimit_Combine:
                            temp_g.add_flight(f)
                            flag_hou = True

                            break
                if flag_qian and flag_hou:
                    # flag_1 = False
                    # flag_3 = False
                    #
print("99999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999")
                    for p in self.pilots[1]:
                        if p.p_a_d[d - 1] == temp_g.p_d and temp_g.t_d - p.t_a_d[d - 1] >=
LowerLimit_Combine and (
                        not p.duty[d]):
                            p.add_group(Group_Flight(temp_g.flights[0]))
```

```
                                p.groups[d][-1].set_type(2)
                                p.add_group(Group_Flight(temp_g.flights[1]))
                                p.groups[d][-1].set_type(1)
                                p.add_group(Group_Flight(temp_g.flights[2]))
                                p.groups[d][-1].set_type(2)
                                self.chengji = self.chengji + 2
                                flag_1 = True
                                break
                        for p in self.pilots[3]:
                            if p.p_a_d[d - 1] == temp_g.p_d and temp_g.t_d - p.t_a_d[d - 1] >= LowerLimit_Combine and (
                            not p.duty[d]):
                                p.add_group(Group_Flight(temp_g.flights[0]))
                                p.groups[d][-1].set_type(2)
                                p.add_group(Group_Flight(temp_g.flights[1]))
                                p.groups[d][-1].set_type(3)
                                p.add_group(Group_Flight(temp_g.flights[2]))
                                p.groups[d][-1].set_type(2)
                                self.chengji = self.chengji + 2
                                flag_3 = True
                                break
                        if not flag_1:
                            for p in self.pilots[2]:
                                if p.p_a_d[d - 1] == temp_g.p_d and temp_g.t_d - p.t_a_d[
                                    d - 1] >= LowerLimit_Combine and (not p.duty[d]):
                                    p.add_group(Group_Flight(temp_g.flights[0]))
                                    p.groups[d][-1].set_type(2)
                                    p.add_group(Group_Flight(temp_g.flights[1]))
                                    p.groups[d][-1].set_type(1)
                                    p.add_group(Group_Flight(temp_g.flights[2]))
                                    p.groups[d][-1].set_type(2)
                                    self.chengji = self.chengji + 2
                                    flag_1 = True
                                    break
                        if not flag_3:
                            for p in self.pilots[2]:
                                if p.p_a_d[d - 1] == temp_g.p_d and temp_g.t_d - p.t_a_d[
                                    d - 1] >= LowerLimit_Combine and (not p.duty[d]):
                                    p.add_group(Group_Flight(temp_g.flights[0]))
                                    p.groups[d][-1].set_type(2)
```

```
                                    p.add_group(Group_Flight(temp_g.flights[1]))
                                    p.groups[d][-1].set_type(3)
                                    p.add_group(Group_Flight(temp_g.flights[2]))
                                    p.groups[d][-1].set_type(2)
                                    self.chengji = self.chengji + 2
                                    flag_3 = True
                                    break

                if not (flag_1 and flag_3):
                    unallocated_g.append(g)
                    num_un = num_un + 1
                    for f in g.flights:
                        table_un = table_un.append({'FltNum': f.id, 'DptrT': f.t_d, 'ArrvT': f.t_a, "DrtrD": f.d_d},
                                                   ignore_index=True)

            for p in self.pilots[1]:
                if p.p_a_d[d - 1] is not p.jidi and (not p.duty[d]):
                    for f in self.map_t2f[d]:
                        if f.p_d == p.p_a_d[d - 1] and f.p_a == p.jidi:
                            p.add_group(Group_Flight(f))
                            p.groups[d][-1].set_type(2)
                            self.chengji = self.chengji + 1
                            print("1111111111111111111111111111111111111111111")
                            break
        self.print_result_p1()

        table_un.to_csv("UncoveredFlights_p1_B.csv", index=False)
        print(num_un)
        print(f"{self.chengji} 乘機次數")
        print("第一天 duty 人数:", sum([p.duty[1] for p in self.pilots[1] + self.pilots[2] + self.pilots[3]]))

    def tanlan_p2_B(self):
        ordered_flight = copy.deepcopy(self.map_t2f)
        num_un = 0
        table_un = pd.DataFrame()
        unallocated_g = []
        self.chengji = 0
        for d in range(1, NUM_D + 1):
```

