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	**21102480019**
队员姓名	**1.李昱硕**
	2.陈怡洲
	3.刘墨林

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 基于混合整数线性规划模型的航空公司机组排班问题优化求解

摘　　要:

针对航空公司机组优化排班问题，本文建立了混合整数线性规划模型对问题进行描述。该模型一体化地考虑了机组人员主次要资格、机组人员基地、乘机任务、航段时间、航段地点、机组资格配置、相邻航段连接时间、执勤相关时间约束、任务环相关约束情况、连续执勤天数等约束，综合优化具有优先级顺序的多个目标。

针对题目中给出的三个复杂化程度递增的子问题，本文以子问题 1 建立的 0-1 整数线性规划模型为基础，层层递进地添加模型决策变量与模型约束，并完善目标函数，最终建立了**完整的机组优化排班问题的混合整数线性规划模型**。

本文使用 Gurobi 求解器对该模型进行求解，以验证其完善性和准确性。针对题目给出的大规模数据集，设计了基于贪婪与深度优先搜索的近似求解算法进行有效求解。

针对子问题 1 的数据集 A，求解器在 1 小时内求得的 MinGAP 为 0.5%的可行解显示，有 206 个航段满足机组配置，0 个航段不满足机组配置，机组人员总体乘机次数为 18 次，总体替补资格使用次数为 3 次。近似算法在 1 秒内求得的结果显示，有 206 个航段满足机组配置，0 个航段不满足机组配置，机组人员总体乘机次数为 8 次，总体替补资格使用次数为 0 次。

针对子问题 2 的数据集 A,求解器在 3.5 小时内求得的 MinGAP 为 3.71%的可行解显示，有 199 个航段满足机组配置，7 个航段不满足机组配置，机组人员总体乘机次数为 44 次，总体替补资格使用次数为 24 次。近似算法在 1 秒内求得的结果显示，有 197 个航段满足机组配置，9 个航段不满足机组配置，机组人员总体乘机次数为 174 次，总体替补资格使用次数为 0 次。

针对子问题 3 的数据集 A，求解器在 105 分钟内求得的 MinGAP 为 1434.9%的可行解显示，有 14 个航段满足机组配置，192 个航段不满足机组配置，机组人员总体乘机次数为 2 次，总体替补资格使用次数为 4 次。近似算法在 1 秒内求得的结果显示，有 192 个航段满足机组配置，14 个航段不满足机组配置，机组人员总体乘机次数为 104 次，总体替补资格使用次数为 0 次。

对比混合整数线性规划模型的精确求解结果和基于贪婪与深度优先搜索的近似算法的求解结果可得，混合整数线性规划模型能恰当地描述复杂约束条件下机组人员优化排班问题，而采用基于贪婪与深度优先搜索的近似求解算法可快速求解该模型，二者的组合是一种行之有效的机组排班问题解决方案。

关键词：机组排班，混合整数线性规划，精确求解，贪婪算法

目录

一、 问题描述与分析 5
1.1. 背景与问题描述 5
1.1.1. 研究背景 5
1.1.2. 问题重述 5
1.1.2.1. 子问题 1 6
1.1.2.2. 子问题 2 6
1.1.2.3. 子问题 3 7
1.2. 问题分析 7
1.2.1. 文献综述 7
1.2.2. 求解思路概述 7
二、 基本假设与符号定义 8
2.1. 基本假设 8
2.2. 符号定义 8
三、 混合整数线性模型构建 10
3.1. 子问题 1 10
3.1.1. 决策变量 10
3.1.2. 目标函数 10
3.1.3. 问题约束 10
3.2. 子问题 2 13
3.2.1. 决策变量 13
3.2.2. 目标函数 13
3.2.3. 问题约束 14
3.3. 子问题 3 14
3.3.1. 决策变量 14
3.3.2. 目标函数 15
3.3.3. 问题约束 15
3.4. 模型分析 17
四、 问题求解 19
4.1. 数学模型验证与求解器求解 19
4.1.1. 求解准备 19
4.1.2. 求解结果 20
4.1.2.1. 子问题 1 求解结果 20
4.1.2.2. 子问题 2 求解结果 20
4.1.2.3. 子问题 3 求解结果 20
4.1.3. 结果分析 21
4.2. 基于贪婪的近似求解 21
4.2.1. 求解算法概述 21
4.2.1.1. 子问题 1 21
4.2.1.2. 子问题 2 23
4.2.1.3. 子问题 3 25
4.2.2. 求解结果与分析 25

4.2.2.1. 子问题 1 求解结果 25
4.2.2.2. 子问题 2 求解结果 26
4.2.2.3. 子问题 3 求解结果 26
4.2.2.4. 结果分析 26
五、 结论与展望 27
六、 参考文献 27
七、 附录 28

图目录

图 1 执勤示例 5
图 2 任务环示例 6
图 3 排班周期示例 6
图 4 文章整体思路架构 7
图 5 虚拟航班假设示意图 8
图 6 子问题 1 近似求解算法流程图 22
图 7 完整航班链生成算法流程示意图 23
图 8 完整执勤链生成算法示意图 24
图 9 子问题 3 近似求解算法流程图 24
图 10 排班周期内执勤情况生成算法流程图 25

表目录

表 1 符号定义表 8
表 2 模型决策变量规模统计（基于数据集 A） 18
表 3 模型约束规模统计（基于数据集 A） 18
表 4 子问题 1 数学模型求解结果指标 20
表 5 子问题 2 数学模型求解结果指标 20
表 6 子问题 3 数学模型求解结果指标 20
表 7 子问题 1 近似求解结果指标 26
表 8 子问题 2 近似求解结果指标 26
表 9 子问题 3 近似求解结果指标 26

一、问题描述与分析

1.1. 背景与问题描述

1.1.1. 研究背景

随着国家之间的经济联系日益紧密，城市之间的人员往来日益频繁，航空公司的总体航班执行量正快速提升。在航空市场的竞争日益激烈的今天，航空公司除了积极拓展业务之外，也十分重视航班运营成本的优化。而以机组排班问题为代表的航空运营优化问题，由于其复杂的问题约束、对航空公司运营成本的巨大影响，正日益成为航空公司运营优化中的关键问题和难点问题。

航空公司的机组排班问题是指在满足国家法律规定、国际公约、政府行政条例、民航局规定和公司规章制度的前提下，为尽可能多的航班安排适当的机组人员并使航班组成连续的任务，并且实现运营成本最低、任务安排最公平等多项目标的最优化。因此机组排班问题的研究，具有非常重要的实际意义。

1.1.2. 问题重述

在机组排班问题的研究中，有一些基本且关键的概念需要熟悉了解。

飞行任务：当机组人员以工作身份（正机长或副机长）执行的航班任务。

乘机任务：当机组人员搭乘航班时称其执行的是乘机任务。

机组资格配置：航班上不同岗位（正机长和副机长）的到岗人数的最低要求称为机组资格配置。

执勤：如图 1 所示，一次执勤由一系列的航段和之间的连接时间构成，同一次执勤内的每一趟航班都必须满足在同一天内出发的要求（到达时间无此要求）。航段间的连接时间属于执勤时间但不属于飞行时间。所以执勤时间由执勤的最后一趟航班的到达时间减去执勤的第一趟航班的出发时间。

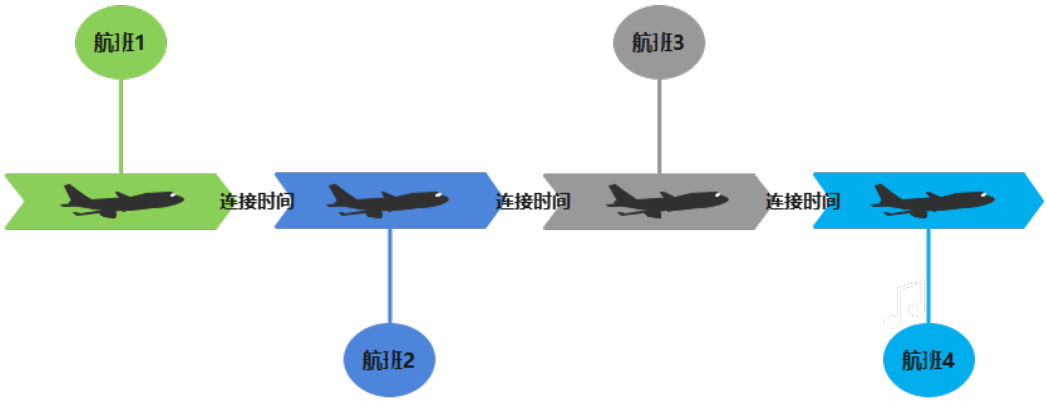

图 1 执勤示例

任务环：如图 2 所示，任务环由一系列的执勤和之间的休息时间组成。任务环一定是机组人员从自己的基地出发然后最终回到基地。

排班周期：如图 3 所示，一系列的任务环和之间的休假时间便组成了一个完整的排班周期。

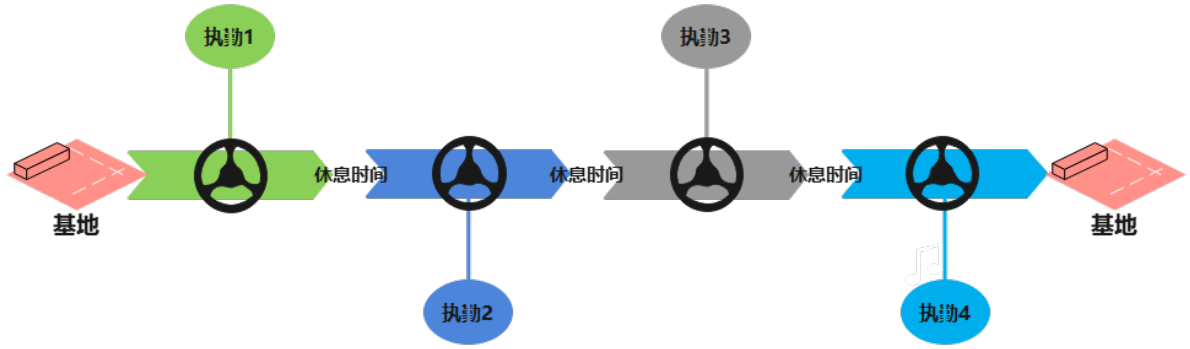

图 2 任务环示例

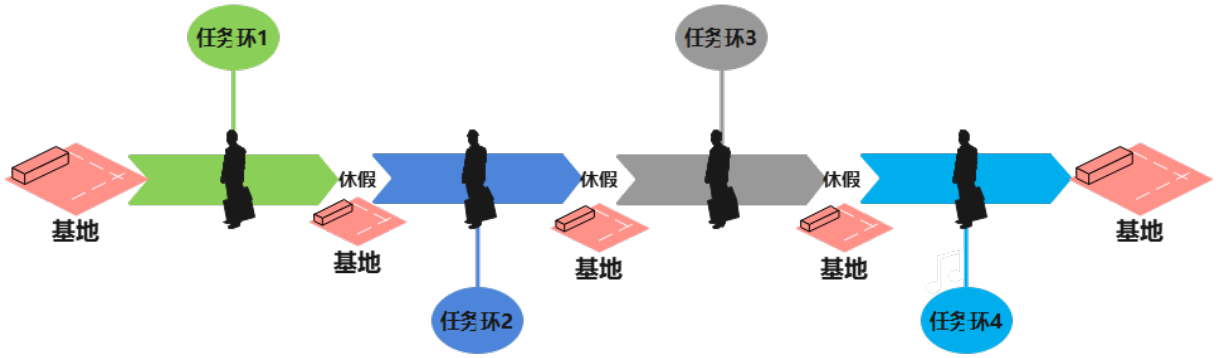

图 3 排班周期示例

本文研究的是复杂约束下的多目标机组优化排班问题，由约束数量和目标数量逐渐增加的三个子问题构成。

1.1.2.1. 子问题 1

该问题是整个研究的基础问题，它旨在帮助航空公司按照如下重要性从高到低的排序满足：

① 满足机组资格配置的航班尽可能多；

④ 所有机组人员的总乘机次数尽可能少；

⑦ 使用替补资格的人次尽可能少。

实现上述目标的前提是收到如下约束：

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一定是上一航段的到达机场；
3. 每个机场人员相邻两个航段之间的连接时间不小于一个下限；

1.1.2.2. 子问题 2

在子问题 1 的基础上，帮助航空公司额外满足：

② 机组人员的总体执勤成本最低；

⑤ 机组人员之间的执勤时长尽可能平衡。

需要额外满足如下约束：

1. 每个机组人员每天至多执行一个执勤；
2. 每次执勤的飞行时间不能超过上限；
3. 每次执勤的时长不能超过上限；
4. 每个机组人员下一执勤的起飞机场一定是上一执勤的到达机场；

5. 每个机组人员相邻两次执勤的休息时间不低于下限。

1.1.2.3. 子问题 3

在子问题 1、子问题 2 的基础上，帮助航空公司额外满足：

③ 机组人员的任务环成本总和最低；

⑥ 机组人员之间的任务环时长尽可能平衡。

上述问题中的约束所对应的约束值，在本文第二部分会给出准确定义。

1.2. 问题分析

1.2.1. 文献综述

现有文献对机组排班问题的求解策略主要分为两种。

大多数已有文献对于机组排班问题的求解均采用两阶段的求解方法，即将机组排班问题分解为任务环生成与人员分配两个子问题求解。Tahir et al.,2021 使用列生成算法并结合机器学习对组排班问题进行分阶段求解。Kornilakis et al., 2002，Souai et al., 2009 等人将机组成员排班问题分解为两个子问题，并对第二个子问题使用遗传算法进行求解。Ahmed et al.,2021 提到将原问题分解为两个子问题求解，在求解过程中缩小了解空间的搜索规模，并最终导致产生的解无法达到全局最优，只能达到局部最优。

但同时也有少数文献将机组排班问题作为一个整体考虑。Saeed et al.,2021 对机组排班和任务环生成问题进行了一体化建模，但其建模并未考虑题目中提到的机组配置约束及人员替补的约束条件，而且其仅考虑了执勤产生的时间约束，未考虑题目中提到的任务环、连续执勤时间、休假等情况，且对于建立的数学规划模型，其并验证模型的准确性和合理性。Özener et al., 2017 将机组排班的两阶段问题作为一个整体考虑，且针对大规模的机组排班问题设计了启发式近似求解算法，但文章并未建立完整的数学规划模型，也并未考虑题目中提到的机组成员资格、机组配置等约束。

结合上述文献分析可知，目前还缺乏一个综合考虑机组人员配置、机组人员资质、执勤情况、任务环情况、机组人员分配、连续飞行等约束，综合考虑满足任务配置航班数、任务环时长平衡、执勤时长平衡、总体乘机次数、执勤与任务环成本、替补资格使用次数等各个目标的，将机组排班问题整体考虑的混合整数线性规划模型，并针对实际数据进行模型正确性与准确性的验证。

1.2.2. 求解思路概述

本文的整体思路架构如图 4 所示。

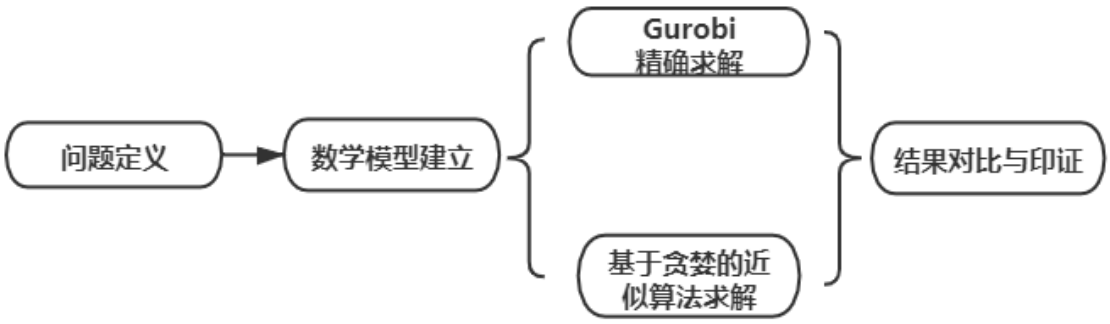

图 4 文章整体思路架构

结合文献综述和其他相关资料，混合整数线性规划模型的建立可以得出问题的全局最优解，该全局最优解可以作为近似求解算法的参考下界。

本文旨在建立一个将机组排班问题作为一个整体考虑的混合整数线性规划模型，在模

型中综合考虑题目中提到的各个方面约束，并考虑有优先级顺序的不同目标。同时使用Gurobi求解器对建立出的数学模型进行正确性进行验证，并得到问题的全局最优解。同时，为了更加高效地求解问题和解决过大问题规模问题，本文设计了基于贪婪地近似求解算法，该算法有效降低了求解的时间和空间复杂度。而小规模数据的全局最优解可以辅助验证近似算法的有效性。

文章后续主要分为四章。第二部分介绍对于问题的基本假设，并逐个解释后续模型所要用到的符号定义。第三部分详细地描述了针对题目中每个子问题所建立的数学规划模型，并对每个子问题数学模型的目标函数、决策变量、约束条件等进行解释。第四部分介绍对数学模型的求解，以及设计的近似求解算法流程，并对两种求解方式的结果进行分析对比。第五部分总结文章贡献，对后续的研究进行展望。

二、基本假设与符号定义

2.1. 基本假设

1. 航班只有满足了一个正机长、一个副机长的资格配置才能起飞；
2. 机组人员在资质符合要求的前提下可以任意组合；
3. 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；
4. 机组人员可以搭乘航班摆渡，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；
5. 虚拟航班假设：如图 5 所示，假设机组人员驾驶或搭乘的第一趟航班之前存在一趟虚拟起始航班，驾驶或搭乘的最后一趟航班之后存在一趟虚拟结束航班。且虚拟起始航班和虚拟结束航班都视为由机组人员的基地飞往基地的“不存在”的航班。该假设的严格定义和构建目的在符号定义和模型分析部分会有详细说明。