```
            print(d)
            ordered_flight[d].sort()
            alled_f = []
            while len(ordered_flight[d]) - len(alled_f) > 0:
                print(len(ordered_flight[d]) - len(alled_f))
                g = Group_Flight(ordered_flight[d][0])
                f_fir = None
                for f in range(0, len(ordered_flight[d])):
                    if f not in alled_f:
                        g = Group_Flight(ordered_flight[d][f])
                        alled_f.append(f)
                        f_fir = f
                        break
                ind_del_f = [0]
                if g.p_d in JIDI:
                    for f in range(f_fir, len(ordered_flight[d])):
                        if f in alled_f:
                            continue
                        if ordered_flight[d][f].p_d == g.p_a and ordered_flight[d][f].t_d - g.t_a >= LowerLimit_Combine and ordered_flight[d][f].t_d - g.t_a <= UpperLimit_Combine:
                            if (g.time_fly + ordered_flight[d][f].t_a - ordered_flight[d][f].t_d) > MaxBlk:
                                continue
                            if (ordered_flight[d][f].t_a - g.t_d) > MaxDP:
                                continue
                            g.add_flight(ordered_flight[d][f])
                            alled_f.append(f)
                            ind_del_f.append(f)
                            if g.p_a in JIDI:
                                break

                # added_f = []
                # flag_xinhuilu = False
                # temp_g=copy.deepcopy(g)
                # for f in range(ind_del_f[-1], len(ordered_flight[d])):
                #     if f in alled_f:
                #         continue
                #     if ordered_flight[d][f].p_d == temp_g.p_a and ordered_flight[d][
                #         f].t_d - temp_g.t_a >= LowerLimit_Combine:
                #         added_f.append(f)
```

```
                    #         temp_g.add_flight(ordered_flight[d][f])
                    #         # alled_f.append(f)
                    #         # ind_del_f.append(f)
                    #         if temp_g.p_a == g.p_a:
                    #             flag_xinhuilu=True
                    #             break
                    # if flag_xinhuilu:
                    #     for f in added_f:
                    #         g.add_flight(ordered_flight[d][f])
                    #         alled_f.append(f)
                    #         ind_del_f.append(f)

                    # print(ordered_flight[d][f].id)
                    # for f in reversed(ind_del_f):
                    #     print(ordered_flight[d][f].id)
                    #     del ordered_flight[d][f]
                    flag_1 = False
                    flag_3 = False
                    ### 先分配今天剩余没用的人 ###
                    for p in self.pilots[1]:
                        if p.p_a_d[d - 1] == g.p_d and g.t_d - p.t_a_d[d - 1] >= LowerLimit_Combine and (g.t_d - p.t_a_d[d - 1] <= UpperLimit_Combine):
                            if p.p_a_d[d] == g.p_d and p.t_d_d[d] - g.t_a >= LowerLimit_Combine and (p.t_d_d[d] - g.t_a<=UpperLimit_Combine):
                                if g.t_d - p.t_a_d[d - 1] >= MinRest and (p.t_d_a[d] - g.t_d<=MaxDP) and (p.time_fly[d]+g.time_fly<MaxBlk):
                                    p.add_group_first(g)
                                    p.groups[d][0].set_type(1)
                                    flag_1 = True
                                    break
                            if p.p_d_d[d] == g.p_a and g.t_d - p.t_a_d[d] >= LowerLimit_Combine and (p.time_fly[d]+g.time_fly<MaxBlk) and (g.t_d - p.t_a_d[d]<=UpperLimit_Combine):
                                if (g.t_a-p.t_d_d[d]<=MaxDP):
                                    p.add_group(g)
                                    p.groups[d][-1].set_type(1)
                                    flag_1 = True
                                    break

                            if (not p.duty[d]):
                                if g.t_d - p.t_a_d[d - 1] >= MinRest:
```

```
                                p.add_group(g)
                                p.groups[d][-1].set_type(1)
                                flag_1 = True
                                break
                    for p in self.pilots[3]:
                        if p.p_a_d[d - 1] == g.p_d and g.t_d - p.t_a_d[d - 1] >= LowerLimit_Combine and (g.t_d - p.t_a_d[d - 1]<=UpperLimit_Combine):
                            if p.p_a_d[d] == g.p_d and p.t_d_d[d] - g.t_a >= LowerLimit_Combine and (p.t_d_d[d] - g.t_a<=UpperLimit_Combine):
                                if g.t_d - p.t_a_d[d - 1] >= MinRest and (p.t_d_a[d] - g.t_d<=MaxDP) and (p.time_fly[d]+g.time_fly<MaxBlk):
                                p.add_group_first(g)
                                p.groups[d][0].set_type(3)
                                flag_3 = True
                                break
                            if p.p_d_d[d] == g.p_a and g.t_d - p.t_a_d[d] >= LowerLimit_Combine and (g.t_d - p.t_a_d[d]<=UpperLimit_Combine):
                                if (g.t_a - p.t_d_d[d] <= MaxDP) and (p.time_fly[d]+g.time_fly<MaxBlk):
                                p.add_group(g)
                                p.groups[d][-1].set_type(3)
                                flag_3 = True
                                break

                            if (not p.duty[d]):
                                if g.t_d - p.t_a_d[d - 1] >= MinRest:
                                p.add_group(g)
                                p.groups[d][-1].set_type(3)
                                flag_3 = True
                                break
                            # p.add_group(g)
                            # p.groups[d][-1].set_type(3)
                            # flag_3 = True
                            # break
                    if not flag_1:
                        for p in self.pilots[2]:
                            if p.p_a_d[d - 1] == g.p_d and g.t_d - p.t_a_d[d - 1] >= LowerLimit_Combine and (g.t_d - p.t_a_d[d - 1]<=UpperLimit_Combine):
                                if p.p_a_d[d] == g.p_d and p.t_d_d[d] - g.t_a >= LowerLimit_Combine and (p.t_d_d[d] - g.t_a<=UpperLimit_Combine):
```

```
                                    if g.t_d - p.t_a_d[d - 1] >= MinRest and (p.t_d_a[d] - g.t_d<=MaxDP) and (p.time_fly[d]+g.time_fly<MaxBlk):
                                        p.add_group_first(g)
                                        p.groups[d][0].set_type(1)
                                        flag_1 = True
                                        break
                                if p.p_d_d[d] == g.p_a and g.t_d - p.t_a_d[d] >= LowerLimit_Combine and (g.t_d - p.t_a_d[d]<=UpperLimit_Combine):
                                    if (g.t_a - p.t_d_d[d] <= MaxDP) and (p.time_fly[d]+g.time_fly<MaxBlk):
                                        p.add_group(g)
                                        p.groups[d][-1].set_type(1)
                                        flag_1 = True
                                        break

                            if (not p.duty[d]):
                                if g.t_d - p.t_a_d[d - 1] >= MinRest:
                                    p.add_group(g)
                                    p.groups[d][-1].set_type(1)
                                    flag_1 = True
                                    break
                            # p.add_group(g)
                            # p.groups[d][-1].set_type(1)
                            # flag_1 = True
                            # break
                if not flag_3:
                    for p in self.pilots[2]:
                        if p.p_a_d[d - 1] == g.p_d and g.t_d - p.t_a_d[d - 1] >= LowerLimit_Combine and (g.t_d - p.t_a_d[d - 1])<=UpperLimit_Combine:
                            if p.p_a_d[d] == g.p_d and p.t_d_d[d] - g.t_a >= LowerLimit_Combine and (p.t_d_d[d] - g.t_a)<=UpperLimit_Combine:
                                if g.t_d - p.t_a_d[d - 1] >= MinRest and (p.t_d_a[d] - g.t_d<=MaxDP) and (p.time_fly[d]+g.time_fly<MaxBlk):
                                    p.add_group_first(g)
                                    p.groups[d][0].set_type(3)
                                    flag_3 = True
                                    break
                            if p.p_d_d[d] == g.p_a and g.t_d - p.t_a_d[d] >= LowerLimit_Combine and (g.t_d - p.t_a_d[d]<=UpperLimit_Combine):
                                if (g.t_a - p.t_d_d[d] <= MaxDP) and
```

```
(p.time_fly[d]+g.time_fly<MaxBlk):
                                    p.add_group(g)
                                    p.groups[d][-1].set_type(3)
                                    flag_3 = True
                                    break

                            if (not p.duty[d]):
                                if g.t_d - p.t_a_d[d - 1] >= MinRest:
                                    p.add_group(g)
                                    p.groups[d][-1].set_type(3)
                                    flag_3 = True
                                    break
                            # p.add_group(g)
                            # p.groups[d][-1].set_type(3)
                            # flag_3 = True
                            # break
                    ### 选择前一天的航班乘机过来 ###
                    if not (flag_1 and flag_3):
                        for f in self.map_t2f[d - 1]:
                            if f.p_d in JIDI and g.p_d == f.p_a and g.t_d - f.t_a >=
LowerLimit_Combine and (g.t_d - f.t_a<=UpperLimit_Combine):
                                if g.t_d - f.t_a < MinRest:
                                    continue
                                if not flag_1:
                                    for p in self.pilots[1]:
                                        if p.p_a_d[max(d - 2, 0)] == f.p_d and f.t_d - p.t_a_d[
                                            max(d - 2, 0)] >= LowerLimit_Combine and (not p.duty[d
- 1]) and (
                                            not p.duty[d]) and (f.t_d - p.t_a_d[max(d - 2,
0)]<=UpperLimit_Combine):
                                            p.add_group(Group_Flight(f))
                                            p.groups[d - 1][-1].set_type(2)
                                            self.chengji = self.chengji + 1
                                            p.add_group(g)
                                            p.groups[d][-1].set_type(1)
                                            flag_1 = True
                                            break
                                if not flag_3:
                                    for p in self.pilots[3]:
                                        if p.p_a_d[max(d - 2, 0)] == f.p_d and f.t_d - p.t_a_d[
```