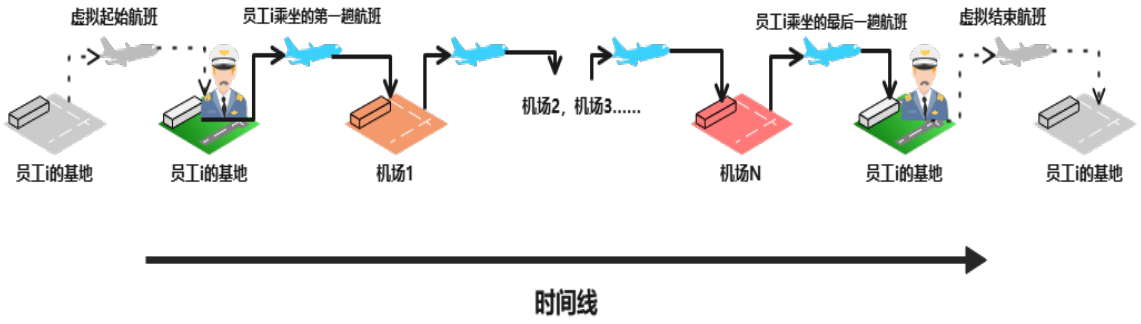

图 5 虚拟航班假设示意图

2.2. 符号定义

文章中使用的参数符号定义如下表所示：

表 1 符号定义表

符号	释义
I	机组人员集合，$i = 1, 2, \dots, \lvert I \rvert$
F	航班集合，$j = 1, 2, \dots, \lvert F \rvert$
j_{source}	虚拟起始航班，理解为从某一机组人员i的基地B_i到基地B_i
j_{sink}	虚拟结束航班，理解为从某一机组人员i的基地B_i到基地B_i

F'	实际航班与虚拟起始航班j_{source}、虚拟结束航班j_{sink}的集合
F_1	实际航班与虚拟起始航班j_{source}的集合
F_2	实际航班与虚拟结束航班j_{sink}的集合
K	排班周期（以天数为单位），$k = 1, 2, \dots, \lvert K \rvert$
K'	除去最后一天的排班周期（以天数为单位），$k = 1, 2, \dots, \lvert K-1 \rvert$
K''	出去最后四天的排班周期（以天数为单位），$k = 1, 2, \dots, \lvert K-4 \rvert$
A	机场集合，$m = 1, 2, \dots, \lvert A \rvert$
B_i	机组人员i的基地
DpT_j	航班j的出发时间（为绝对时间,单位为分钟）
ArT_j	航班j的到达时间（为绝对时间,单位为分钟）,对于虚拟起始航班j_{source}，$ArT_{j_{source}} = -MinCT$,对于虚拟结束航班$j_{sink}$，$DpT_{j_{sink}} = \max ArT_j (j \in F) + MinCT$
F_k	出发时间在第k天内的航班集合，$F_k = \{j_k, 满足\ (k-1) \cdot 1440 \le DpT_{j_k} \le k \cdot 1440\}$
$F_{k'}$	出发时间在第k天内且在航班j_k $(j_k \in F_k)$之后起飞的航班集合，$F_{k'} = \{j_{k'} \in F_k，\ 满足\ DpT_{j'_k} > ArT_{j_k}\ \forall\ j_k \in F_k\}$
$F_{\to B_i}$	到达机组人员i的基地的航班集合
$F_{B_i \to}$	从机组人员i的基地出发的航班集合
R	机组人员在航班上的角色集合，$r = \{r_1, r_2, r_3\}$ 其中r_1表示员工担任主机长，r_2表示员工担任副机长，r_3表示员工搭乘航班
ε_i^1	员工i担任主机长的资质情况，能够担任为 1，不能担任为 0
ε_i^2	员工i担任副机长的资质情况，能够担任为 1，不能担任为 0
$f_{m,n}^j$	航班j的出发点与到达地情况，航班j从机场m出发且到达机场n，则为 1，否则为 0；虚拟起始航班j_{source}满足$f_{B_i,B_i}^{j_{source}} = 1$，虚拟终止航班$j_{sink}$满足$f_{B_i,B_i}^{j_{sink}} = 1$。
L	一个大数
$MinCT$	每个机组人员相邻两个航段之间的连接时间的最小值
$MaxBlk$	每个机组人员每次执勤的飞行时间的最大值
$MaxDP$	每个机组人员每次执勤的时长的最大值
$MinRest$	每个机组人员相邻两个执勤之间的休息时间的最小值
$MaxTAFB$	每个机组人员每个任务环的总时长的最大值
$MinVacDay$	每个机组人员相邻两个任务环的休息天数的最小值
$MaxSuccOn$	每个机组人员连续执勤天数的最大值
$DutyCostPerHr$	每单位小时执勤成本
$ParingCostPerHr$	每单位小时任务环成本
$MaxSuccOn$	连续执勤天数的最大值

三、混合整数线性模型构建

3.1. 子问题 1

3.1.1. 决策变量

在子问题 1 中，我们定义了$X_{i,j,r_1}, X_{i,j,r_2}, X_{i,j,r_3}, X_{i,j}$和$Y_{i,j,j'}$五个决策变量。其中：

$$X_{i,j,r_1} = \begin{cases} 1, & 机组人员i以主机长的身份飞行航班j \\ x, & 其他情况 \end{cases}$$

$$X_{i,j,r_2} = \begin{cases} 1, & 机组人员i以副机长的身份飞行航班j \\ 0, & 其他情况 \end{cases}$$

$$X_{i,j,r_3} = \begin{cases} 1, & 机组人员i搭乘航班j \\ x, & 其他情况 \end{cases}$$

$$X_{i,j} = X_{i,j,r_1} + X_{i,j,r_2} + X_{i,j,r_3} \quad 机组人员i在航班j上$$

$$Y_{i,j,j'} = \begin{cases} 1, & 对机组人员i，航班j'是他乘坐航班j后的下一个航班 \\ 0, & 其他情况 \end{cases}$$

3.1.2. 目标函数

满足机组配置的航班的数目表示为：$\sum_{j\in F}\sum_{i\in I} X_{i,j,r_1}\cdot \varepsilon_i^1$。

总体乘机次数表示为：$\sum_{j\in F}\sum_{i\in I} X_{i,j,r_3}$。

使用替补资格的次数表示为：$\sum_{j\in F}\sum_{i\in I} X_{i,j,r_2}\cdot \varepsilon_i^2\cdot \varepsilon_i^1$。

所以子问题 1 的目标函数为：

$$\mathrm{Obj}_1 = \min\left(-\omega_1\cdot\sum_{j\in F}\sum_{i\in I} X_{i,j,r_1}\cdot\varepsilon_i^1 + \omega_4\cdot\sum_{j\in F}\sum_{i\in I} X_{i,j,r_3} + \omega_7\cdot\sum_{j\in F}\sum_{i\in I} X_{i,j,r_2}\cdot\varepsilon_i^2\cdot\varepsilon_i^1\right) \quad (1)$$

其中$\omega_1, \omega_4, \omega_7$分别表示目标 1、4 和 7 的权重系数。

3.1.3. 问题约束

子问题 1 需要如下（2）-（25）所示的约束集：

$$X_{i,j,r_1} + X_{i,j,r_2} + X_{i,j,r_3} \le 1, \quad \forall i\in I, \forall j\in F' \quad (2)$$

$$X_{i,j,r_1} \le \varepsilon_i^1, \quad \forall i\in I, \forall j\in F' \quad (3)$$

$$X_{i,j,r_2} \le \varepsilon_i^2, \quad \forall i\in I, \forall j\in F' \quad (4)$$

$$\sum_{i\in I} X_{i,j,r_1}\varepsilon_i^1 \le 1, \quad \forall j\in F' \quad (5)$$

$$\sum_{i\in I} X_{i,j,r_2}\varepsilon_i^2 \le 1, \qquad \forall j \in F' \tag{6}$$

$$\sum_{i\in I} X_{i,j,r_1}\varepsilon_i^1 = \sum_{i\in I} X_{i,j,r_2}\varepsilon_i^2, \qquad \forall j \in F' \tag{7}$$

$$X_{i,j} \le \sum_{i'\in I} X_{i',j,r_1}\varepsilon_i^1, \qquad \forall i \in I, \forall j \in F' \tag{8}$$

$$\sum_{j'\in F_2 \text{且} j' \text{在} j \text{后}} Y_{i,j,j'} \le 1, \qquad \forall i \in I, \forall j \in F_1 \tag{9}$$

$$\sum_{j\in F_1 \text{且} j \text{在} j' \text{前}} Y_{i,j,j'} \le 1, \qquad \forall i \in I, \forall j' \in F_2 \tag{10}$$

$$\sum_{n\in A} X_{i,j'} f_{k,n}^{j'} - \sum_{m\in A} X_{i,j} f_{m,k}^{j} + L\left(3 - X_{i,j} - X_{i,j'} - Y_{i,j,j'}\right) \ge 0,$$
$$\forall i \in I, \forall j \in F_1, j' \in F_2 \text{且} j' \text{在} j \text{后}, \forall k \in A \tag{11}$$

$$\sum_{n\in A} X_{i,j'} \cdot f_{k,n}^{j'} - \sum_{m\in A} X_{i,j} \cdot f_{m,k}^{j} - L\left(3 - X_{i,j} - X_{i,j'} - Y_{i,j,j'}\right) \le 0,$$
$$\forall\, i \in I, \forall j \in F_1, \forall j' \in F_2 \text{且} j' \text{在} j \text{后}, \forall k \in A \tag{12}$$

$$\sum_{j\in F'} \sum_{n\in A} X_{i,j} \cdot f_{B_i,n}^{j} - \sum_{j\in F'} \sum_{m\in A} X_{i,j} \cdot f_{m,B_i}^{j} = 0, \qquad \forall i \in I \tag{13}$$

$$L \times \left(1 - Y_{i,j,j'}\right) + X_{i,j} + X_{i,j'} \ge 2,$$
$$\forall i \in I, \forall j \in F_1, \forall j' \in F_2 \text{且} j' \text{在} j \text{后} \tag{14}$$

$$L \times \left(1 - X_{i,j}\right) + \sum_{j'\in F_2 \text{且} j' \text{在} j \text{后}} Y_{i,j,j'} \ge 1, \qquad \forall i \in I, \forall j \in F_1 \tag{15}$$

$$-L \times \left(1 - X_{i,j}\right) + \sum_{j'\in F_2 \text{且} j' \text{在} j \text{后}} Y_{i,j,j'} \le 1, \qquad \forall i \in I, \forall j \in F_1 \tag{16}$$

$$L \times \left(1 - X_{i,j'}\right) + \sum_{j\in F_1 \text{且} j \text{在} j' \text{前}} Y_{i,j,j'} \ge 1, \qquad \forall i \in I, \forall j' \in F_2 \tag{17}$$

$$-L\times\left(1-X_{i,j'}\right)+\sum_{j\in F_1且j在j'前}Y_{i,j,j'}\le 1,\qquad \forall i\in I,\forall j'\in F_2 \tag{18}$$

$$DpT_{j'}-ArT_j+L\times\left(1-Y_{i,j,j'}\right)\ge MinCT,$$
$$\forall i\in I,\forall j\in F_1,\forall j'\in F_2且j'在j后 \tag{19}$$

$$X_{i_1,j_{source},r_1}=1 \tag{20}$$
$$X_{i_2,j_{source},r_2}=1 \tag{21}$$

$$X_{i,j_{source},r_3}=1,\qquad \forall i\in I,i\ne i_1且i\ne i_2 \tag{22}$$

$$X_{i_1,j_{sink},r_1}=1 \tag{23}$$
$$X_{i_2,j_{sink},r_2}=1 \tag{24}$$

$$X_{i,j_{sink},r_3}=1,\qquad \forall i\in I,i\ne i_1且i\ne i_2 \tag{25}$$

下面对上述约束逐个进行解释：

约束(2)确保每一名机组人员i至多以一种身份出现在每一趟航班j上。

约束(3)和约束(4)中，ε_i^1是机组人员i的主机长资质参数，ε_i^2是机组人员i的副机长资质参数。这两式表示，对于机组人员i，只有其具备担任主机长或副机长的资质，才能够以主机长或副机长的身份乘坐在某一趟航班j上。

约束(5)、约束(6)和约束(7)属于航班配置的 C1F1 约束。具体来说，约束(5)要求，对于任意航班j，在所有机组成员中最多有 1 名机组成员以主机长的身份乘坐此航班（C1 约束）。约束(6)要求，对于任意航班j，在所有机组成员中最多有 1 名机组成员以副机长的身份乘坐此航班（F1 约束）。约束(7)要求，对于任意航班，主机长和副机长（C1F1）必须同时满足，或者同时不满足。

约束(8)为关于航班取消与否的约束。不等式右侧表示，对于航班j，所有以主机长身份乘机的人。当不等式右侧等于 0 时，则表示没有人以主机长身份乘坐航班j，则此时根据问题要求航班应该取消，对应不等式左侧，即表示不得有任何人以任何身份乘坐航班j。

约束(9)和约束(10)属于关于决策变量Y的约束。约束(9)要求对于任意机组人员i，任意航班j的，在 j 所有的后续航班中，机组人员i至多只能乘坐一个航班作为后续航班。相似地，约束(10)要求对于任意机组人员i，任意航班j'，在j'的所有前序航班中，机组人员i至多只能乘坐一个航班作为前序航班。

约束(11)和约束(12)属于关于决策变量Y中前后续航班连接地点的约束。在两个约束共同作用下，如果任意机组人员i乘坐了任一航班j，并接续地乘坐了航班j之后的航班j'，那么要求航班j的到达地点与航班j'的出发点相同。由于在参数定义中，我们以矩阵的形式定义了航班的出发到达地点参数，为了方便模型求解，故在这两个约束中，我们对两航班的换乘地k进行了循环。若两个航班的换乘地不为k，则约束的第一项、第二项各等于 0。若两个航班的换乘地为k，则约束的第一项、第二项各等于 1。

约束(13)为关于机组成员与基地之间的关系。约束(13)要求任意机组成员，乘坐航班从该成员对应的基地出发的总次数和乘坐航班到达该成员对应的基地的总次数相等。从而保证了当机组人员初始从基地出发时，最终回到的一定是基地。

约束(14)-约束(18)为关于决策变量Y和决策变量X之间的关系的约束。

约束(14)要求对于任意机组人员i，$Y_{i,j,j'}$为 1 时，该机组成员i必须乘坐了航班j和航班j'，即对应其每个航班对应的$X_{i,j}$变量为 1。

约束(15)-约束(18)等效为两个等式约束，该约束保证，对于任意机组成员i，一旦其乘坐了某航班j，则该机组成员必定乘坐了 1 个且仅乘坐了 1 个后续航班j'；相似地，对于任意机组成员i，一旦其乘坐了某航班j'，则该机组成员必定乘坐了 1 个且仅乘坐了 1 个前续航班j。

注意对于约束(15)-(18)，由于虚拟航班j_{source}和j_{sink}的存在，且所有机组成员都乘坐了这两个虚拟航班，所以，对于机组人员乘坐的第一班真实航班和最后一班真实航班，约束约束(15)-(18)亦能成立。

约束(19)为关于题目中要求的最小连接时间（$MinCT$）约束。该约束要求对于任意机组成员i，其乘坐的接续的两个航班j、j'应满足前序航班到达时间与后序航班出发时间的间隔要求

约束(20)-约束(25)定义了虚拟航班的决策变量。对于航班j_{source}和j_{sink}，均直接指定有主机长资质的机组成员i_1和有副机长资质的机组成员i_2担任机长和副机长，以保证该航班不被取消。其他机组成员均以乘机（deadhead）身份乘坐两个虚拟航班。

注意，在子问题 1 以及后续所有子问题的目标函数中，并不统计虚拟航班的乘机身份、虚拟航班的乘机时间、虚拟航班导致的执勤时间等决策变量。引入虚拟航班主要用于保证决策变量Y及其对应约束在建模中的完整性，并保证能以线性模型的形式，表达所有机组成员均从其对应的基地出发、且最后回到其对应的基地这一条件。

3.2. 子问题 2

3.2.1. 决策变量

在子问题 1 的基础上，增添三个新的决策变量：

$$DT_{i,k}: \textit{机组人员}i\textit{在第}k\textit{天的执勤时间}$$

$$DTmax: \textit{所有机组人员在整个排产周期内的执勤时间总和的最大值}$$

$$DTmin: \textit{所有机组人员在整个排产周期内的执勤内的执勤时间总和的最小值}$$

3.2.2. 目标函数

机组人员的总体执勤成本表示为：$\sum_{i\in I}\sum_{k\in K} DT_{i,k}\cdot\frac{DutyCostPerHr}{60}$。

通过极差的方式来反映不同机组人员的执勤时长的平衡状况，当极差很小时，可以认为机组人员之间的执勤时长相差不大，较为平衡，表示为：$DTmax - DTmin$。

所以在Obj_1的基础上，子问题 2 的目标函数为：

$$\mathrm{Obj}_2 = \min\left(-\omega_1 \cdot \sum_{j\in F}\sum_{i\in I} X_{i,j,r_1} \cdot \varepsilon_i^1 + \omega_2 \cdot \sum_{i\in I}\sum_{k\in K} DT_{i,k} \cdot \frac{DutyCosrPerHr}{60} + \omega_4 \cdot \sum_{j\in F}\sum_{i\in I} X_{i,j,r_3} + \omega_5 \cdot (DTmax - DTmin) + \omega_7 \cdot \sum_{j\in F}\sum_{i\in I} X_{i,j,r_2} \cdot \varepsilon_i^2 \cdot \varepsilon_i^1 \right) \quad (26)$$

类似的，ω_2代表目标 2 的权重系数，ω_5代表目标 5 的权重系数。

3.2.3. 问题约束

在子问题 1 的基础上，新增加如下（27）-（32）的约束集：

$$\sum_{j_k\in F_k} X_{i,j_k,r} \cdot \left(ArT_{j_k} - DpT_{j_k}\right) \le MaxBlk, \qquad \forall i \in I, \forall k \in K \quad (27)$$

$$ArT_{j_k'} - DpT_{j_k} - L \times (2 - X_{i,j_k} - X_{i,j_k'}) \le MaxDP, \quad \forall i \in I, \forall k \in K, \forall j_k \in F_k, \forall j_k' \in F_k' \quad (28)$$

$$ArT_{j_k'} - DpT_{j_k} - L \times (2 - X_{i,j_k} - X_{i,j_k'}) \le DT_{i,k}, \quad \forall i \in I, \forall k \in K, \forall j_k \in F_k, \forall j_k' \in F_k' \quad (29)$$

$$X_{i,j_{k+1}} \cdot DpT_{j_{k+1}} - X_{i,j_k} \cdot ArT_{j_k} + L \times \left(1 - Y_{i,j_k,j_{k+1}}\right) \ge MinRest, \quad \forall i \in I, \forall k \in K', \forall j_k \in F_k, \forall j_{k+1} \in F_{k+1} \quad (30)$$

$$\sum_{k\in K} DT_{i,k} \le DTmax, \qquad \forall i \in I \quad (31)$$

$$\sum_{k\in K} DT_{i,k} \ge DTmin, \qquad \forall i \in I \quad (32)$$

下面对上述约束逐个进行解释：

约束(27)是每次执勤飞行时间的时长约束。不等式左侧表示机组人员i在第k天执勤的每趟航班的飞行时长之和，不等式右侧的$MaxBlk$表示单次执勤飞行时长的上限。

约束(28)是每次执勤时间（包含飞行时间和连接时间）的时长约束。机组人员i在第k天的执勤时长总可以由出发时间在第k天内的某两趟航班j_k和j_k'以不等式左侧的表达式表示，而不等式右侧的$MaxDP$则表示单次执勤时长的上限。