```
                                        max(d - 2, 0)] >= LowerLimit_Combine and (not p.duty[d - 1]) and (
                                    not p.duty[d]) and (f.t_d - p.t_a_d[
                                        max(d - 2, 0)]<=UpperLimit_Combine):
                                        p.add_group(Group_Flight(f))
                                        p.groups[d - 1][-1].set_type(2)
                                        self.chengji = self.chengji + 1
                                        p.add_group(g)
                                        p.groups[d][-1].set_type(3)
                                        flag_3 = True
                                        break
                                if not flag_1:
                                    for p in self.pilots[2]:
                                        if p.p_a_d[max(d - 2, 0)] == f.p_d and f.t_d - p.t_a_d[
                                        max(d - 2, 0)] >= LowerLimit_Combine and (not p.duty[d - 1]) and (
                                    not p.duty[d]) and (f.t_d - p.t_a_d[
                                        max(d - 2, 0)]<=UpperLimit_Combine):
                                        p.add_group(Group_Flight(f))
                                        p.groups[d - 1][-1].set_type(2)
                                        self.chengji = self.chengji + 1
                                        p.add_group(g)
                                        p.groups[d][-1].set_type(1)
                                        flag_1 = True
                                        break
                                if not flag_3:
                                    for p in self.pilots[2]:
                                        if p.p_a_d[max(d - 2, 0)] == f.p_d and f.t_d - p.t_a_d[
                                        max(d - 2, 0)] >= LowerLimit_Combine and (not p.duty[d - 1]) and (
                                    not p.duty[d]) and (f.t_d - p.t_a_d[
                                        max(d - 2, 0)] <=UpperLimit_Combine):
                                        p.add_group(Group_Flight(f))
                                        p.groups[d - 1][-1].set_type(2)
                                        self.chengji = self.chengji + 1
                                        p.add_group(g)
                                        p.groups[d][-1].set_type(3)
                                        flag_3 = True
                                        break
                    ### 在同一天找前面后面各一驾航班 乘两次机 ###
```

```
                    temp_g = copy.deepcopy(g)
                    if not (flag_1 and flag_3):
                        flag_qian = False
                        flag_hou = False
                        for f in reversed(self.map_t2f[d]):
                            if f.p_d in JIDI and f.p_a == g.p_d and g.t_d - f.t_a >= LowerLimit_Combine and (g.t_d - f.t_a<=UpperLimit_Combine):
                                temp_g.add_flight_first(f)
                                if temp_g.time_fly > MaxBlk:
                                    flag_qian = False
                                    break
                                if temp_g.t_a - temp_g.t_d > MaxDP:
                                    flag_qian = False
                                    break
                                flag_qian = True
                                break
                        if flag_qian:
                            for f in self.map_t2f[d]:
                                if f.p_a in JIDI and f.p_d == g.p_a and f.t_d - g.t_a >= LowerLimit_Combine and (f.t_d - g.t_a<=UpperLimit_Combine):
                                    temp_g.add_flight(f)
                                    if temp_g.time_fly > MaxBlk:
                                        flag_hou = False
                                        break
                                    if temp_g.t_a - temp_g.t_d > MaxDP:
                                        flag_qian = False
                                        break
                                    flag_hou = True

                                    break
                        if flag_qian and flag_hou:
                            # flag_1 = False
                            # flag_3 = False
                            # print("99999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999")
                            for p in self.pilots[1]:
                                if temp_g.t_d - p.t_a_d[d - 1] < MinRest:
                                    continue
                                if p.p_a_d[d - 1] == temp_g.p_d and temp_g.t_d - p.t_a_d[d - 1] >= LowerLimit_Combine and (
```

```
                        not p.duty[d]) and (temp_g.t_d - p.t_a_d[d -
1]<=UpperLimit_Combine):
                            p.add_group(Group_Flight(temp_g.flights[0]))
                            p.groups[d][-1].set_type(2)
                            p.add_group(Group_Flight(temp_g.flights[1]))
                            p.groups[d][-1].set_type(1)
                            p.add_group(Group_Flight(temp_g.flights[2]))
                            p.groups[d][-1].set_type(2)
                            self.chengji = self.chengji + 2
                            flag_1 = True
                            break
                    for p in self.pilots[3]:
                        if p.p_a_d[d - 1] == temp_g.p_d and temp_g.t_d - p.t_a_d[d - 1] >=
LowerLimit_Combine and (
                        not p.duty[d]) and (temp_g.t_d - p.t_a_d[d -
1]<=UpperLimit_Combine):
                            p.add_group(Group_Flight(temp_g.flights[0]))
                            p.groups[d][-1].set_type(2)
                            p.add_group(Group_Flight(temp_g.flights[1]))
                            p.groups[d][-1].set_type(3)
                            p.add_group(Group_Flight(temp_g.flights[2]))
                            p.groups[d][-1].set_type(2)
                            self.chengji = self.chengji + 2
                            flag_3 = True
                            break
                    if not flag_1:
                        for p in self.pilots[2]:
                            if p.p_a_d[d - 1] == temp_g.p_d and temp_g.t_d - p.t_a_d[
                                d - 1] >= LowerLimit_Combine and (not p.duty[d]) and
(temp_g.t_d - p.t_a_d[
                                d - 1]<=UpperLimit_Combine):
                                p.add_group(Group_Flight(temp_g.flights[0]))
                                p.groups[d][-1].set_type(2)
                                p.add_group(Group_Flight(temp_g.flights[1]))
                                p.groups[d][-1].set_type(1)
                                p.add_group(Group_Flight(temp_g.flights[2]))
                                p.groups[d][-1].set_type(2)
                                self.chengji = self.chengji + 2
                                flag_1 = True
                                break
```

```
                            if not flag_3:
                                for p in self.pilots[2]:
                                    if p.p_a_d[d - 1] == temp_g.p_d and temp_g.t_d - p.t_a_d[
                                        d - 1] >= LowerLimit_Combine and (not p.duty[d]) and (temp_g.t_d - p.t_a_d[
                                        d - 1]<=UpperLimit_Combine):
                                        p.add_group(Group_Flight(temp_g.flights[0]))
                                        p.groups[d][-1].set_type(2)
                                        p.add_group(Group_Flight(temp_g.flights[1]))
                                        p.groups[d][-1].set_type(3)
                                        p.add_group(Group_Flight(temp_g.flights[2]))
                                        p.groups[d][-1].set_type(2)
                                        self.chengji = self.chengji + 2
                                        flag_3 = True
                                        break

                    if not (flag_1 and flag_3):
                        unallocated_g.append(g)
                        num_un = num_un + 1
                        for f in g.flights:
                            table_un = table_un.append({'FltNum': f.id, 'DptrT': f.t_d, 'ArrvT': f.t_a, "DrtrD": f.d_d},
                                                       ignore_index=True)

                for p in self.pilots[1]:
                    if p.p_a_d[d - 1] is not p.jidi and (not p.duty[d]):
                        for f in self.map_t2f[d]:
                            if f.p_d == p.p_a_d[d - 1] and f.p_a == p.jidi:
                                p.add_group(Group_Flight(f))
                                p.groups[d][-1].set_type(2)
                                self.chengji = self.chengji + 1
                                print("1111111111111111111111111111111111111111111")
                                break
        self.print_result_p2()

        table_un.to_csv("UncoveredFlights_p2_A.csv", index=False)
        print(num_un)
        print(f"{self.chengji} 乘機次数")
        print("第一天 duty 人数:", sum([p.duty[1] for p in self.pilots[1] + self.pilots[2] + self.pilots[3]]))
```

```
        total_fly = 0
        if len(table_un)>0:
            list_unall=table_un["FltNum"].to_list()

            for d in range(1,NUM_D+1):
                for f in self.map_t2f[d]:
                    if f.id not in list_unall:
                        total_fly=total_fly+(f.t_a-f.t_d)*2
        else:
            for d in range(1, NUM_D + 1):
                for f in self.map_t2f[d]:
                    total_fly = total_fly + (f.t_a - f.t_d) * 2
        print("机组总体利用率: ", total_fly/self.total_zhiqin)

    def tanlan_p3_B(self):
        ordered_flight = copy.deepcopy(self.map_t2f)
        num_un = 0
        table_un = pd.DataFrame()
        unallocated_g = []
        self.chengji = 0
        for d in range(1, NUM_D + 1):
            print(d)
            ordered_flight[d].sort()
            alled_f = []
            while len(ordered_flight[d]) - len(alled_f) > 0:
                print(len(ordered_flight[d]) - len(alled_f))
                g = Group_Flight(ordered_flight[d][0])
                f_fir = None
                for f in range(0, len(ordered_flight[d])):
                    if f not in alled_f:
                        g = Group_Flight(ordered_flight[d][f])
                        alled_f.append(f)
                        f_fir = f
                        break
                ind_del_f = [0]
                if g.p_d in JIDI:
                    for f in range(f_fir, len(ordered_flight[d])):
                        if f in alled_f:
                            continue
                        if ordered_flight[d][f].p_d == g.p_a and ordered_flight[d][f].t_d -
```