约束(29)是为了约束决策变量$DT_{i,k}$，与约束(28)类似，不等式左侧表达的最大值对应的便是机组人员i在第k天的执勤时间。

约束(30)是相邻执勤之间的休息时长约束。当相邻执勤发生在相邻两天内，只需后一个执勤的第一趟航班的出发时间和前一个执勤的最后一趟航班的到达时间之差满足休息时长约束即可，此时$Y_{i,j_k,j_{k+1}} = 1$；当相邻执勤不在相邻两天发生，此时$Y_{i,j_k,j_{k+1}} = 0$，大数L约束起到作用。不等式右侧的$MinRest$表示相邻执勤之间的最短休息时长。

约束(31)和(32)则是为了找到机组人员的个人总执勤时长的最大值和最小值。

3.3. 子问题 3

3.3.1. 决策变量

在子问题 1、子问题 2 的基础上，增添五个新的决策变量：PT_i, $PTmax$, $PTmin$, $Duty_{i,k}$和Z_{i,j_1,j_2}，定义如下：

PT_i: *机组人员i在排班周期内的所有任务环时长总和*

$PTmax$: *机组人员任务环时长总和的最大值*

$PTmin$: *机组人员任务环时长总和的最小值*

$$Duty_{i,k} = \begin{cases} 1, & \text{机组人员}i\text{在排班周期的第}k\text{天执勤了} \\ 0, & \text{机组人员}i\text{在排班周期的第}k\text{天没有执勤} \end{cases}$$

$$Z_{i,j_1,j_2} = \begin{cases} 1, & j_1\text{和}j_2\text{是机组人员}i\text{的某一个任务环的起始航班任务和终止航班任务} \\ 0, & x \geq 0 \end{cases}$$

且有：$j_1 \in F_{B_i\rightarrow}, j_2 \in F_{\rightarrow B_i}$

3.3.2. 目标函数

机组人员的总体任务环成本表示为：$\sum_{i\in I} PT_i \cdot \frac{ParingCostPerHr}{60}$。

类似于机组人员之间执勤时长平衡的思路，机组人员之间的任务环平衡同样通过极差来反映。当极差很小时，说明任务环时长很接近，更为平衡。表示为：$PTmax - PTmin$。

所以在Obj_2的基础上，子问题 3 目标函数为：

$$\begin{aligned} \mathrm{Obj}_3 = \min\Bigg(& -\omega_1 \cdot \sum_{j\in F}\sum_{i\in I} X_{i,j,r_1} \cdot \varepsilon_i^1 + \omega_2 \cdot \sum_{i\in I}\sum_{k\in K} DT_{i,k} \cdot \frac{DutyCosrPerHr}{60} + \omega_3 \\ & \cdot \sum_{i\in I} PT_i \cdot \frac{ParingCostPerHr}{60} + \omega_4 \cdot \sum_{j\in F}\sum_{i\in I} X_{i,j,r_3} + \omega_5 \cdot (DTmax - DTmin) \\ & + \omega_6 \cdot (PTmax - PTmin) + \omega_7 \sum_{j\in F}\sum_{i\in I} X_{i,j,r_2} \cdot \varepsilon_i^2 \cdot \varepsilon_i^1 \Bigg) \end{aligned} \tag{33}$$

类似的，ω_3代表目标 3 的权重系数，ω_6代表目标 6 的权重系数。

3.3.3. 问题约束

在子问题 1、子问题 2 的基础上，新增加如下（34）-（49）的约束集：

$$Duty_{i,k} \leq \sum_{j_k\in F_k} X_{i,j_k}, \qquad \forall i \in I, \forall k \in K \tag{34}$$

$$Duty_{i,k} \geq X_{i,j_k}, \qquad \forall i \in I, \forall k \in K, \forall j_k \in F_k \tag{35}$$

$$Duty_{i,k} + Duty_{i,k+1} + Duty_{i,k+2} + Duty_{i,k+3} + Duty_{i,k+4} \leq MaxSuccOn, \quad \forall i \in I, \forall k \in K'' \tag{36}$$

$$X_{i,j_1} + X_{i,j_2} + L \times (1 - Z_{i,j_1,j_2}) \geq 2, \quad \forall i \in I, \forall j_1 \in F_{B_i\rightarrow}, \forall j_2 \in F_{\rightarrow B_i}\text{且}j_2\text{在}j_1\text{之后} \tag{37}$$

$$L \times \left(1 - Y_{i,j_{source},j}\right) + \sum_{j' \in F_{\rightarrow B_i} 且 j' 在 j 之后} Z_{i,j,j'} \geq 1,$$
$$\forall i \in I, \forall j \in F_{B_i \rightarrow} \tag{38}$$

$$-L \times \left(1 - Y_{i,j_{source},j}\right) + \sum_{j' \in F_{\rightarrow B_i} 且 j' 在 j 之后} Z_{i,j,j'} \leq 1,$$
$$\forall i \in I, \forall j \in F_{B_i \rightarrow} \tag{39}$$

$$L \times \left(1 - Y_{i,j,j_{sink}}\right) + \sum_{j' \in F_{B_i \rightarrow}, 且 j' 在 j 之前} Z_{i,j',j} \geq 1,$$
$$\forall i \in I, \forall j \in F_{\rightarrow B_i} \tag{40}$$

$$-L \times \left(1 - Y_{i,j,j_{sink}}\right) + \sum_{j' \in F_{B_i \rightarrow} 且 j' 在 j 之前} Z_{i,j',j} \leq 1,$$
$$\forall i \in I, \forall j \in F_{\rightarrow B_i} \tag{41}$$

$$L \times \left(2 - Z_{i,j_1,j_2} - Y_{i,j_2,j_3}\right) + \sum_{j_4 \in F_{\rightarrow B_i} 且 j_4 在 j_3 后} Z_{i,j_3,j_4} \geq 1,$$
$$\forall i \in I, \forall j_1 \in F_{B_i \rightarrow}, \forall j_2 \in F_{\rightarrow B_i} 且 j_2 在 j_1 后, \forall j_3 \in F_{B_i \rightarrow} 且 j_3 在 j_2 后 \tag{42}$$

$$-L \times \left(2 - Z_{i,j_1,j_2} - Y_{i,j_2,j_3}\right) + \sum_{j_4 \in F_{\rightarrow B_i} 且 j_4 在 j_3 后} Z_{i,j_3,j_4} \leq 1,$$
$$\forall i \in I, \forall j_1 \in F_{B_i \rightarrow}, \forall j_2 \in F_{\rightarrow B_i} 且 j_2 在 j_1 后, \forall j_3 \in F_{B_i \rightarrow} 且 j_3 在 j_2 后 \tag{43}$$

$$ArT_{j_2} - DpT_{j_1} - L \times \left(1 - Z_{i,j_1,j_2}\right) \leq MaxTAFB,$$
$$\forall i \in I, \forall j_1 \in F_{B_i \rightarrow}, \forall j_2 \in F_{\rightarrow B_i} 且 j_2 在 j_1 之后 \tag{44}$$

$$DpT_{j_1'} - ArT_{j_2} + L \times \left(3 - Z_{i,j_1,j_2} - Z_{i,j_1',j_2'} - Y_{i,j_2,j_1'}\right) \geq MinVacDay,$$
$$\forall i \in I, \forall j_1 \in F_{B_i \rightarrow}, \forall j_2 \in F_{\rightarrow B_i} 且 j_2 在 j_1 后, \forall j_1' \in F_{B_i \rightarrow} 且 j_1' 在 j_2 后, \forall j_2' \in F_{\rightarrow B_i} 且 j_2' 在 j_1' 后 \tag{45}$$

$$PT_i = \sum_{j_1 \in F_{B_i \rightarrow}} \sum_{j_2 \in F_{\rightarrow B_i} 且 j_2 在 j_1 后} \left(ArT_{j_2} - DpT_{j_1}\right) \cdot Z_{i,j_1,j_2}, \qquad \forall i \in I \tag{46}$$

$$PT_i \leq PTmax, \qquad \forall i \in I \tag{47}$$

$$PT_i \geq PTmin, \qquad \forall i \in I \tag{48}$$

下面对上述约束逐个进行解释：

约束(34)和(35)是对决策变量$Duty_{i,k}$的定义约束。如果$Duty_{i,k} = 0$，即机组人员i在第k天没有执勤，那么他不会乘坐任何一架在第k天起飞的航班；相反地，只要机组人员i乘坐了在第k天起飞的任一航班，那么$Duty_{i,k} = 1$。

约束(36)是对最长连续执勤天数的约束。不等式右侧的$MaxSuccOn$表示连续执勤天数的上限（为 4 天），所以只需要让连续 5 天的$Duty_{i,k}$之和不超过$MaxSuccOn = 4$即可。

约束(37)是对决策变量Z_{i,j_1,j_2}的定义约束。当$Z_{i,j_1,j_2} = 1$时，意味着机组人员i执行了j_1和j_2两个航班任务，也就意味着$X_{i,j_1} = X_{i,j_2} = 1$。

约束(38)和约束(39)保证每个机组人员i的初始任务环一定是其第一次从基地出发的航班为起始。当$Y_{i,j_{source},j} = 1$时，表明此时的j航班是机组人员i乘坐的第一趟航班（由虚拟起始航班j_{source}决定），那么这趟j航班一定是且唯一是某一个任务环的起始航班任务。由于该条件为等式约束，所以拆解为(38)和(39)两个约束。

约束(40)和约束(41)保证每个机组人员i的最后一个任务环的终点一定是其在整个排班周期内搭乘的最后一趟航班。与约束(38)和约束(39)的逻辑相似，当$Y_{i,j,j_{sink}} = 1$时，表明此时的j航班一定是机组人员i乘坐的最后一趟航班（由虚拟终止航班j_{sink}决定），那么这趟 j 航班一定是且唯一是某一个任务环的终止航班任务。同样的，由于该条件为等式约束，所以拆解为(40)和(41)两个约束。

约束(38)、(39)、(40)和(41)一起，保证了整个排班周期的任务环划分是从机组人员i的第一趟航班开始到最后一趟航班结束的。

约束(42)和约束(43)保证每个机组人员i的每个任务环之间是环环相接的。当$Z_{i,j_1,j_2} = 1$且$Y_{i,j_2,j_3} = 1$时，表明j_1和j_2是机组人员i的一个任务环的起始航班和终止航班并且航班j_3和j_2是他/她相邻乘坐的两趟航班，那么一定有航班j_3是下一个任务环的起始航班。这样便保证了整个排班周期内各环是依次相接的。由于该条件为等式约束，所以拆解为(42)和(43)两个约束。需要注意的是，航班j_1, j_2, j_3是按照时间先后顺序进行选择的。

约束(44)是对每个任务环的时长约束。如果j_1和j_2分别是机组人员i的某一个任务环的起始航班和终止航班，那么该任务环的时长便可由航班j_2的到达时间减去航班j_1的出发时间表示，时长不超过$MaxTAFB$（单个任务环总时长的上限）。需要注意的是，航班j_1, j_2, j_1'和j_2'是按照时间先后顺序进行选择的。

约束(45)是相邻任务环之间的休息时长约束。当$Z_{i,j_1,j_2} = Z_{i,j_1',j_2'} = Y_{i,j_2,j_1'} = 1$时，表明$j_1, j_2$和$j_1', j_2'$是前后两个相邻的任务环，故而只需让航班$j_1'$的出发时间减去航班$j_2$的到达时间不低于$MinVacDay$（相邻任务环的休息时长下限）。

约束(46)是对决策变量PT_i的定义约束。机组人员i的所有任务环时长总和由每个任务环时长求和即可得到。

约束(47)和约束(48)是对任务环时长总和的最大值和最小值的定义约束。

3.4. 模型分析

上述目标函数和约束构成的模型为典型的混合整数线性规划模型。

子问题 1 对应的模型（记为模型 1）有 5 种 0-1 型决策变量，满足子问题 1 和子问题 2 约束的模型（记为模型 2）有 5 种 0-1 型决策变量和 3 种连续型决策变量，最终同时满足 3 个子问题约束的模型（记为模型 3）有 7 种 0-1 型决策变量和 6 种连续型决策变量。在这些决策变量中，0-1 型决策变量$X_{i,j}, Y_{i,j_{,,}j'}$和Z_{i,j_1,j_2}对模型的约束构建最为重要，且$Y_{i,j,j'}$和

Z_{i,j_1,j_2}对模型的复杂度起关键性的决定作用。基于数据集 A，统计出模型中各决策变量的规模，如下表 2 所示。$Y_{i,j,j'}$和Z_{i,j_1,j_2}的数量分别达到了 135429 和 222705，远超其他决策变量。这也表明了模型规模的庞大。

表 2 模型决策变量规模统计（基于数据集 A）

变量类型	变量名称	模型 1	模型 2	模型 3
0-1 型决策变量	X_{i,j,r_1}	4368	4368	4368
	X_{i,j,r_2}	4368	4368	4368
	X_{i,j,r_3}	4368	4368	4368
	$X_{i,j}$	4368	4368	4368
	$Y_{i.j\ .j'}$	**135429**	**135429**	135429
	$Duty_{i,k}$	---	---	315
	Z_{i,j_1,j_2}	---	---	**222705**
连续型决策变量	$DT_{i,k}$	---	315	315
	$DTmax$	---	1	1
	$DTmin$	---	1	1
	PT_i	---	---	21
	$PTmax$	---	---	1
	$PTmin$	---	---	1
其他变量	$Total\_binary$	148533	148533	371553
	$Total\_continuous$	0	317	340

模型中的约束同样类型丰富、数量庞大。约束基本可分为如下几种类别：

a. 描述决策变量性质的约束，如约束(2)-(6)，约束(9)-(10)等；
b. 连接不同决策变量间关系的约束，如约束(14)-(18)，约束(34)-(42)等；
c. 满足问题中实际要求的约束，如约束(19)，约束(28)，约束(30)等；
d. 用于表示新决策变量（如$DT_{i,k}, PT_i$）的约束，如约束(29)，约束(46)等。

其中，描述变量性质和连接不同决策变量间关系的两类约束对于模型结构至关重要，只有当决策变量正确满足定义且符合相互间的逻辑关系，模型才有意义。同样基于数据集 A，我们统计出了模型中的约束规模如下表 3 所示。约束的数量达到百万级甚至千万级，可见模型的复杂程度之高。

表 3 模型约束规模统计（基于数据集 A）

	模型 1	模型 2	模型 3
约束数量	2211105	2240190	13202379

此外，关于模型中的虚拟起始航班j_{source}和虚拟结束航班j_{sink}，则是在模型构建过程中为了满足特定约束并方便约束表达而引入的，出发点为：为了解决机组人员i乘坐的相邻两趟航班j和j'的相关约束时，提出了 0-1 决策变量$Y_{i,j,j'}$。该变量可以很好的描述机组人员所乘坐的位于初始航班和结束航班之间的每趟航班的前序航班和后序航班情况，但唯独不能表示初始航班的前序航班和结束航班的后序航班，从而对“任一机组人员乘坐的航班必须环环相扣且这条序列的起始和终点必须是初始航班和结束航班”这条要求无法构建出可行的约束。故而引入虚拟起始航班j_{source}和虚拟结束航班j_{sink}并将其定义为从机组人员i的基地飞往基地的这趟实际不存在的航班，这样$Y_{i,j,j'}$便可解决上述的缺陷，从而构造出对应的

约束。

最后，需要强调的是，本章节建立的混合整数线性规划模型具有良好的泛化能力，并非只针对特定参数的特定取值而设计，其在不同的问题假设情境下只需更改模型中的部分参数即可继续成立。例如，若更改题目中所给条件，要求所有只具备副机长资质的机组人员在排班周期结束后前往机场 15 参加培训。则根据模型中参数与虚拟航班的定义，只需更增添一个从机场 15 飞往机场 15 的虚拟结束航班，并增添约束使得所有具备副机长资质的机组人员必须乘坐机场 15 的虚拟结束航班而非回到基地的虚拟结束航班。

四、问题求解

针对题目给出的两套数据集，设计了运用求解器的精确求解和基于贪婪的启发式算法近似求解两种求解方法。

分析两个待求解数据集发现，数据集 A 的规模较小，仅有 21 名待指派机组成员和 206 次待指派航班，而数据集 B 包括 465 名待指派机组成员和 13954 次待指派航班，数据集 B 相对于数据集 A 的规模有显著增加。

考虑到模型中的变量规模、使用的计算机内存及运算速度的限制，决定使用精确求解和近似求解两种方案分别求解数据集 A，只使用近似求解算法求解数据集 B。对于精确求解算法，使用目前最高效的 Gurobi 求解器进行建模求解，能够有效验证模型的正确性与可行性，同时精确求解得到的目标值。该目标值可以用于评价近似求解算法的有效性。而使用近似求解方法能够有效降低大规模问题的求解时间与存储空间消耗，更加直观高效。

基于第三章建立的三个数学模型，分别编写三个精确求解算法代码和三个近似求解算法代码。同时还编写了原始数据处理代码和最终要求结果生成代码等辅助代码。所有代码的编写均使用 Python 3.8 编程语言，在 Intel Core i7_7700HQ 16GB 内存四核八处理器配置的笔记本电脑上运行求解。模型求解代码附于附录中，计算具体结果详见附件。

4.1. 数学模型验证与求解器求解

4.1.1. 求解准备

实际求解时，需要对模型目标函数中七个目标的权重进行合理的赋值。权重赋值的基本思想为：$权重 = \frac{优先级系数}{标准化系数}$。首先，由于各目标之间的实际含义和取值范围可能存在极大的差异，例如总执勤成本和未被取消的航班数，所以需要对目标进行标准化处理。某单个目标的标准化系数可近似由该目标的取值范围表示。例如，目标总执勤成本的取值范围便是：$0 \rightarrow \sum_{机组人员}（某机组人员的最大执勤时长 \times DutyCostPerHr \times \frac{1}{60} \times 天数）$。其次，优先级系数的选择主要依据各目标的重要性程度来判断。在该问题中，重要性最高的三个目标为未被取消的航班数、机组人员的总体执勤成本和机组人员的总体任务环成本。结合实际分析，保证航班的低取消率和尽可能低的总成本是航空最为关注的优化目标，其重要性远高于其他目标。故而结合实际经验和小规模算例实验，我们将上述重要性最高的三个目标的优先级系数分别设为：$10^7, 5 \times 10^4$和10^4。重要性次之的四个目标的优先级系数分别为：$10^3, 5 \times 10^2, 10^2$和1。