```
                g.t_a >= LowerLimit_Combine:
2547                            if (g.time_fly + ordered_flight[d][f].t_a -
ordered_flight[d][f].t_d) > MaxBlk:
2548                                continue
2549                            if (ordered_flight[d][f].t_a - g.t_d) > MaxDP:
2550                                continue
2551                            g.add_flight(ordered_flight[d][f])
2552                            alled_f.append(f)
2553                            ind_del_f.append(f)
2554                            if g.p_a in JIDI:
2555                                break
2556
2557                    # added_f = []
2558                    # flag_xinhuilu = False
2559                    # temp_g=copy.deepcopy(g)
2560                    # for f in range(ind_del_f[-1], len(ordered_flight[d])):
2561                    #     if f in alled_f:
2562                    #         continue
2563                    #     if ordered_flight[d][f].p_d == temp_g.p_a and ordered_flight[d][
2564                    #         f].t_d - temp_g.t_a >= LowerLimit_Combine:
2565                    #         added_f.append(f)
2566                    #         temp_g.add_flight(ordered_flight[d][f])
2567                    #         # alled_f.append(f)
2568                    #         # ind_del_f.append(f)
2569                    #         if temp_g.p_a == g.p_a:
2570                    #             flag_xinhuilu=True
2571                    #             break
2572                    # if flag_xinhuilu:
2573                    #     for f in added_f:
2574                    #         g.add_flight(ordered_flight[d][f])
2575                    #         alled_f.append(f)
2576                    #         ind_del_f.append(f)
2577
2578                    # print(ordered_flight[d][f].id)
2579                    # for f in reversed(ind_del_f):
2580                    #     print(ordered_flight[d][f].id)
2581                    #     del ordered_flight[d][f]
2582                    flag_1 = False
2583                    flag_3 = False
2584                    ### 先分配今天剩余没用的人 ###
```

```
                for p in self.pilots[1]:
                    if p.p_a_d[d-1] == g.p_d and g.t_d - p.t_a_d[d-1] >= LowerLimit_Combine:
                        if p.p_a_d[d] == g.p_d and p.t_d_d[d] - g.t_a >= LowerLimit_Combine:
                            if g.t_d - p.t_a_d[d-1] >= MinRest and (p.t_d_a[d] - g.t_d<=MaxDP) and (p.time_fly[d]+g.time_fly<MaxBlk):
                                p.add_group_first(g)
                                p.groups[d][0].set_type(1)
                                flag_1 = True
                                break
                        if p.p_d_d[d] == g.p_a and g.t_d - p.t_a_d[d] >= LowerLimit_Combine and (p.time_fly[d]+g.time_fly<MaxBlk):
                            if (g.t_a-p.t_d_d[d]<=MaxDP):
                                p.add_group(g)
                                p.groups[d][-1].set_type(1)
                                flag_1 = True
                                break

                    if (not p.duty[d]):
                        if g.t_d - p.t_a_d[d-1] >= MinRest:
                            p.add_group(g)
                            p.groups[d][-1].set_type(1)
                            flag_1 = True
                            break
                for p in self.pilots[3]:
                    if p.p_a_d[d-1] == g.p_d and g.t_d - p.t_a_d[d-1] >= LowerLimit_Combine:
                        if p.p_a_d[d] == g.p_d and p.t_d_d[d] - g.t_a >= LowerLimit_Combine:
                            if g.t_d - p.t_a_d[d-1] >= MinRest and (p.t_d_a[d] - g.t_d<=MaxDP) and (p.time_fly[d]+g.time_fly<MaxBlk):
                                p.add_group_first(g)
                                p.groups[d][0].set_type(3)
                                flag_3 = True
                                break
                        if p.p_d_d[d] == g.p_a and g.t_d - p.t_a_d[d] >= LowerLimit_Combine:
                            if (g.t_a - p.t_d_d[d] <= MaxDP) and (p.time_fly[d]+g.time_fly<MaxBlk):
                                p.add_group(g)
                                p.groups[d][-1].set_type(3)
                                flag_3 = True
                                break
```

```
                        if (not p.duty[d]):
                            if g.t_d - p.t_a_d[d - 1] >= MinRest:
                                p.add_group(g)
                                p.groups[d][-1].set_type(3)
                                flag_3 = True
                                break
                        # p.add_group(g)
                        # p.groups[d][-1].set_type(3)
                        # flag_3 = True
                        # break
                if not flag_1:
                    for p in self.pilots[2]:
                        if p.p_a_d[d - 1] == g.p_d and g.t_d - p.t_a_d[d - 1] >= LowerLimit_Combine:
                            if p.p_a_d[d] == g.p_d and p.t_d_d[d] - g.t_a >= LowerLimit_Combine:
                                if g.t_d - p.t_a_d[d - 1] >= MinRest and (p.t_d_a[d] - g.t_d<=MaxDP) and (p.time_fly[d]+g.time_fly<MaxBlk):
                                    p.add_group_first(g)
                                    p.groups[d][0].set_type(1)
                                    flag_1 = True
                                    break
                            if p.p_d_d[d] == g.p_a and g.t_d - p.t_a_d[d] >= LowerLimit_Combine:
                                if (g.t_a - p.t_d_d[d] <= MaxDP) and (p.time_fly[d]+g.time_fly<MaxBlk):
                                    p.add_group(g)
                                    p.groups[d][-1].set_type(1)
                                    flag_1 = True
                                    break

                        if (not p.duty[d]):
                            if g.t_d - p.t_a_d[d - 1] >= MinRest:
                                p.add_group(g)
                                p.groups[d][-1].set_type(1)
                                flag_1 = True
                                break
                        # p.add_group(g)
                        # p.groups[d][-1].set_type(1)
                        # flag_1 = True
                        # break
                if not flag_3:
```