这样的权重设置保证了在有限的求解时间内，Gurobi 在求解过程中的主要优化目标为优先级最高的目标，即尽可能多的航班满足机组配置。例如在模型 1 的求解过程中，发现模型求解最后 0.5%GAP 的时间消耗非常显著。若不对模型设置求解时间限制，模型运行

将近 3 小时依然无法得到 GAP 为 0.01%以下的解。求解时间为 1 小时时的最优解已经可以保证所有航班均满足机组配置，1 小时后的求解时间主要消耗在最小化替补资格使用次数和最小化乘机次数等次要目标中。

在利用 Gurobi 编写求解代码的过程中，也发现了一些提高求解效率、优化求解过程的方法。比如利用 numpy 包替代 python 自带的字典，可大幅度缩短 Gurobi 模型构建需要的时间（在实验过程中，存在从几小时缩短到几十分钟的情况）。此外，由于模型规模过于庞大且个别约束过于复杂，存在三层甚至四层的 for 循环嵌套结构，此时通过结合约束的实际含义（比如航班时间上的先后关系，前序航班的到达地必须是后序航班的出发地等），对内层 for 循环的搜索空间进行可观的缩减，从而降低模型对内存的需求，提升了求解速度。

4.1.2. 求解结果

4.1.2.1. 子问题 1 求解结果

表 4 子问题 1 数学模型求解结果指标

结果指标	指标值
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	18
替补资格使用次数	3
程序运行分钟数	60

Gurobi 求解过程中完整的控制台输出已附于附件中。

在求解时，设置 Gurobi 在求解至 1 小时时停止求解，控制台输出显示此时求得的最优解与求解器的估计下界之间的差距（MinGAP）为 0.5%。

4.1.2.2. 子问题 2 求解结果

表 5 子问题 2 数学模型求解结果指标

结果指标	指标值
不满足机组配置航班数	7
满足机组配置航班数	199
机组人员总体乘机次数	44
替补资格使用次数	24
机组总体利用率	85.13%
最小/平均/最大 一次执勤飞行时长	0/150.65/490
最小/平均/最大 一次执勤执勤时长	0/176.97/675
总体执勤成本（万元）	42.84
程序运行分钟数	210

Gurobi 求解过程中完整的控制台输出已附于附件中。

在求解时，设置 Gurobi 在求解至 3 小时时停止求解，控制台输出显示此时求得的最优解与求解器的估计下界之间的差距（MinGAP）为 3.71%。

4.1.2.3. 子问题 3 求解结果

表 6 子问题 3 数学模型求解结果指标

结果指标	指标值
不满足机组配置航班数	14

满足机组配置航班数	192
机组人员总体乘机次数	2
替补资格使用次数	4
机组总体利用率	81.77%
最小/平均/最大 一次执勤飞行时长	0/9.68/285
最小/平均/最大 一次执勤执勤时长	0/11.84/675
最小/平均/最大 机组人员执勤天数	0/0.33/2
一/二/三/四/五天任务环数量分布	6/0/2/1/1
六/七/八/九/十天任务环数量分布	0/0/0/0/2
总体执勤成本（万元）	2.36
程序运行分钟数	105

Gurobi 求解过程中完整的控制台输出已附于附件中。

由于参赛时间和计算机内存的限制，本文设置了求解时间上限为 105 分钟，以期获得一个可行解。

针对求解时间上限为 105 分钟时，Gurobi 求解器求得的最优可行解，控制台输出显示此时求得的最优解与求解器的估计下界之间的差距（MinGAP）为 1434.9 %。由于模型规模较大与计算机性能限制，使用的笔记本电脑 16GB 内存无法满足要求，开辟了虚拟内存以供求解使用，从而导致求解速度进一步降低，考虑到较短的求解时间上限，因此该解的求解质量一般。

4.1.3. 结果分析

对比子问题 1、子问题 2 和子问题 3 的实验结果，当约束条件相对较少时，子问题 1 中所有的航班均满足了机组配置要求，可以正常飞行。而随着约束逐渐增多和优化目标更加多元，满足机组配置要求的航班数目略有减少但仍然非常可观。

此外，程序的运行时间显著提升很多，且最优解与求解器的估计下界的差距（MinGAP）逐渐变大。因此，为了防止内存不足的情况且在本次竞赛的有限时间内获取理想的解情况，子问题 2 的求解时间设为了 3 小时，由于模型规模较大、求解时间较短、计算机性能限制，子问题 3 的求解时间设为了 105 分钟。

整体而言，三个子问题的结果都非常理想，证明了模型构建的完整性和合理性。

4.2. 基于贪婪的近似求解

4.2.1. 求解算法概述

4.2.1.1. 子问题 1

针对子问题 1 设计了基于贪婪的深度优先近似求解算法。该求解算法的总体流程图如图 6 所示。

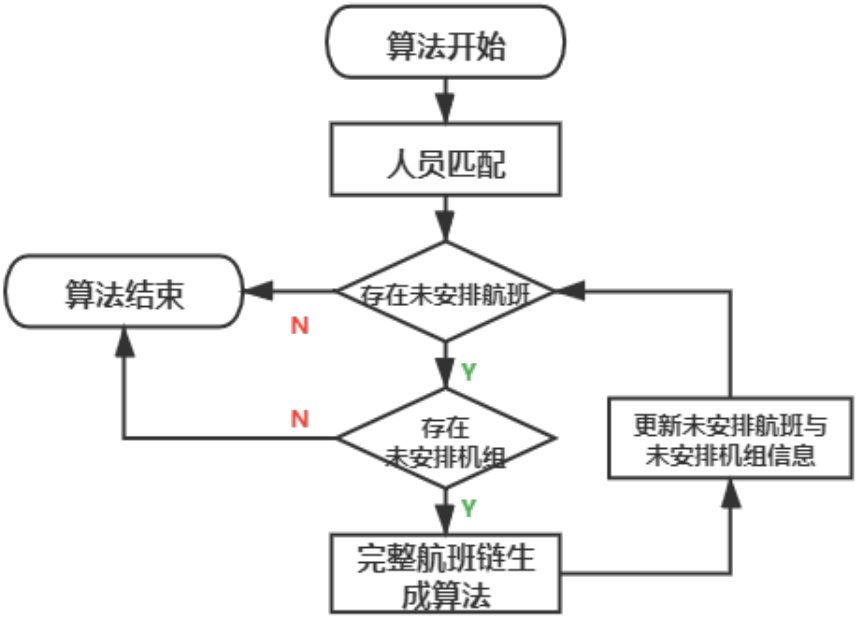

图 6 子问题 1 近似求解算法流程图

该算法首先考虑人员匹配问题，将所有机组成员按照 C1F1 的机组配置要求进行贪心组合，组合过程中优先考虑仅有主机长资格的机组成员任主机长，优先考虑仅有副机长资格的机组成员任副机长。随后检查所有航班组合中是否存在未安排航班和未被安排的机组成员组合。若两项均存在，则按照完整航班链生成算法生成针对该机组成员组合的航班链，并更新未安排航班与未安排机组成员组合信息；若两项有任意组合不存在，则终止算法。

关于完整航班链的生成算法的具体流程图如图 7 所示。

该算法依然基于贪婪的思想，对于每次输入的待安排机组人员组合和未安排航班集合，首先判断对当前航班链的最末航班，是否有满足当前时间约束（最小连接时间）与地点约束（出发地与到达地相同）的航班。若均存在，则直接选择该航班并更新航班链；若地点约束无法满足，则首先选择最早的别处起飞航班，同时查找是否存在链接当前地点和执飞地点的航班，且该航班满足时间约束，可以插入两航班之间，若存在，则在航班链中加入这两次航班，若不存在，则对于跳过该不满足地点约束航班，在以后对于当前航班链的查找中不再考虑该航班；若当前航班链的最末航班后已没有满足时间约束的航班，则直接终止算法。

在对航班链完成最终更新后，若最后一次航班不是回到基地的航班，则应删除该航班链中最后一次回到基地的航班之后的所有航班，并将这些被删除的航班重新计入未安排的航班。

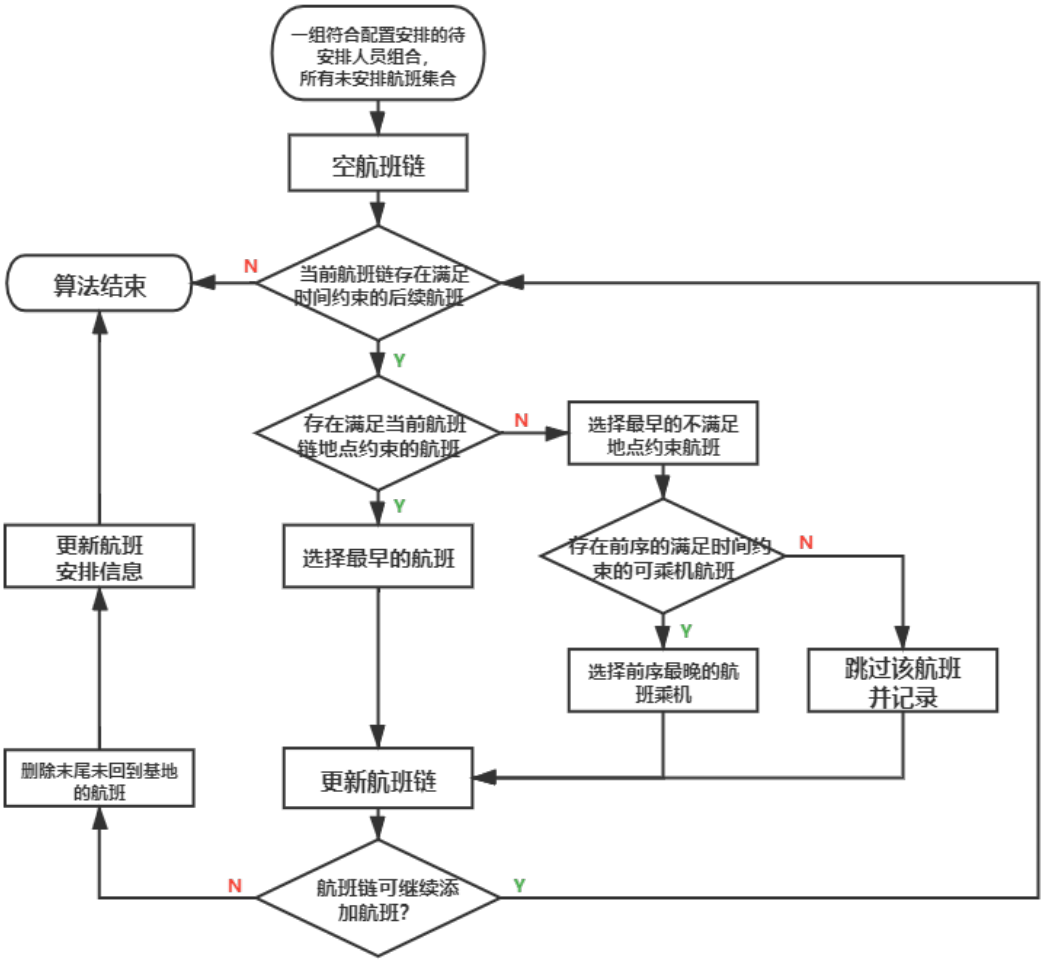

图 7 完整航班链生成算法流程示意图

4.2.1.2. 子问题 2

子问题 2 的总体求解流程与子问题 1 的总体流程一致（见图 6）。

但子问题而的航班链生成过程中需要额外考虑子问题 2 相对子问题 1 新加入的约束（执勤总时间、执勤间隔、执勤飞行时间），由于子问题 2 定义了执勤的概念，因此将子问题 1 中的航班链更名为执勤链，以方便理解。

在子问题 2 对应的执勤链生成算法如图 8 所示。

在该完整执勤链求解流程中，与子问题 1 中完整执勤链生成的区别主要在于时间约束的变化。在完整执勤链的生成过程中，需对当前执勤链的末尾航班的所在日期进行考虑，同时考虑待选航班能否被选中时需额外判定如下约束：

若末尾航班与待选航班在的起飞日期相同，则需额外判定待选航班加入后的最小执勤时间约束、最小飞行时间约束、最小连接时间约束。

若末尾航班在待选航班的起飞日期前一天，则需额外判定最小休息时间约束。

对于末尾航班后执飞插入地点不接续的航班的情况，则需对当前执勤链的末尾航班、待插入的乘机航班、待插入的地点不接续的执飞航班次三个航班同时考虑上述约束。

相似的，在对执勤链完成最终更新后，若最后一次航班不是回到基地的航班，则应删除该航班链中最后一次回到基地的航班之后的所有航班，并将这些被删除的航班重新计入未安排的航班。

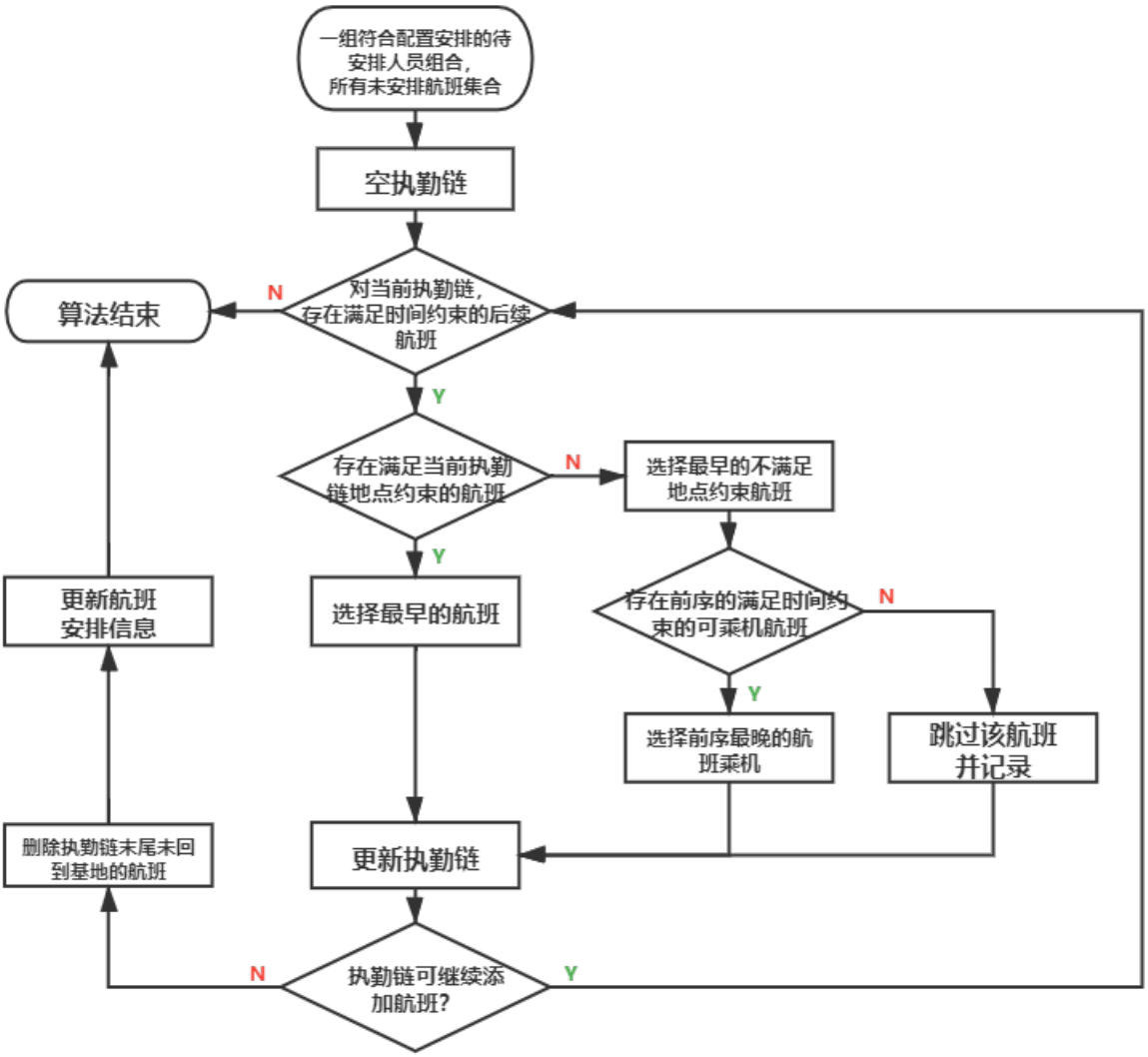

图 8 完整执勤链生成算法示意图

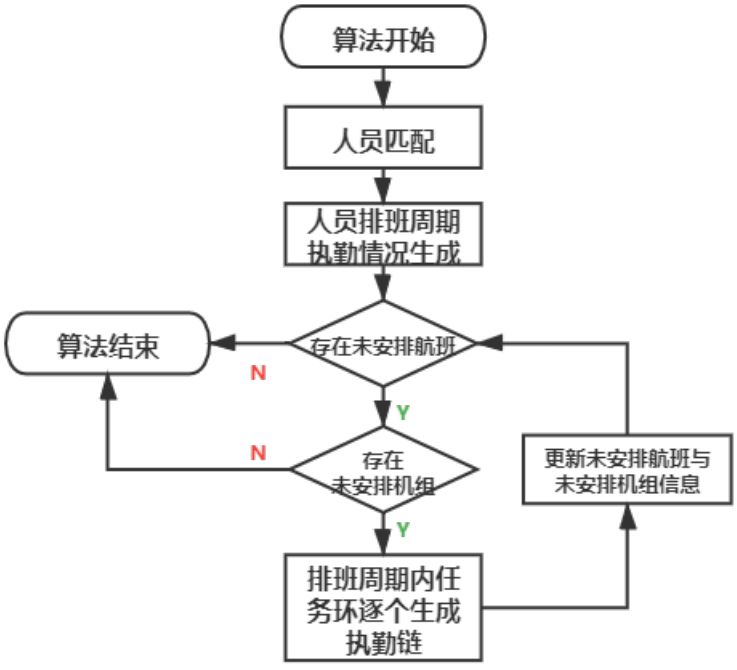

图 9 子问题 3 近似求解算法流程图

4.2.1.3. 子问题 3

子问题 3 相对于子问题 2 引入了任务环的概念，同时对连续执勤、不同任务环之间的休息时长等也进行了额外的限制。

子问题 3 的基于贪婪地近似求解算法流程如图 9 所示。

相对于子问题 1、子问题 2 的整体求解流程，子问题三基于贪婪的策略，同时考虑不同排班周期不同机组人员的任务环平衡，事先进行了排班周期内的执勤情况生成。同时，针对排班周期内的执勤情况，定义了任务环的起始与结束时点。

每次循环时，首先选择任一未进行排班的机组人员组合安排一个排班周期内的执勤情况，针对安排的排班周期内的每个任务环，按照子问题 2 中完整执勤链的生成思路，逐个生成一个完整的执勤链。然后选择下一个未进行排班的机组人员组合和排班周期执勤情况继续循环。直至所有航班安排完毕，或所有匹配好的机组人员安排完毕。

在针对排产周期内的任务环生成执勤链时，生成流程与子问题 2 中的生成流程一致，但需要额外考虑连续执勤时长的约束。由于在进行排班周期内执勤情况生成时已经安排了连续执勤后的休息时间，因此生成执勤链时只需额外保证待选航班的起飞日期不能与安排好的休息日相同。

排班周期内执勤情况的生成策略具体描述如图 10 所示。

为保证排班周期内不同日期的覆盖情况，加入随机数以保证最终执勤日期覆盖的平衡性。流程中的检查某日期是否为可执勤主要检查该日期的前序安排是否满足连续执勤天数与任务环之间休息间隔（MinRest）约束。

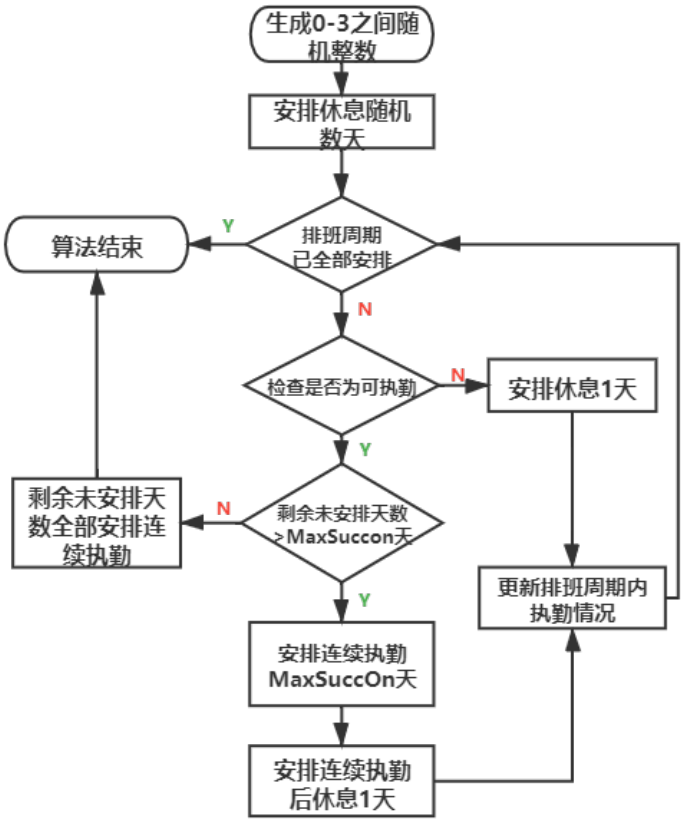

图 10 排班周期内执勤情况生成算法流程图

4.2.2. 求解结果与分析

4.2.2.1. 子问题 1 求解结果

表 7 子问题 1 近似求解结果指标

结果指标	DATA-A	DATA-B
不满足机组配置航班数	0	50
满足机组配置航班数	206	13904
机组人员总体乘机次数	8	168
替补资格使用次数	0	2029
程序运行分钟数	0.02	1

4.2.2.2. 子问题 2 求解结果

表 8 子问题 2 近似求解结果指标

结果指标	DATA-A	DATA-B
不满足机组配置航班数	9	763
满足机组配置航班数	197	13191
机组人员总体乘机次数	174	3642
替补资格使用次数	0	1541
程序运行分钟数	0.03	1

4.2.2.3. 子问题 3 求解结果

表 9 子问题 3 近似求解结果指标

结果指标	DATA-A	DATA-B
不满足机组配置航班数	14	1724
满足机组配置航班数	192	12230
机组人员总体乘机次数	104	1806
替补资格使用次数	0	1171
程序运行分钟数	0.03	1

4.2.2.4. 结果分析

分析以上算法的算法复杂度，发现基于贪婪的近似求解算法的算法时间复杂度在多项式阶次内。根据问题规模，可以保证在多项式时间内得到近似解。

分析近似求解算法的求解结果。首先，近似求解算法的运行时间均在可接受范围内。对于数据集 A，由于数据规模较小，程序运行时间均在 3s 以内。对于数据集 B，即使数据集的规模急剧扩大,近似算法也能保证在 1 分钟左右完成求解,求解效率得到了有效保证。其次，近似算法的求解结果指标均在可接受范围内。以上分析证明了近似算法的高效便捷性。

将近似求解算法的求解结果与模型精确求解结果对比。近似求解算法在对按目标优先级的求解上与模型的精确结果差距不大。以子问题 2 为例，相同的数据集 A 下，模型精确求解在 3 个小时的求解时间下依然未得到全局最优解，但基于贪婪的近似求解算法在极短的时间内迅速得到了与精确求解的可行解首要目标差距在 5%以内、次要目标差距不大的解，对于子问题 1，近似算法甚至在 1s 左右的求解时间中求出了与全局最优解首要目标相同、次要目标更优的解。

相对于严谨的数学表述，近似求解算法将认为经验与业务实际背景相结合，更加有针对性和高效地解决了在时间复杂度上较高的实际问题，相较于数学模型，更易于理解，也更便于根据人为经验调整，更加直观便捷。

以上分析充分证明了设计并使用近似算法求解的有效性和必要性。

五、结论与展望

综上所述，我们针对三个递进的多目标、多复杂约束的机组排班问题，建立了三个混合整数线性规划模型。通过 Gurobi 寻找其准确全局解，实验结果证明了模型结构的完善性和合理性。然后又设计相应的基于贪婪和深度优先搜索的启发式算法寻找近似解，实验结果证明了启发式算法的高效性。再通过与精确解的对比，发现两者的解情况十分接近，又进一步证实了启发式算法的可靠性。

虽然本文所建立的模型和提出的求解算法效果如此可观，它们依然有一定的不足之处，可作为后续研究的探索方向。

文章所建立的数学模型，虽在编程过程中已对变量数目和约束数目进行了优化，但其时间和空间规模依然较大，在求解过程中对于时间和存储资源的消耗也较大，分析问题的具体情况，找到可能存在的冗余约束是未来可以探索的方向。

针对问题中提到的具有优先级的多个目标共同优化，本文采用了设置权重的方式将多个目标单目标化，以降低求解的复杂度。但这种设置方式主要为主观的感性考量，缺乏客观的数据化的分析来真正平衡目标重要程度。寻找一个更加合理的权重设置方式或多目标求解模型是另外一个未来探索的方向。

六、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. 吴宇光. 航空公司机组排班的优化模型研究与算法实现[D].复旦大学,2012.
3. Tahir A, Quesnel F, Desaulniers G, et al. An Improved Integral Column Generation Algorithm Using Machine Learning for Aircrew Pairing[J]. Transportation Science, 2021.
4. Souai N, Teghem J. Genetic algorithm based approach for the integrated airline crew-pairing and rostering problem[J]. European Journal of Operational Research, 2009, 199(3): 674-683.
5. Kornilakis H, Stamatopoulos P. Crew pairing optimization with genetic algorithms[C]//Hellenic conference on artificial intelligence. Springer, Berlin, Heidelberg, 2002: 109-120.
6. Ahmed M B, Hryhoryeva M, Hvattum L M, et al. A matheuristic for the robust integrated airline fleet assignment, aircraft routing, and crew pairing problem[J]. Computers & Operations Research, 2021: 105551.
7. 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
8. Özener O Ö, Matoğlu M Ö, Erdoğan G, et al. Solving a large-scale integrated fleet assignment and crew pairing problem[J]. Annals of Operations Research, 2017, 253(1): 477-500.
9. Ahmed M B, Hryhoryeva M, Hvattum L M, et al. A matheuristic for the robust integrated airline fleet assignment, aircraft routing, and crew pairing problem[J]. Computers & Operations Research, 2021: 105551.
10. Baradaran V, Hosseinian A H. A Multi-Objective Mathematical Formulation for the Airline Crew Scheduling Problem: MODE and NSGA-II Solution Approaches[J]. Journal of Industrial Management Perspective, 2021, 11(1, Spring 2021): 247-269.

11. Özkır V, Özgür M S. Two-Phase Heuristic Algorithm for Integrated Airline Fleet Assignment and Routing Problem[J]. Energies, 2021, 14(11): 3327.