```
                    for p in self.pilots[2]:
                        if p.p_a_d[d - 1] == g.p_d and g.t_d - p.t_a_d[d - 1] >= LowerLimit_Combine:
                            if p.p_a_d[d] == g.p_d and p.t_d_d[d] - g.t_a >= LowerLimit_Combine:
                                if g.t_d - p.t_a_d[d - 1] >= MinRest and (p.t_d_a[d] - g.t_d<=MaxDP) and (p.time_fly[d]+g.time_fly<MaxBlk):
                                    p.add_group_first(g)
                                    p.groups[d][0].set_type(3)
                                    flag_3 = True
                                    break
                            if p.p_d_d[d] == g.p_a and g.t_d - p.t_a_d[d] >= LowerLimit_Combine:
                                if (g.t_a - p.t_d_d[d] <= MaxDP) and (p.time_fly[d]+g.time_fly<MaxBlk):
                                    p.add_group(g)
                                    p.groups[d][-1].set_type(3)
                                    flag_3 = True
                                    break

                        if (not p.duty[d]):
                            if g.t_d - p.t_a_d[d - 1] >= MinRest:
                                p.add_group(g)
                                p.groups[d][-1].set_type(3)
                                flag_3 = True
                                break
                            # p.add_group(g)
                            # p.groups[d][-1].set_type(3)
                            # flag_3 = True
                            # break
                ### 选择前一天的航班乘机过来 ###
                if not (flag_1 and flag_3):
                    for f in self.map_t2f[d - 1]:
                        if f.p_d in JIDI and g.p_d == f.p_a and g.t_d - f.t_a >= LowerLimit_Combine:
                            if g.t_d - f.t_a < MinRest:
                                continue
                            if not flag_1:
                                for p in self.pilots[1]:
                                    if p.p_a_d[max(d - 2, 0)] == f.p_d and f.t_d - p.t_a_d[max(d - 2, 0)] >= LowerLimit_Combine and (not p.duty[d - 1]) and (
```

```
                                    not p.duty[d]):
                                        p.add_group(Group_Flight(f))
                                        p.groups[d - 1][-1].set_type(2)
                                        self.chengji = self.chengji + 1
                                        p.add_group(g)
                                        p.groups[d][-1].set_type(1)
                                        flag_1 = True
                                        break
                            if not flag_3:
                                for p in self.pilots[3]:
                                    if p.p_a_d[max(d - 2, 0)] == f.p_d and f.t_d - p.t_a_d[
                                        max(d - 2, 0)] >= LowerLimit_Combine and (not p.duty[d
- 1]) and (
                                        not p.duty[d]):
                                        p.add_group(Group_Flight(f))
                                        p.groups[d - 1][-1].set_type(2)
                                        self.chengji = self.chengji + 1
                                        p.add_group(g)
                                        p.groups[d][-1].set_type(3)
                                        flag_3 = True
                                        break
                            if not flag_1:
                                for p in self.pilots[2]:
                                    if p.p_a_d[max(d - 2, 0)] == f.p_d and f.t_d - p.t_a_d[
                                        max(d - 2, 0)] >= LowerLimit_Combine and (not p.duty[d
- 1]) and (
                                        not p.duty[d]):
                                        p.add_group(Group_Flight(f))
                                        p.groups[d - 1][-1].set_type(2)
                                        self.chengji = self.chengji + 1
                                        p.add_group(g)
                                        p.groups[d][-1].set_type(1)
                                        flag_1 = True
                                        break
                            if not flag_3:
                                for p in self.pilots[2]:
                                    if p.p_a_d[max(d - 2, 0)] == f.p_d and f.t_d - p.t_a_d[
                                        max(d - 2, 0)] >= LowerLimit_Combine and (not p.duty[d
- 1]) and (
                                        not p.duty[d]):
```

```
                                    p.add_group(Group_Flight(f))
                                    p.groups[d - 1][-1].set_type(2)
                                    self.chengji = self.chengji + 1
                                    p.add_group(g)
                                    p.groups[d][-1].set_type(3)
                                    flag_3 = True
                                    break
                ### 在同一天找前面后面各一驾航班 乘两次机 ###
                temp_g = copy.deepcopy(g)
                if not (flag_1 and flag_3):
                    flag_qian = False
                    flag_hou = False
                    for f in reversed(self.map_t2f[d]):
                        if f.p_d in JIDI and f.p_a == g.p_d and g.t_d - f.t_a >= LowerLimit_Combine:
                            temp_g.add_flight_first(f)
                            if temp_g.time_fly > MaxBlk:
                                flag_qian = False
                                break
                            if temp_g.t_a - temp_g.t_d > MaxDP:
                                flag_qian = False
                                break
                            flag_qian = True
                            break
                    if flag_qian:
                        for f in self.map_t2f[d]:
                            if f.p_a in JIDI and f.p_d == g.p_a and f.t_d - g.t_a >= LowerLimit_Combine:
                                temp_g.add_flight(f)
                                if temp_g.time_fly > MaxBlk:
                                    flag_hou = False
                                    break
                                if temp_g.t_a - temp_g.t_d > MaxDP:
                                    flag_qian = False
                                    break
                                flag_hou = True

                                break
                    if flag_qian and flag_hou:
                        # flag_1 = False
```

```
                        # flag_3 = False
                        # print("99999999999999999999999999999999999999999999999999999999999999999999999999999999999999999")
                    for p in self.pilots[1]:
                        if temp_g.t_d - p.t_a_d[d - 1] < MinRest:
                            continue
                        if p.p_a_d[d - 1] == temp_g.p_d and temp_g.t_d - p.t_a_d[d - 1] >= LowerLimit_Combine and (
                        not p.duty[d]):
                            p.add_group(Group_Flight(temp_g.flights[0]))
                            p.groups[d][-1].set_type(2)
                            p.add_group(Group_Flight(temp_g.flights[1]))
                            p.groups[d][-1].set_type(1)
                            p.add_group(Group_Flight(temp_g.flights[2]))
                            p.groups[d][-1].set_type(2)
                            self.chengji = self.chengji + 2
                            flag_1 = True
                            break
                    for p in self.pilots[3]:
                        if p.p_a_d[d - 1] == temp_g.p_d and temp_g.t_d - p.t_a_d[d - 1] >= LowerLimit_Combine and (
                        not p.duty[d]):
                            p.add_group(Group_Flight(temp_g.flights[0]))
                            p.groups[d][-1].set_type(2)
                            p.add_group(Group_Flight(temp_g.flights[1]))
                            p.groups[d][-1].set_type(3)
                            p.add_group(Group_Flight(temp_g.flights[2]))
                            p.groups[d][-1].set_type(2)
                            self.chengji = self.chengji + 2
                            flag_3 = True
                            break
                    if not flag_1:
                        for p in self.pilots[2]:
                            if p.p_a_d[d - 1] == temp_g.p_d and temp_g.t_d - p.t_a_d[
                                d - 1] >= LowerLimit_Combine and (not p.duty[d]):
                                p.add_group(Group_Flight(temp_g.flights[0]))
                                p.groups[d][-1].set_type(2)
                                p.add_group(Group_Flight(temp_g.flights[1]))
                                p.groups[d][-1].set_type(1)
                                p.add_group(Group_Flight(temp_g.flights[2]))
```

```
                                    p.groups[d][-1].set_type(2)
                                    self.chengji = self.chengji + 2
                                    flag_1 = True
                                    break
                            if not flag_3:
                                for p in self.pilots[2]:
                                    if p.p_a_d[d - 1] == temp_g.p_d and temp_g.t_d - p.t_a_d[
                                    d - 1] >= LowerLimit_Combine and (not p.duty[d]):
                                    p.add_group(Group_Flight(temp_g.flights[0]))
                                    p.groups[d][-1].set_type(2)
                                    p.add_group(Group_Flight(temp_g.flights[1]))
                                    p.groups[d][-1].set_type(3)
                                    p.add_group(Group_Flight(temp_g.flights[2]))
                                    p.groups[d][-1].set_type(2)
                                    self.chengji = self.chengji + 2
                                    flag_3 = True
                                    break

                    if not (flag_1 and flag_3):
                        unallocated_g.append(g)
                        num_un = num_un + 1
                        for f in g.flights:
                            table_un = table_un.append({'FltNum': f.id, 'DptrT': f.t_d, 'ArrvT': f.t_a, "DrtrD": f.d_d},
                                                       ignore_index=True)

                for p in self.pilots[1]:
                    if p.p_a_d[d - 1] is not p.jidi and (not p.duty[d]):
                        for f in self.map_t2f[d]:
                            if f.p_d == p.p_a_d[d - 1] and f.p_a == p.jidi:
                                p.add_group(Group_Flight(f))
                                p.groups[d][-1].set_type(2)
                                self.chengji = self.chengji + 1
                                print("1111111111111111111111111111111111111111111")
                                break
        self.print_result_p3()

        table_un.to_csv("UncoveredFlights_p3_B.csv", index=False)
        print(num_un)
        print(f"{self.chengji} 乗機次数")
```

```
        print("第一天 duty 人数:", sum([p.duty[1] for p in self.pilots[1] + self.pilots[2] + self.pilots[3]]))
        list_unall=table_un["FltNum"].to_list()
        total_fly=0
        for d in range(1,NUM_D+1):
            for f in self.map_t2f[d]:
                if f.id not in list_unall:
                    total_fly=total_fly+(f.t_a-f.t_d)*2
        print("机组总体利用率: ", total_fly/self.total_zhiqin)


if __name__ == "__main__":
    start = time.time()
    # long running
    # do something other

    g = Greedy()
    g.tanlan_p2_B()
    end = time.time()
    print(end - start)
```