七、附录

本文编写的所有代码均附与后续附录中。

附录

```
##############################
####### Read DATA ############
##############################

import numpy as np
import pandas as pd
import os

BASE_MAP = {"NKX":0, "HOM":0, "TGD":1}
OUTPUT_DIR = "./data/"
if not os.path.exists(OUTPUT_DIR):
    os.mkdir(OUTPUT_DIR)
# 将机组人员数据 csv读取到 dataframe
def read_crew_data_to_dataframe(crew_data_csv_file):
    df_crew= pd.read_csv(crew_data_csv_file, sep=",")
    if "ParingCostPerHr" in df_crew.columns.values:
        df_crew["ParingCostPerHour"] = df_crew["ParingCostPerHr"]
    if "DutyCostPerHr" in df_crew.columns.values:
        df_crew["DutyCostPerHour"] = df_crew["DutyCostPerHr"]

    df_crew["Captain"] = df_crew["Captain"].map(lambda x: 1 if x=="Y" else 0)
    df_crew["FirstOfficer"] = df_crew["FirstOfficer"].map(lambda x: 1 if x=="Y" else 0)
    df_crew["Deadhead"] = df_crew["Deadhead"].map(lambda x: 1 if x=="Y" else 0)
    df_crew["EmpNoInt"] = df_crew["EmpNo"].map(lambda x:int(x[1:])-1)
    df_crew["BaseInt"] = df_crew["Base"].map(lambda x:BASE_MAP[x])
    df_crew = df_crew[["EmpNo","EmpNoInt","Captain","FirstOfficer",\
        "Deadhead","Base","BaseInt","DutyCostPerHour","ParingCostPerHour"]]
    print(df_crew)
    return df_crew



# test
crew_data_A_file = "./data/机组排班Data␣A-Crew.csv"
crew_data_B_file = "./data/机组排班Data␣B-Crew.csv"

df_crew_A = read_crew_data_to_dataframe(crew_data_A_file)
df_crew_B = read_crew_data_to_dataframe(crew_data_B_file)

df_crew_A.to_csv(os.path.join(OUTPUT_DIR,"prepared_crew_A.csv"),index=False)
df_crew_B.to_csv(os.path.join(OUTPUT_DIR,"prepared_crew_B.csv"),index=False)

# 将航班数据 csv读取到 dataframe
def read_flight_data_to_dataframe(flight_data_csv_file):
    df_flight = pd.read_csv(flight_data_csv_file, sep=",")



    df_flight["DptrDate"] = pd.to_datetime(df_flight["DptrDate"])
```

```
    df_flight["DptrTime"] = pd.to_datetime(df_flight["DptrTime"], format='%H:%M').dt.time
    df_flight["ArrvDate"] = pd.to_datetime(df_flight["ArrvDate"])
    df_flight["ArrvTime"] = pd.to_datetime(df_flight["ArrvTime"], format='%H:%M').dt.time

    min_timestamp = (df_flight[["DptrDate","ArrvDate"]].min()).min()
    # print("timestamp origin:", min_timestamp)
    df_flight["DptrDateInt"] = df_flight["DptrDate"] - min_timestamp
    df_flight["DptrDateInt"] = df_flight["DptrDateInt"].map(lambda x:x.days)
    df_flight["ArrvDateInt"] = df_flight["ArrvDate"] - min_timestamp
    df_flight["ArrvDateInt"] = df_flight["ArrvDateInt"].map(lambda x:x.days)

    df_flight["DptrTimeInt"] = df_flight["DptrTime"].map(lambda x:x.hour*60+x.minute)
    df_flight["ArrvTimeInt"] = df_flight["ArrvTime"].map(lambda x:x.hour*60+x.minute)
    df_flight["DptrTimestamp"] = df_flight["DptrDateInt"]*60*24 + df_flight["DptrTimeInt"]
    df_flight["ArrvTimestamp"] = df_flight["ArrvDateInt"]*60*24 + df_flight["ArrvTimeInt"]

    ports = list(pd.unique(df_flight["DptrStn"].append(df_flight["ArrvStn"])))
    port_map = {}
    for base in BASE_MAP.keys():
        if base in ports:
            ports.remove(base)
            port_map[base] = BASE_MAP[base]
    base_count = len(port_map)
    port_map.update({port:i+base_count for (i,port) in enumerate(ports)})
    assert len(port_map) == len(ports) + base_count
    df_flight["DptrStnInt"] = df_flight["DptrStn"].map(lambda x:port_map[x])
    df_flight["ArrvStnInt"] = df_flight["ArrvStn"].map(lambda x:port_map[x])
    df_flight = df_flight.sort_values(by="DptrTimestamp",ignore_index=True)

    df_flight["FltIndex"] = df_flight.index

    df_flight = df_flight[["FltIndex","FltNum","DptrDate","DptrDateInt","DptrTime",\
        "DptrTimeInt","DptrTimestamp","DptrStn","DptrStnInt","ArrvDate","ArrvDateInt",\
            "ArrvTime","ArrvTimeInt","ArrvTimestamp","ArrvStn","ArrvStnInt","Comp"]]
    print(df_flight)
    return df_flight

# test
flight_data_A_file = "./data/机组排班Data␣A-Flight.csv"
flight_data_B_file = "./data/机组排班Data␣B-Flight.csv"

df_flight_A = read_flight_data_to_dataframe(flight_data_A_file)
df_flight_B = read_flight_data_to_dataframe(flight_data_B_file)

df_flight_A.to_csv(os.path.join(OUTPUT_DIR,"prepared_flight_A.csv"),index=False)
df_flight_B.to_csv(os.path.join(OUTPUT_DIR,"prepared_flight_B.csv"),index=False)

# data analysis
flight_A_dep_or_arr_base = (df_flight_A["DptrStnInt"].to_numpy() == 0) +\
```

```
    (df_flight_A["ArrvStnInt"].to_numpy() == 0)
print(np.all(flight_A_dep_or_arr_base))

flight_B_dep_or_arr_base = (df_flight_B["DptrStnInt"].to_numpy() == 1) +\
    (df_flight_B["ArrvStnInt"].to_numpy() == 1) +\
    (df_flight_B["DptrStnInt"].to_numpy() == 0) +\
        (df_flight_B["ArrvStnInt"].to_numpy() == 0)
print("Rate that a flight departs from or arrives in base:", \
    np.sum(flight_B_dep_or_arr_base)/flight_B_dep_or_arr_base.shape[0])
df_flight_B[~flight_B_dep_or_arr_base]

df_flight_B["FltTime"] = df_flight_B["ArrvTimestamp"] - df_flight_B["DptrTimestamp"]
print("Max flight time:",df_flight_B["FltTime"].max())
print("Min flight time:",df_flight_B["FltTime"].min())

df_flight_A["FltTime"] = df_flight_A["ArrvTimestamp"] - df_flight_A["DptrTimestamp"]
print("Max flight time:",df_flight_A["FltTime"].max())
print("Min flight time:",df_flight_A["FltTime"].min())
```

```
############################
####### TASK 1 ############
############################
import gurobipy as grb
import numpy as np
import pandas as pd
import os
import json

DATA = "A"
DATA_PATH = "./data/"
PREPARED_CREW_FILE = os.path.join(DATA_PATH, "prepared_crew_{}.csv".format(DATA))
PREPARED_FLIGHT_FILE = os.path.join(DATA_PATH, "prepared_flight_{}.csv".format(DATA))

dfCrew = pd.read_csv(PREPARED_CREW_FILE, sep=",")
dfFlight = pd.read_csv(PREPARED_FLIGHT_FILE, sep=",")

# 每个航班最大deadhead人数
# MaxDH = 5
# 大数
L = int(1E6)
# 最小连接时间
MinCT = 40

# 员工集合
I = list(np.sort(pd.unique(dfCrew["EmpNoInt"])))
# 航班集合
F = list(np.sort(pd.unique(dfFlight["FltIndex"])))

# 角色集合 0:主机长 1: 副机长 2: 搭乘
R = [0,1,2]
# 日期集合
D = list(np.sort(pd.unique(dfFlight["DptrDateInt"])))
# 机场集合
A = np.sort(np.unique(list(dfFlight["DptrStnInt"])+list(dfFlight["ArrvStnInt"])))
print("Airport set:", A)
# 员工基地map # B[i]
B = {i:dfCrew.loc[i,"BaseInt"] for i in I}
B_set = list(np.unique(list(B.values())))
BASE = 0
assert B_set == [0]
print("Base set:", B_set)

# 虚拟开始结束航班 + 航班集合
VirF = F + [-1,-2] # -1:源 -2: 汇
print("Virtual F:", VirF)

# 对于j属于F, 具有上一个或下一个航班可能性的集合
NextF = {j:[] for j in F}
```

```
PrevF = {j:[] for j in F}
# 虚拟航班下一航班列表，虚拟航班无上一航班
NextF[-1] = list(dfFlight.loc[dfFlight["DptrStnInt"]==BASE,"FltIndex"].index) + [-2]
PrevF[-1] = []
NextF[-2] = []
PrevF[-2] = list(dfFlight.loc[dfFlight["ArrvStnInt"]==BASE,"FltIndex"].index) + [-1]

# 不存在上一个和下一个航班的航班集合
LastF = [-2,] # VirF - F1
FirstF = [-1,] # VirF - F2

# j1 > j2 if Dpt[j1]>=Dpt[j2]
# for j in F:
#     NextF[j] = [jj for jj in F if jj > j] + [-2]
#     PrevF[j] = [jj for jj in F if jj < j] + [-1]

for j in F:
    arrvTimestamp_j = dfFlight.loc[j, "ArrvTimestamp"]
    arrvStnInt_j = dfFlight.loc[j, "ArrvStnInt"]
    possibleNextFlightFlag = (dfFlight["DptrStnInt"]==arrvStnInt_j)&\
        (dfFlight["DptrTimestamp"]>arrvTimestamp_j)
    if arrvStnInt_j in B_set:
        NextF[j].append(-2) #虚拟结束航班是任何目的地为base航班的下一个可能航班
    if np.any(possibleNextFlightFlag):
        NextF[j].extend(list(dfFlight.loc[possibleNextFlightFlag, "FltIndex"]))
    if not NextF[j]:
        print("no possible next flight for: ", j)
        LastF.append(j)
for j in F:
    dptrTimestamp_j = dfFlight.loc[j, "DptrTimestamp"]
    dptrStnInt_j = dfFlight.loc[j, "DptrStnInt"]
    possiblePrevFlightFlag = (dfFlight["ArrvStnInt"]==dptrStnInt_j)&\
        (dfFlight["ArrvTimestamp"]<dptrTimestamp_j)
    if (dptrStnInt_j in B_set):
        PrevF[j].append(-1) #虚拟开始航班是任何出发地为base航班的上一个可能航班
    if np.any(possiblePrevFlightFlag):
        PrevF[j].extend(list(dfFlight.loc[possiblePrevFlightFlag, "FltIndex"]))
    if not PrevF[j]:
        print("no possible previous flight for: ", j)
        FirstF.append(j)

print("no possible next flight for: ", LastF)
print("no possible previous flight for: ", FirstF)

# NotLastF = F - lastF
NotLastF = list(set(VirF)-set(LastF)) # F1
NotLastF.sort()
assert (-1 in NotLastF)
assert (-2 not in NotLastF)
```

```
# NotFirstF = F - FirstF
NotFirstF = list(set(VirF)-set(FirstF)) # F2
NotFirstF.sort()
assert(-1 not in NotFirstF)
assert(-2 in NotFirstF)



for j in F:
    assert NextF[j]
    assert PrevF[j]

jNextFj = []
for j in NotLastF:
    for jj in NextF[j]:
        jNextFj.append((j,jj))
jNextFj = set(jNextFj)

jPrevFj = []
for jj in NotFirstF:
    for j in PrevF[jj]:
        jPrevFj.append((j,jj))
jPrevFj = set(jPrevFj)

print((jPrevFj|jNextFj)-(jPrevFj&jNextFj))
print(len(jPrevFj|jNextFj))
print(len(jPrevFj&jNextFj))

assert len(jNextFj)==len(jPrevFj)
assert jNextFj == jPrevFj

# print(NextF[0])
# print(NextF[F[-1]])

# 航班起落机场0-1参数
f = np.zeros((len(VirF),len(A),len(A)), dtype=np.int8)
for base in B_set:
    f[-1,base,base] = 1 # 虚拟航班 从base开到同一个base
    f[-2,base,base] = 1
for j in F:
    for m in A:
        for n in A:
            f[j,m,n] = 1 if (dfFlight.loc[j,"DptrStnInt"]==m)and\
                (dfFlight.loc[j,"ArrvStnInt"]==n) else 0

# 航班起飞绝对时间
DpT = {j:dfFlight.loc[j,"DptrTimestamp"] for j in F}
DpT[-1] = -MinCT # 虚拟开始航班出发时间
DpT[-2] = dfFlight["ArrvTimestamp"].max()+MinCT+10 # 虚拟结束航班出发时间
# 航班落地绝对时间
```

```
ArT = {j:dfFlight.loc[j,"ArrvTimestamp"] for j in F}
ArT[-1] = -MinCT # 虚拟开始航班到达时间
ArT[-2] = dfFlight["ArrvTimestamp"].max()+MinCT+10 # 虚拟结束航班到达时间

# 机长资格0-1
EpsC = {i:1 if dfCrew.loc[i,"Captain"]==1 else 0 for i in I}
# 副机长资格
EpsF = {i:1 if dfCrew.loc[i,"FirstOfficer"]==1 else 0 for i in I}




# create a gurobi model
model = grb.Model("model_task1")
model.setParam("TimeLimit", 3600)
model.setParam("MIPGap", 0.001)
# 决策变量 X_i,j,r==1 说明 员工i以角色r身份登上飞机j
X = np.empty((len(I),len(VirF),len(R)),dtype=object)
for i in I:
    for j in VirF:
        for r in R:
            X[i,j,r] = model.addVar(vtype=grb.GRB.BINARY)

## 决策变量 Y_i_j_j'==1 对员工i而言, 航班j'是他乘坐航班j后的下一个航班
Y = np.empty((len(I),len(VirF),len(VirF)),dtype=object)
for i in I:
    for j in NotLastF: #包括-1不包括-2
        for jj in NextF[j]:
            Y[i,j,jj] = model.addVar(vtype=grb.GRB.BINARY)



i_captain = dfCrew.loc[(dfCrew["Captain"]==1)&(dfCrew["FirstOfficer"]==0),\
    "EmpNoInt"].idxmin()
i_officer = dfCrew.loc[(dfCrew["Captain"]==0)&(dfCrew["FirstOfficer"]==1), \
    "EmpNoInt"].idxmin()

# 虚拟航班约束X
for i in I:
    if i==i_captain:
        model.addConstr((X[i,-1,0]==1))
        model.addConstr((X[i,-2,0]==1))
    elif i==i_officer:
        model.addConstr((X[i,-1,1]==1))
        model.addConstr((X[i,-2,1]==1))
    else:
        model.addConstr((X[i,-1,2]==1))
        model.addConstr((X[i,-2,2]==1))
```

```
# 约束: 员工i同一航班不能身兼多职
for i in I:
    for j in VirF:
        model.addConstr((grb.quicksum(X[i,j,r] for r in R)<=1))

# 约束: 机长资格确认 r=0
for i in I:
    for j in VirF:
        model.addConstr((X[i,j,0]<=EpsC[i]))

# 约束: 副机长资格确认 r=1
for i in I:
    for j in VirF:
        model.addConstr((X[(i,j,1)]<=EpsF[i]))

# 约束: C1F1 和 约束: C=F 一个机长配一个副机长或者都为0 和
# 约束: 任意航班deadhead人数不超过MaxDH=5
for j in VirF:
    model.addConstr((grb.quicksum(X[i,j,0]*EpsC[i] for i in I)<=1))
    model.addConstr((grb.quicksum(X[i,j,1]*EpsF[i] for i in I)<=1))
    model.addConstr((grb.quicksum(X[i,j,0]*EpsC[i] for i in I) == \
        grb.quicksum(X[i,j,1]*EpsF[i] for i in I) )) # =0 or =1
    # model.addConstr((grb.quicksum(X[i,j,2] for i in I) <= MaxDH))

# 约束: 航班取消, 员工不执行该项工作
for i in I:
    for j in VirF:
        model.addConstr((grb.quicksum(X[i,j,r] for r in R) <= \
            grb.quicksum(X[ii,j,0]*EpsC[ii] for ii in I)))


# Y[(i,j,j')]本质约束
for i in I:
    for j in NotLastF:
        model.addConstr((grb.quicksum(Y[i,j,jj] for jj in NextF[j]) <= 1))
for i in I:
    for jj in NotFirstF:
        model.addConstr((grb.quicksum(Y[i,j,jj] for j in PrevF[jj]) <= 1))


# 约束: 员工i从他基地出发的次数等于回到基地的次数
A_except_base = {i:A for i in I}# {i:list(set(A)-set([B[i]])) for i in I}
for i in I:
    model.addConstr((grb.quicksum(X[i,j,r]*f[j,B[i],n] for r in R \
        for n in A_except_base[i] for j in VirF) \
        == grb.quicksum(X[i,j,r]*f[j,m,B[i]] for r in R \
            for m in A_except_base[i] for j in VirF) ))

# 约束: 员工i的航班前后连续性 Y[i,j,j']
```

```
for k in A:
    for i in I:
        for j in NotLastF: # j包括-1不包括-2
            for jj in NextF[j]: # jj不包括-1包括-2
                model.addConstr((grb.quicksum(X[i,jj,r]*f[jj,k,n]\
                    for r in R for n in A_except_base[i]) -\
                        grb.quicksum(X[i,j,r]*f[j,m,k]\
                            for r in R for m in A_except_base[i]) +\
                            L*(3-grb.quicksum(X[i,j,r] for r in R) -\
                                grb.quicksum(X[i,jj,r]for r in R)\
                                    -Y[i,j,jj]) >= 0 ))
                model.addConstr((grb.quicksum(X[i,jj,r]*f[jj,k,n]\
                    for r in R for n in A_except_base[i]) -\
                        grb.quicksum(X[i,j,r]*f[j,m,k] \
                            for r in R for m in A_except_base[i]) -\
                            L*(3-grb.quicksum(X[i,j,r] for r in R) -\
                                grb.quicksum(X[i,jj,r]for r in R)\
                                    -Y[i,j,jj]) <= 0 ))
# 约束：XY的关系
for i in I:
    for j in NotLastF: #F1
        for jj in NextF[j]:
            model.addConstr((L*(1-Y[i,j,jj]) + grb.quicksum(X[i,j,r] for r in R) \
                + grb.quicksum(X[i,jj,r] for r in R) >= 2))

# 约束：连接时间不小于MinCT分钟
for i in I:
    for j in NotLastF: # 包括-1
        for jj in NextF[j]: # jj不包括-1
            model.addConstr((DpT[jj]-ArT[j]+(1-Y[i,j,jj])*L >= MinCT ))

for i in I:
    for j in NotLastF: #F1
        model.addConstr((L*(1-grb.quicksum(X[i,j,r] for r in R))\
            +grb.quicksum(Y[i,j,jj] for jj in NextF[j]) >= 1))
        model.addConstr((-L*(1-grb.quicksum(X[i,j,r] for r in R))\
            +grb.quicksum(Y[i,j,jj] for jj in NextF[j]) <= 1))

for i in I:
    for jj in NotFirstF: #F2
        model.addConstr((L*(1-grb.quicksum(X[i,jj,r] for r in R))\
            +grb.quicksum(Y[i,j,jj] for j in PrevF[jj]) >= 1))
        model.addConstr((-L* (1-grb.quicksum(X[i,jj,r] for r in R))\
             +grb.quicksum(Y[i,j,jj] for j in PrevF[jj]) <= 1))
```

```
# 1st objective: 最大化可执行航班
objMaxFlightNum = (grb.quicksum(X[i,j,0] for i in I for j in F))
# 4th objective: 最小化总体deadhead次数
objMinDeadheadNum = (grb.quicksum(X[i,j,2] for i in I for j in F))
# 7th objective: 最小化替补资格
objMinBenchNum = (grb.quicksum(X[i,j,1]*EpsC[i]*EpsF[i] for i in I for j in F))



# 目标函数
w1 = 10
w2 = 1
w3 = 0.1
objMinCost = (-w1*objMaxFlightNum + w2*objMinDeadheadNum + w3*objMinBenchNum)
model.setObjective(objMinCost, grb.GRB.MINIMIZE)

model.presolve()

model.optimize()

# Save raw results to json file
result = {}
result['objMaxFlightNum'] = objMaxFlightNum.getValue()
result['objMinDeadheadNum'] = objMinDeadheadNum.getValue()
result['objMinBenchNum'] = objMinBenchNum.getValue()
result['X'] = {}
for i in I:
    for j in VirF:
        for r in R:
            result['X'][str("{},{},{}".format(i,j,r))] = X[i,j,r].x

result['Y'] = {}
for i in I:
    for j in NotLastF:
        for jj in NextF[j]:
            result['Y'][str("{},{},{}".format(i,j,jj))] = Y[i,j,jj].x
# dataName is "day_data" or "week_data"
with open("./raw_results_task1.json", 'w') as f:
    json.dump(result, f)
```

```
############################
####### TASK 2 ############
############################
import gurobipy as grb
import numpy as np
import pandas as pd
import os
import json

DATA = "A"
DATA_PATH = "./data/"
PREPARED_CREW_FILE = os.path.join(DATA_PATH,\
      "prepared_crew_{}.csv".format(DATA))
PREPARED_FLIGHT_FILE = os.path.join(DATA_PATH,\
      "prepared_flight_{}.csv".format(DATA))

dfCrew = pd.read_csv(PREPARED_CREW_FILE, sep=",")
dfFlight = pd.read_csv(PREPARED_FLIGHT_FILE, sep=",")

# 每个航班最大deadhead人数
# MaxDH = 5
# 大数
L = int(1E6)

MinCT = 40 # 最小连接时间
MaxBlk = 600 # 一次执勤飞行时长最多不超过
MaxDP = 720 # • 执勤时长最多不超过
MinRest = 660 # 相邻执勤之间的休息时间不少于
MaxTAFB = 14400 # 排班周期单个机组人员任务环总时长

# 员工集合
I = list(np.sort(pd.unique(dfCrew["EmpNoInt"])))
# 航班集合
F = list(np.sort(pd.unique(dfFlight["FltIndex"])))

# 角色集合 0:主机长 1: 副机长 2: 搭乘
R = [0,1,2]
# 日期集合
D = list(np.sort(pd.unique(dfFlight["DptrDateInt"])))
# 机场集合
A = np.sort(np.unique(list(dfFlight["DptrStnInt"])+\
    list(dfFlight["ArrvStnInt"])))
print("Airport set:", A)
# 员工基地map # B[i]
B = {i:dfCrew.loc[i,"BaseInt"] for i in I}
B_set = list(np.unique(list(B.values())))
BASE = 0
assert B_set == [0]
```

```
print("Base set:", B_set)

# 虚拟开始结束航班 + 航班集合
VirF = F + [-1,-2] # -1:源 -2: 汇
# print("Virtual F:", VirF)

# 对于j属于F, 具有上一个或下一个航班可能性的集合
NextF = {j:[] for j in F}
PrevF = {j:[] for j in F}
# 虚拟航班下一航班列表, 虚拟航班无上一航班
NextF[-1] = list(dfFlight.loc[dfFlight["DptrStnInt"]==BASE,\
    "FltIndex"].index) + [-2]
PrevF[-1] = []
NextF[-2] = []
PrevF[-2] = list(dfFlight.loc[dfFlight["ArrvStnInt"]==BASE,\
    "FltIndex"].index) + [-1]

# 不存在上一个和下一个航班的航班集合
LastF = [-2,] # VirF - F1
FirstF = [-1,] # VirF - F2

# j1 > j2 if Dpt[j1]>=Dpt[j2]
# for j in F:
#     NextF[j] = [jj for jj in F if jj > j] + [-2]
#     PrevF[j] = [jj for jj in F if jj < j] + [-1]

for j in F:
    arrvTimestamp_j = dfFlight.loc[j, "ArrvTimestamp"]
    arrvStnInt_j = dfFlight.loc[j, "ArrvStnInt"]
    possibleNextFlightFlag = (dfFlight["DptrStnInt"]==arrvStnInt_j)\
        &(dfFlight["DptrTimestamp"]>arrvTimestamp_j)
    if arrvStnInt_j in B_set:
        NextF[j].append(-2) #虚拟结束航班是任何目的地为base航班的下一个可能航班
    if np.any(possibleNextFlightFlag):
        NextF[j].extend(list(dfFlight.loc[possibleNextFlightFlag,\
            "FltIndex"]))
    if not NextF[j]:
        print("no possible next flight for:", j)
        LastF.append(j)
for j in F:
    dptrTimestamp_j = dfFlight.loc[j, "DptrTimestamp"]
    dptrStnInt_j = dfFlight.loc[j, "DptrStnInt"]
    possiblePrevFlightFlag = (dfFlight["ArrvStnInt"]==dptrStnInt_j)\
        &(dfFlight["ArrvTimestamp"]<dptrTimestamp_j)
    if (dptrStnInt_j in B_set):
        PrevF[j].append(-1) #虚拟开始航班是任何出发地为base航班的上一个可能航班
    if np.any(possiblePrevFlightFlag):
        PrevF[j].extend(list(dfFlight.loc[possiblePrevFlightFlag, "FltIndex"]))
    if not PrevF[j]:
```

```
        print("no possible previous flight for: ", j)
        FirstF.append(j)

print("no possible next flight for: ", LastF)
print("no possible previous flight for: ", FirstF)

# NotLastF = F - lastF
NotLastF = list(set(VirF)-set(LastF)) # F1
NotLastF.sort()
assert (-1 in NotLastF)
assert (-2 not in NotLastF)
# NotFirstF = F - FirstF
NotFirstF = list(set(VirF)-set(FirstF)) # F2
NotFirstF.sort()
assert(-1 not in NotFirstF)
assert(-2 in NotFirstF)



for j in F:
    assert NextF[j]
    assert PrevF[j]

jNextFj = []
for j in NotLastF:
    for jj in NextF[j]:
        jNextFj.append((j,jj))
jNextFj = set(jNextFj)

jPrevFj = []
for jj in NotFirstF:
    for j in PrevF[jj]:
        jPrevFj.append((j,jj))
jPrevFj = set(jPrevFj)

print((jPrevFj|jNextFj)-(jPrevFj&jNextFj))
print(len(jPrevFj|jNextFj))
print(len(jPrevFj&jNextFj))

assert len(jNextFj)==len(jPrevFj)
assert jNextFj == jPrevFj

## 问题2新增

DutyCost = dfCrew["DutyCostPerHour"].to_numpy()

# 同一天出发的航班集合
FbyD = {d:list(dfFlight.loc[dfFlight["DptrDateInt"]==d,"FltIndex"]) for d in D}
# 航班j同一天出发的航班里接下来可能的下一班航班(不考虑虚拟航班)
NextFSameD = {j:[] for j in F}
```

```
for j in F:
    dptrDateInt_j = dfFlight.loc[j,"DptrDateInt"]
    arrvTimestamp_j = dfFlight.loc[j, "ArrvTimestamp"]
    possibleNextFlightFlag = (dfFlight["DptrTimestamp"]>arrvTimestamp_j)
    possibleNextFSameDayFlag = (dfFlight["DptrDateInt"]==dptrDateInt_j)\
        &possibleNextFlightFlag
    if np.any(possibleNextFSameDayFlag):
        NextFSameD[j] = list(dfFlight.loc[possibleNextFSameDayFlag, "FltIndex"])

LastFSameD = {k:[j for j in FbyD[k] if not NextFSameD[j]] for k in D}
 #同一天里不存在接续航班的航班集合
NotLastFSameD = {k:list(set(FbyD[k])-set(LastFSameD[k])) for k in D}
 #同一天里存在接续航班的航班集合

for k in D:
    assert set(LastFSameD[k])|set(NotLastFSameD[k]) == set(FbyD[k])




# 航班起落机场0-1参数
f = np.zeros((len(VirF),len(A),len(A)), dtype=np.int8)
for base in B_set:
    f[-1,base,base] = 1 # 虚拟航班 从base开到同一个base
    f[-2,base,base] = 1
for j in F:
    for m in A:
        for n in A:
            f[j,m,n] = 1 if (dfFlight.loc[j,"DptrStnInt"]==m)and\
                (dfFlight.loc[j,"ArrvStnInt"]==n) else 0

# 航班起飞绝对时间
DpT = {j:dfFlight.loc[j,"DptrTimestamp"] for j in F}
DpT[-1] = -MinCT # 虚拟开始航班出发时间
DpT[-2] = dfFlight["ArrvTimestamp"].max()+MinCT+10 # 虚拟结束航班出发时间
# 航班落地绝对时间
ArT = {j:dfFlight.loc[j,"ArrvTimestamp"] for j in F}
ArT[-1] = -MinCT # 虚拟开始航班到达时间
ArT[-2] = dfFlight["ArrvTimestamp"].max()+MinCT+10 # 虚拟结束航班到达时间

# 机长资格0-1
EpsC = {i:1 if dfCrew.loc[i,"Captain"]==1 else 0 for i in I}
# 副机长资格
EpsF = {i:1 if dfCrew.loc[i,"FirstOfficer"]==1 else 0 for i in I}
```

```
# print(NextFSameD[190])
# print(list(set(FbyD[13+1])&set(NextF[191])))
# assert False

# create a gurobi model
model = grb.Model("model_task2")
model.setParam("TimeLimit", 21600)
model.setParam("MIPGap", 0.001)
# 决策变量 X_i,j,r==1 说明 员工i以角色r身份登上飞机j
X = np.empty((len(I),len(VirF),len(R)),dtype=object)
for i in I:
    for j in VirF:
        for r in R:
            X[i,j,r] = model.addVar(vtype=grb.GRB.BINARY)

## 决策变量 Y_i_j_j'==1 对员工i而言, 航班j'是他乘坐航班j后的下一个航班
Y = np.empty((len(I),len(VirF),len(VirF)),dtype=object)
for i in I:
    for j in NotLastF: #包括-1不包括-2
        for jj in NextF[j]:
            Y[i,j,jj] = model.addVar(vtype=grb.GRB.BINARY)


i_captain = dfCrew.loc[(dfCrew["Captain"]==1)&(dfCrew["FirstOfficer"]==0),\
    "EmpNoInt"].idxmin()
i_officer = dfCrew.loc[(dfCrew["Captain"]==0)&(dfCrew["FirstOfficer"]==1),\
    "EmpNoInt"].idxmin()

# 虚拟航班约束X
for i in I:
    if i==i_captain:
        model.addConstr((X[i,-1,0]==1))
        model.addConstr((X[i,-2,0]==1))
    elif i==i_officer:
        model.addConstr((X[i,-1,1]==1))
        model.addConstr((X[i,-2,1]==1))
    else:
        model.addConstr((X[i,-1,2]==1))
        model.addConstr((X[i,-2,2]==1))



# 约束: 员工i同一航班不能身兼多职
for i in I:
    for j in VirF:
        model.addConstr((grb.quicksum(X[i,j,r] for r in R)<=1))

# 约束: 机长资格确认 r=0
```

```
for i in I:
    for j in VirF:
        model.addConstr((X[i,j,0]<=EpsC[i]))

# 约束: 副机长资格确认 r=1
for i in I:
    for j in VirF:
        model.addConstr((X[(i,j,1)]<=EpsF[i]))

# 约束: C1F1 和 约束: C=F 一个机长配一个副机长或者都为0 和 约束: 任意航班deadhead人数不超过M
for j in VirF:
    model.addConstr((grb.quicksum(X[i,j,0]*EpsC[i] for i in I)<=1))
    model.addConstr((grb.quicksum(X[i,j,1]*EpsF[i] for i in I)<=1))
    model.addConstr((grb.quicksum(X[i,j,0]*EpsC[i] for i in I) ==\
         grb.quicksum(X[i,j,1]*EpsF[i] for i in I) )) # =0 or =1
    # model.addConstr((grb.quicksum(X[i,j,2] for i in I) <= MaxDH))

# 约束: 航班取消, 员工不执行该项工作
for i in I:
    for j in VirF:
        model.addConstr((grb.quicksum(X[i,j,r] for r in R) <=\
             grb.quicksum(X[ii,j,0]*EpsC[ii] for ii in I)))


# Y[(i,j,j')]本质约束
for i in I:
    for j in NotLastF:
        model.addConstr((grb.quicksum(Y[i,j,jj] for jj in NextF[j]) <= 1))
for i in I:
    for jj in NotFirstF:
        model.addConstr((grb.quicksum(Y[i,j,jj] for j in PrevF[jj]) <= 1))


# 约束: 员工i从他基地出发的次数等于回到基地的次数
A_except_base = {i:A for i in I}# {i:list(set(A)-set([B[i]])) for i in I}
for i in I:
    model.addConstr((grb.quicksum(X[i,j,r]*f[j,B[i],n] for r in R\
         for n in A_except_base[i] for j in VirF) \
        == grb.quicksum(X[i,j,r]*f[j,m,B[i]] for r in R\
             for m in A_except_base[i] for j in VirF) ))

# 约束: 员工i的航班前后连续性 Y[i,j,j']
for k in A:
    for i in I:
        for j in NotLastF: # j包括-1不包括-2
            for jj in NextF[j]: # jj不包括-1包括-2
                model.addConstr((grb.quicksum(X[i,jj,r]*f[jj,k,n]\
                     for r in R for n in A) -\
                          grb.quicksum(X[i,j,r]*f[j,m,k]\
```

```
                                    for r in R for m in A) +\
                                L*(3-grb.quicksum(X[i,j,r] for r in R)\
                                    - grb.quicksum(X[i,jj,r]for r in R)\
                                        -Y[i,j,jj]) >= 0 ))
            model.addConstr((grb.quicksum(X[i,jj,r]*f[jj,k,n]\
                for r in R for n in A) -\
                        grb.quicksum(X[i,j,r]*f[j,m,k]\
                            for r in R for m in A) -\
                        L*(3-grb.quicksum(X[i,j,r] \
                            for r in R) - grb.quicksum(X[i,jj,r]\
                                for r in R) -Y[i,j,jj]) <= 0 ))
# 约束: XY的关系
for i in I:
    for j in NotLastF: #F1
        for jj in NextF[j]:
            model.addConstr((L*(1-Y[i,j,jj]) + grb.quicksum(X[i,j,r]\
                for r in R) + grb.quicksum(X[i,jj,r] for r in R) >= 2))



# 约束: 连接时间不小于MinCT分钟
for i in I:
    for j in NotLastF: # 包括-1
        for jj in NextF[j]: # jj不包括-1
            model.addConstr((DpT[jj]-ArT[j]+(1-Y[i,j,jj])*L >= MinCT ))

for i in I:
    for j in NotLastF: #F1
        model.addConstr((L*(1-grb.quicksum(X[i,j,r] for r in R))\
            +grb.quicksum(Y[i,j,jj] for jj in NextF[j]) >= 1))
        model.addConstr((-L*(1-grb.quicksum(X[i,j,r] for r in R))\
            +grb.quicksum(Y[i,j,jj] for jj in NextF[j]) <= 1))

for i in I:
    for jj in NotFirstF: #F2
        model.addConstr((L*(1-grb.quicksum(X[i,jj,r] for r in R))\
            +grb.quicksum(Y[i,j,jj] for j in PrevF[jj]) >= 1))
        model.addConstr((-L* (1-grb.quicksum(X[i,jj,r] for r in R))\
             +grb.quicksum(Y[i,j,jj] for j in PrevF[jj]) <= 1))



## 问题2新增决策变量
# 机组人员i在第k天的的执勤时间
DT = np.empty((len(I),len(D)),dtype=object)
for i in I:
```

```
    for k in D:
        DT[i,k] = model.addVar(lb=0.0, vtype=grb.GRB.CONTINUOUS)

# 机组人员的执勤时间的最大值,最小值
DTmax = model.addVar(lb=0.0, vtype=grb.GRB.CONTINUOUS)
DTmin = model.addVar(lb=0.0, vtype=grb.GRB.CONTINUOUS)



## 问题2新增约束

# 单次执勤飞行时长约束
for i in I:
    for k in D:
        model.addConstr((grb.quicksum(X[i,jk,r]*(ArT[jk]-DpT[jk]))\
             for r in R for jk in FbyD[k]) <= MaxBlk))

# 一次执勤时长约束
for i in I:
    for k in D:
        for jk in NotLastFSameD[k]:
            for jjk in NextFSameD[jk]:
                model.addConstr((grb.quicksum(X[i,jjk,r] for r in R)*ArT[jjk]\
                     - grb.quicksum(X[i,jk,r] for r in R)*DpT[jk]\
                    - L * (2-grb.quicksum(X[i,jjk,r] for r in R)\
                        -grb.quicksum(X[i,jk,r] for r in R)) <= MaxDP))
                model.addConstr((grb.quicksum(X[i,jjk,r] for r in R)*ArT[jjk]\
                     - grb.quicksum(X[i,jk,r] for r in R)*DpT[jk]\
                    - L * (2-grb.quicksum(X[i,jjk,r] for r in R)\
                        -grb.quicksum(X[i,jk,r] for r in R)) <= DT[i,k]))

# 相邻执勤间休息时间约束
for i in I:
    for k in D[:-1]:
        for jk in FbyD[k]:
            for jkk in list(set(FbyD[k+1])&set(NextF[jk])):
                model.addConstr(( grb.quicksum(X[i,jkk,r] for r in R)*DpT[jkk]\
                     - grb.quicksum(X[i,jk,r] for r in R)*ArT[jk]\
                    + L*(1-Y[i,jk,jkk]) >= MinRest))

for i in I:
    model.addConstr((grb.quicksum(DT[i,k] for k in D) <= DTmax))
    model.addConstr((grb.quicksum(DT[i,k] for k in D) >= DTmin))
```

```
# 1st objective: 最大化可执行航班
objMaxFlightNum = (grb.quicksum(X[i,j,0] for i in I for j in F))
# 2nd objective: 机组人员的总体执勤成本最低
objMinTotalDutyCost = (1.0/60.0 * grb.quicksum(DT[i,k]*DutyCost[i] \
    for k in D for i in I))

# 4th objective: 最小化总体deadhead次数
objMinDeadheadNum = (grb.quicksum(X[i,j,2] for i in I for j in F))

# 5th: 机组人员之间的执勤时长尽可能平衡
objMinDutyTimeRange = (DTmax–DTmin)



# 7th objective: 最小化替补资格
objMinBenchNum = (grb.quicksum(X[i,j,1]*EpsC[i]*EpsF[i] for i in I for j in F))




obj1UB = len(F)
obj2UB = len(D)*MaxDP*dfCrew["DutyCostPerHour"].sum()/60
obj4UB = 0.5*len(F)
obj5UB = len(D)*MaxDP*dfCrew["DutyCostPerHour"].max()/60
obj7UB = 0.5*len(F)
# 目标函数
w1 = 1E5/obj1UB
w2 = 1E3/obj2UB
w4 = 10/obj4UB
w5 = 1/obj5UB
w7 = 1E-3/obj7UB
objMinCost = (-w1*objMaxFlightNum + w2*objMinTotalDutyCost + w4*objMinDeadheadNum +\
    w5*objMinDutyTimeRange + w7*objMinBenchNum)
model.setObjective(objMinCost, grb.GRB.MINIMIZE)

model.presolve()

model.optimize()

# Save raw results to json file
result = {}
result['objMaxFlightNum'] = objMaxFlightNum.getValue()
result['objMinDeadheadNum'] = objMinDeadheadNum.getValue()
result['objMinBenchNum'] = objMinBenchNum.getValue()
result['objMinTotalDutyCost'] = objMinTotalDutyCost.getValue()
result['objMinDutyTimeRange'] = objMinDutyTimeRange.getValue()
```

```
result['X'] = {}
for i in I:
    for j in VirF:
        for r in R:
            result['X'][str("{},{},{}".format(i,j,r))] = X[i,j,r].x

result['Y'] = {}
for i in I:
    for j in NotLastF:
        for jj in NextF[j]:
            result['Y'][str("{},{},{}".format(i,j,jj))] = Y[i,j,jj].x

result['DT'] = {str("{},{}".format(i,k)):DT[i,k].x for i in I for k in D}
result['DTmax'] = DTmax.x
result['DTmin'] = DTmin.x

# dataName is "day_data" or "week_data"
with open("./raw_results_task2.json", 'w') as f:
    json.dump(result, f)
```

```
############################
####### TASK 3 ############
############################
import gurobipy as grb
import numpy as np
import pandas as pd
import os
import json



DATA = "A"
DATA_PATH = "./data/"
PREPARED_CREW_FILE = os.path.join(DATA_PATH, "prepared_crew_{}.csv".format(DATA))
PREPARED_FLIGHT_FILE = os.path.join(DATA_PATH, "prepared_flight_{}.csv".format(DATA))

dfCrew = pd.read_csv(PREPARED_CREW_FILE, sep=",")
dfFlight = pd.read_csv(PREPARED_FLIGHT_FILE, sep=",")

# 每个航班最大deadhead人数
# MaxDH = 5
# 大数
L = int(1E6)

MinCT = 40 # 最小连接时间
MaxBlk = 600 # 一次执勤飞行时长最多不超过
MaxDP = 720 # • 执勤时长最多不超过
MinRest = 660 # 相邻执勤之间的休息时间不少于
MaxTAFB = 14400 # 排班周期单个机组人员任务环总时长
MaxSuccOn = 4 #连续执勤天数不超过4天
MinVacDay = 2 #相邻两个任务环之间至少有2天休息



# 员工集合
I = list(np.sort(pd.unique(dfCrew["EmpNoInt"])))
# 航班集合
F = list(np.sort(pd.unique(dfFlight["FltIndex"])))

# 角色集合 0:主机长 1: 副机长 2: 搭乘
R = [0,1,2]
# 日期集合
D = list(np.sort(pd.unique(dfFlight["DptrDateInt"])))
# 机场集合
A = np.sort(np.unique(list(dfFlight["DptrStnInt"])+list(dfFlight["ArrvStnInt"])))
print("Airport set:", A)
# 员工基地map # B[i]
B = {i:dfCrew.loc[i,"BaseInt"] for i in I}
B_set = list(np.unique(list(B.values())))
BASE = 0
assert B_set == [0]
```

```
print("Base set:", B_set)

# 虚拟开始结束航班 + 航班集合
VirF = F + [-1,-2] # -1:源 -2: 汇
# print("Virtual F:", VirF)

# 对于j属于F, 具有上一个或下一个航班可能性的集合
NextF = {j:[] for j in F}
PrevF = {j:[] for j in F}
# 虚拟航班下一航班列表, 虚拟航班无上一航班
NextF[-1] = list(dfFlight.loc[dfFlight["DptrStnInt"]==BASE,"FltIndex"].index) + [-2]
PrevF[-1] = []
NextF[-2] = []
PrevF[-2] = list(dfFlight.loc[dfFlight["ArrvStnInt"]==BASE,"FltIndex"].index) + [-1]

# 不存在上一个和下一个航班的航班集合
LastF = [-2,] # VirF - F1
FirstF = [-1,] # VirF - F2



for j in F:
    arrvTimestamp_j = dfFlight.loc[j, "ArrvTimestamp"]
    arrvStnInt_j = dfFlight.loc[j, "ArrvStnInt"]
    possibleNextFlightFlag = (dfFlight["DptrStnInt"]==arrvStnInt_j)&\
        (dfFlight["DptrTimestamp"]>arrvTimestamp_j)
    if arrvStnInt_j in B_set:
        NextF[j].append(-2) #虚拟结束航班是任何目的地为base航班的下一个可能航班
    if np.any(possibleNextFlightFlag):
        NextF[j].extend(list(dfFlight.loc[possibleNextFlightFlag, "FltIndex"]))
    if not NextF[j]:
        print("no possible next flight for: ", j)
        LastF.append(j)
for j in F:
    dptrTimestamp_j = dfFlight.loc[j, "DptrTimestamp"]
    dptrStnInt_j = dfFlight.loc[j, "DptrStnInt"]
    possiblePrevFlightFlag = (dfFlight["ArrvStnInt"]==dptrStnInt_j)&\
        (dfFlight["ArrvTimestamp"]<dptrTimestamp_j)
    if (dptrStnInt_j in B_set):
        PrevF[j].append(-1) #虚拟开始航班是任何出发地为base航班的上一个可能航班
    if np.any(possiblePrevFlightFlag):
        PrevF[j].extend(list(dfFlight.loc[possiblePrevFlightFlag, "FltIndex"]))
    if not PrevF[j]:
        print("no possible previous flight for: ", j)
        FirstF.append(j)

print("no possible next flight for: ", LastF)
print("no possible previous flight for: ", FirstF)

# NotLastF = F - lastF
```

```
NotLastF = list(set(VirF)-set(LastF)) # F1
NotLastF.sort()
assert (-1 in NotLastF)
assert (-2 not in NotLastF)
# NotFirstF = F - FirstF
NotFirstF = list(set(VirF)-set(FirstF)) # F2
NotFirstF.sort()
assert(-1 not in NotFirstF)
assert(-2 in NotFirstF)



for j in F:
    assert NextF[j]
    assert PrevF[j]

jNextFj = []
for j in NotLastF:
    for jj in NextF[j]:
        jNextFj.append((j,jj))
jNextFj = set(jNextFj)

jPrevFj = []
for jj in NotFirstF:
    for j in PrevF[jj]:
        jPrevFj.append((j,jj))
jPrevFj = set(jPrevFj)

print((jPrevFj|jNextFj)-(jPrevFj&jNextFj))
print(len(jPrevFj|jNextFj))
print(len(jPrevFj&jNextFj))

assert len(jNextFj)==len(jPrevFj)
assert jNextFj == jPrevFj



# 航班起落机场 0-1参数
f = np.zeros((len(VirF),len(A),len(A)), dtype=np.int8)
for base in B_set:
    f[-1,base,base] = 1 # 虚拟航班 从base开到同一个base
    f[-2,base,base] = 1
for j in F:
    for m in A:
        for n in A:
            f[j,m,n] = 1 if (dfFlight.loc[j,"DptrStnInt"]==m)and(dfFlight.loc[j,\
                "ArrvStnInt"]==n) else 0

# 航班起飞绝对时间
DpT = {j:dfFlight.loc[j,"DptrTimestamp"] for j in F}
DpT[-1] = -MinCT # 虚拟开始航班出发时间
```

```
DpT[-2] = dfFlight["ArrvTimestamp"].max()+MinCT+10 # 虚拟结束航班出发时间
# 航班落地绝对时间
ArT = {j:dfFlight.loc[j,"ArrvTimestamp"] for j in F}
ArT[-1] = -MinCT # 虚拟开始航班到达时间
ArT[-2] = dfFlight["ArrvTimestamp"].max()+MinCT+10 # 虚拟结束航班到达时间

# 机长资格0-1
EpsC = {i:1 if dfCrew.loc[i,"Captain"]==1 else 0 for i in I}
# 副机长资格
EpsF = {i:1 if dfCrew.loc[i,"FirstOfficer"]==1 else 0 for i in I}



################################
## 问题2新增参数 ###############
################################

DutyCost = dfCrew["DutyCostPerHour"].to_numpy()

# 同一天出发的航班集合
FbyD = {d:list(dfFlight.loc[dfFlight["DptrDateInt"]==d,"FltIndex"]) for d in D}
# 航班j同一天出发的航班里接下来可能的下一班航班(不考虑虚拟航班)
NextFSameD = {j:[] for j in F}
for j in F:
    dptrDateInt_j = dfFlight.loc[j,"DptrDateInt"]
    arrvTimestamp_j = dfFlight.loc[j, "ArrvTimestamp"]
    possibleNextFlightFlag = (dfFlight["DptrTimestamp"]>arrvTimestamp_j)
    possibleNextFSameDayFlag = (dfFlight["DptrDateInt"]==dptrDateInt_j)&\
        possibleNextFlightFlag
    if np.any(possibleNextFSameDayFlag):
        NextFSameD[j] = list(dfFlight.loc[possibleNextFSameDayFlag, "FltIndex"])

LastFSameD = {k:[j for j in FbyD[k] if not NextFSameD[j]] for k in D}
 #同一天里不存在接续航班的航班集合
NotLastFSameD = {k:list(set(FbyD[k])-set(LastFSameD[k])) for k in D}
 #同一天里存在接续航班的航班集合

for k in D:
    assert set(LastFSameD[k])|set(NotLastFSameD[k]) == set(FbyD[k])



################################
## 问题3新增参数 ###############
################################
```

```
# 数据A中只有一个base
FfromB = list(dfFlight.loc[dfFlight["DptrStnInt"]==BASE,"FltIndex"])
FtoB = list(dfFlight.loc[dfFlight["ArrvStnInt"]==BASE,"FltIndex"])

FtoBafterJ = {j1:[] for j1 in FfromB}
for j1 in FfromB:
    arrvTimestamp_j1 = dfFlight.loc[j1,"ArrvTimestamp"]
    dfFlightFtoB = dfFlight.loc[FtoB,:]
    flightAfterJ1Flag = dfFlightFtoB["DptrTimestamp"]>arrvTimestamp_j1
    if np.any(flightAfterJ1Flag):
        FtoBafterJ[j1] = list(dfFlightFtoB.loc[flightAfterJ1Flag, "FltIndex"])
    else:
        print("No flight to BASE after j1:",j1)
        print("!!!!")
        assert False

FfromBbeforeJ = {j2:[] for j2 in FtoB}
for j2 in FtoB:
    dptrTimestamp_j2 = dfFlight.loc[j2,"DptrTimestamp"]
    dfFlightfromB = dfFlight.loc[FfromB,:]
    flightBeforeJ2Flag = dfFlightfromB["ArrvTimestamp"]<dptrTimestamp_j2
    if np.any(flightBeforeJ2Flag):
        FfromBbeforeJ[j2] = list(dfFlightfromB.loc[flightBeforeJ2Flag, "FltIndex"])
    else:
        print("No flight from BASE before j2:",j2)
        print("!!!!")
        assert False


PairingCost = dfCrew["ParingCostPerHour"].to_numpy()




################################
## 模型建立 ####################
################################

# create a gurobi model
model = grb.Model("model_task3")
model.setParam("TimeLimit", 36000)
model.setParam("MIPGap", 0.001)
```

```
# 决策变量 X_i,j,r==1 说明 员工i以角色r身份登上飞机j
X = np.empty((len(I),len(VirF),len(R)),dtype=object)
for i in I:
    for j in VirF:
        for r in R:
            X[i,j,r] = model.addVar(vtype=grb.GRB.BINARY)

## 决策变量 Y_i_j_j'==1 对员工i而言, 航班j'是他乘坐航班j后的下一个航班
Y = np.empty((len(I),len(VirF),len(VirF)),dtype=object)
for i in I:
    for j in NotLastF: #包括-1不包括-2
        for jj in NextF[j]:
            Y[i,j,jj] = model.addVar(vtype=grb.GRB.BINARY)


i_captain = dfCrew.loc[(dfCrew["Captain"]==1)&(dfCrew["FirstOfficer"]==0),\
    "EmpNoInt"].idxmin()
i_officer = dfCrew.loc[(dfCrew["Captain"]==0)&(dfCrew["FirstOfficer"]==1),\
    "EmpNoInt"].idxmin()

# 虚拟航班约束X
for i in I:
    if i==i_captain:
        model.addConstr((X[i,-1,0]==1))
        model.addConstr((X[i,-2,0]==1))
    elif i==i_officer:
        model.addConstr((X[i,-1,1]==1))
        model.addConstr((X[i,-2,1]==1))
    else:
        model.addConstr((X[i,-1,2]==1))
        model.addConstr((X[i,-2,2]==1))



# 约束: 员工i同一航班不能身兼多职
for i in I:
    for j in VirF:
        model.addConstr((grb.quicksum(X[i,j,r] for r in R)<=1))

# 约束: 机长资格确认 r=0
for i in I:
    for j in VirF:
        model.addConstr((X[i,j,0]<=EpsC[i]))

# 约束: 副机长资格确认 r=1
for i in I:
    for j in VirF:
```

```
        model.addConstr((X[(i,j,1)]<=EpsF[i]))

# 约束: C1F1 和 约束: C=F 一个机长配一个副机长或者都为0 和
## 约束: 任意航班deadhead人数不超过MaxDH=5
for j in VirF:
    model.addConstr((grb.quicksum(X[i,j,0]*EpsC[i] for i in I)<=1))
    model.addConstr((grb.quicksum(X[i,j,1]*EpsF[i] for i in I)<=1))
    model.addConstr((grb.quicksum(X[i,j,0]*EpsC[i] for i in I) ==\
        grb.quicksum(X[i,j,1]*EpsF[i] for i in I) )) # =0 or =1
    # model.addConstr((grb.quicksum(X[i,j,2] for i in I) <= MaxDH))

# 约束: 航班取消, 员工不执行该项工作
for i in I:
    for j in VirF:
        model.addConstr((grb.quicksum(X[i,j,r] for r in R)\
            <= grb.quicksum(X[ii,j,0]*EpsC[ii] for ii in I)))


# Y[(i,j,j')]本质约束 冗余约束
for i in I:
    for j in NotLastF:
        model.addConstr((grb.quicksum(Y[i,j,jj] for jj in NextF[j]) <= 1))
for i in I:
    for jj in NotFirstF:
        model.addConstr((grb.quicksum(Y[i,j,jj] for j in PrevF[jj]) <= 1))


# 约束: 员工i从他基地出发的次数等于回到基地的次数 冗余约束
A_except_base = {i:A for i in I}# {i:list(set(A)-set([B[i]])) for i in I}
for i in I:
    model.addConstr((grb.quicksum(X[i,j,r]*f[j,B[i],n] \
        for r in R for n in A_except_base[i] for j in VirF) \
        == grb.quicksum(X[i,j,r]*f[j,m,B[i]]\
            for r in R for m in A_except_base[i] for j in VirF) ))

# 约束: 员工i的航班前后连续性 Y[i,j,j']
for k in A:
    for i in I:
        for j in NotLastF: # j包括-1不包括-2
            for jj in NextF[j]: # jj不包括-1包括-2
                model.addConstr((grb.quicksum(X[i,jj,r]*f[jj,k,n] \
                    for r in R for n in A) -\
                        grb.quicksum(X[i,j,r]*f[j,m,k]\
                            for r in R for m in A) +\
                        L*(3-grb.quicksum(X[i,j,r] for r in R) \
                            - grb.quicksum(X[i,jj,r]for r in R)\
                                -Y[i,j,jj]) >= 0 ))
                model.addConstr((grb.quicksum(X[i,jj,r]*f[jj,k,n]\
                    for r in R for n in A) -\
```

```
                                    grb.quicksum(X[i,j,r]*f[j,m,k]\
                                        for r in R for m in A) -\
                                    L*(3-grb.quicksum(X[i,j,r] for r in R) \
                                        - grb.quicksum(X[i,jj,r]for r in R)\
                                            -Y[i,j,jj]) <= 0 ))
# 约束: XY的关系
for i in I:
    for j in NotLastF: #F1
        for jj in NextF[j]:
            model.addConstr((L*(1-Y[i,j,jj]) + grb.quicksum(X[i,j,r]\
                for r in R) + grb.quicksum(X[i,jj,r] for r in R) >= 2))



# 约束: 连接时间不小于MinCT分钟
for i in I:
    for j in NotLastF: # 包括-1
        for jj in NextF[j]: # jj不包括-1
            model.addConstr((DpT[jj]-ArT[j]+(1-Y[i,j,jj])*L >= MinCT ))

for i in I:
    for j in NotLastF: #F1
        model.addConstr((L*(1-grb.quicksum(X[i,j,r] for r in R))\
            +grb.quicksum(Y[i,j,jj] for jj in NextF[j]) >= 1))
        model.addConstr((-L*(1-grb.quicksum(X[i,j,r] for r in R))\
            +grb.quicksum(Y[i,j,jj] for jj in NextF[j]) <= 1))

for i in I:
    for jj in NotFirstF: #F2
        model.addConstr((L*(1-grb.quicksum(X[i,jj,r] for r in R))\
            +grb.quicksum(Y[i,j,jj] for j in PrevF[jj]) >= 1))
        model.addConstr((-L* (1-grb.quicksum(X[i,jj,r] for r in R))\
             +grb.quicksum(Y[i,j,jj] for j in PrevF[jj]) <= 1))









################################
#### 问题2新增决策变量 ########
################################

# 机组人员i在第k天的的执勤时间
DT = np.empty((len(I),len(D)),dtype=object)
for i in I:
    for k in D:
        DT[i,k] = model.addVar(lb=0.0, vtype=grb.GRB.CONTINUOUS)

# 机组人员的执勤时间的最大值,最小值
```

```
DTmax = model.addVar(lb=0.0, vtype=grb.GRB.CONTINUOUS)
DTmin = model.addVar(lb=0.0, vtype=grb.GRB.CONTINUOUS)

# ###################################
# ##### 问题2新增约束 ################
# ###################################



# 单次执勤飞行时长约束
for i in I:
    for k in D:
        model.addConstr((grb.quicksum(X[i,jk,r]*(ArT[jk]-DpT[jk])\
             for r in R for jk in FbyD[k]) <= MaxBlk))

# 一次执勤时长约束
for i in I:
    for k in D:
        for jk in NotLastFSameD[k]:
            for jjk in NextFSameD[jk]:
                model.addConstr((grb.quicksum(X[i,jjk,r] for r in R)*ArT[jjk]\
                     - grb.quicksum(X[i,jk,r] for r in R)*DpT[jk]\
                    - L * (2-grb.quicksum(X[i,jjk,r] for r in R)-\
                        grb.quicksum(X[i,jk,r] for r in R)) <= MaxDP))
                model.addConstr((grb.quicksum(X[i,jjk,r] for r in R)*\
                    ArT[jjk] - grb.quicksum(X[i,jk,r] for r in R)*DpT[jk]\
                    - L * (2-grb.quicksum(X[i,jjk,r] for r in R)-\
                        grb.quicksum(X[i,jk,r] for r in R)) <= DT[i,k]))

# 相邻执勤间休息时间约束
for i in I:
    for k in D[:-1]:
        for jk in FbyD[k]:
            for jkk in list(set(FbyD[k+1])&set(NextF[jk])):
                model.addConstr(( grb.quicksum(X[i,jkk,r] for r in R)\
                    *DpT[jkk] - grb.quicksum(X[i,jk,r] for r in R)*ArT[jk]\
                    + L*(1-Y[i,jk,jkk]) >= MinRest))

for i in I:
    model.addConstr((grb.quicksum(DT[i,k] for k in D) <= DTmax))
    model.addConstr((grb.quicksum(DT[i,k] for k in D) >= DTmin))




################################
#### 问题3新增决策变量 ########
################################

# 员工i排班的任务环总时间
```

```
PT = np.empty((len(I),),dtype=object)
for i in I:
    PT[i] = model.addVar(lb=0.0, vtype=grb.GRB.CONTINUOUS)

PTmax = model.addVar(lb=0.0, vtype=grb.GRB.CONTINUOUS)
PTmin = model.addVar(lb=0.0, vtype=grb.GRB.CONTINUOUS)

OnDuty = np.empty((len(I),len(D)),dtype=object)
for i in I:
    for k in D:
        OnDuty[i,k] = model.addVar(vtype=grb.GRB.BINARY)

Z = np.empty((len(I),len(F),len(F)),dtype=object)
for i in I:
    for j1 in FfromB:
        for j2 in FtoB:
            Z[i,j1,j2] = model.addVar(vtype=grb.GRB.BINARY)




#####################################
##### 问题3新增约束 ################
#####################################

# PT[i]和Z[i,j1,j2]的关系约束
for i in I:
    model.addConstr((PT[i] == grb.quicksum((ArT[j2]-DpT[j1])*Z[i,j1,j2]\
         for j1 in FfromB for j2 in FtoBafterJ[j1])))



# 当天是否执勤与当天是否搭乘航班的关系
for i in I:
    for k in D:
        model.addConstr((OnDuty[i,k] <= grb.quicksum(X[i,jk,r]\
             for r in R for jk in FbyD[k])))
for i in I:
    for k in D:
        for jk in FbyD[k]:
            model.addConstr((OnDuty[i,k] >= \
                grb.quicksum(X[i,jk,r] for r in R)))

#  连续执勤天数约束
for i in I:
    for k in D[:-MaxSuccOn]:
        model.addConstr((grb.quicksum(OnDuty[i,k+t] \
            for t in range(MaxSuccOn+1)) <= MaxSuccOn))

# 每个任务环首尾航班必须被执行
```

```
for i in I:
    for j1 in FfromB:
        for j2 in FtoBafterJ[j1]:
            model.addConstr((grb.quicksum(X[i,j1,r] for r in R)\
                 + grb.quicksum(X[i,j2,r] for r in R)\
                + L*(1-Z[i,j1,j2]) >= 2))

jsrc = -1
jsink = -2
# 虚拟开始航班与第一个任务环的关系
for i in I:
    for j1 in FfromB:
        model.addConstr(( L * (1-Y[i,jsrc,j1]) +\
             grb.quicksum(Z[i,j1,j2] for j2 in FtoBafterJ[j1]) >= 1))
        model.addConstr((-L * (1-Y[i,jsrc,j1]) +\
             grb.quicksum(Z[i,j1,j2] for j2 in FtoBafterJ[j1]) <= 1))
# 虚拟结束航班与最后一个任务环的关系
for i in I:
    for j2 in FtoB:
        model.addConstr(( L * (1-Y[i,j2,jsink]) +\
             grb.quicksum(Z[i,j1,j2] for j1 in FfromBbeforeJ[j2]) >= 1))
        model.addConstr((-L * (1-Y[i,j2,jsink]) +\
             grb.quicksum(Z[i,j1,j2] for j1 in FfromBbeforeJ[j2]) <= 1))

# 任务环与员工i航班连续性Y[i,j,j']的关系
for i in I:
    for j1 in FfromB:
        for j2 in list(set(FtoBafterJ[j1])&set(NotLastF)):
            for j3 in list(set(NextF[j2])-set([jsink])): \
                # j2 to B and j3 in NextF[j2] -> j3 from B
                model.addConstr(( L * (2-Z[i,j1,j2]-Y[i,j2,j3]) +\
                     grb.quicksum(Z[i,j3,j4] for j4 in FtoBafterJ[j3]) >= 1))
                model.addConstr((-L * (2-Z[i,j1,j2]-Y[i,j2,j3]) +\
                     grb.quicksum(Z[i,j3,j4] for j4 in FtoBafterJ[j3]) <= 1))


# 单个任务环总时长约束
for i in I:
    for j1 in FfromB:
        for j2 in FtoBafterJ[j1]:
            model.addConstr((ArT[j2] - DpT[j1] - L * (1-Z[i,j1,j2]) <= MaxTAFB))
            # model.addConstr((DpT[j2] - ArT[j1] + L * (1-Z[i,j1,j2]) >= MinCT))



# 任务环之间休息最小时间约束 简化版
for i in I:
    for j1 in FfromB:
        for j2 in list(set(FtoBafterJ[j1])&set(NotLastF)):
```

```
            for jj1 in list(set(NextF[j2])-set([jsink])): \
                # j2 to B and jj1 in NextF[j2] -> jj1 from B
                model.addConstr((DpT[jj1] - ArT[j2] + L *\
                    (2 - Z[i,j1,j2] - Y[i,j2,jj1]) >= MinVacDay))



# PTmax PTmin
for i in I:
    model.addConstr((PT[i] <= PTmax))
    model.addConstr((PT[i] >= PTmin))




# 1st objective: 最大化可执行航班
objMaxFlightNum = (grb.quicksum(X[i,j,0] for i in I for j in F))
# 2nd objective: 机组人员的总体执勤成本最低
objMinTotalDutyCost = (1.0/60.0 * grb.quicksum(DT[i,k]*DutyCost[i]\
     for k in D for i in I))

# 3rd objective: 机组人员的总体任务环成本最低
objMinTotalPairingCost = (1.0/60.0 * grb.quicksum(PT[i]*\
    PairingCost[i] for i in I))

# 4th objective: 最小化总体deadhead次数
objMinDeadheadNum = (grb.quicksum(X[i,j,2] for i in I for j in F))

# 5th: 机组人员之间的执勤时长尽可能平衡
objMinDutyTimeRange = (DTmax–DTmin)

# 6th: 机组人员之间的任务环时长尽可能平衡
objMinPairingTimeRange = (PTmax–PTmin)

# 7th objective: 最小化替补资格
objMinBenchNum = (grb.quicksum(X[i,j,1]*EpsC[i]*EpsF[i] \
    for i in I for j in F))



obj1UB = len(F)
obj2UB = len(D)*MaxDP*dfCrew["DutyCostPerHour"].sum()/60
obj3UB = len(D)*dfCrew["ParingCostPerHour"].sum()
obj4UB = 0.5*len(F)
obj5UB = len(D)*MaxDP*dfCrew["DutyCostPerHour"].max()/60
obj6UB = len(D)*dfCrew["ParingCostPerHour"].max()
obj7UB = 0.5*len(F)
# 目标函数
w1 = 1E7/obj1UB
w2 = 5E4/obj2UB
w3 = 1E4/obj3UB
```

```
w4 = 1E3/obj4UB
w5 = 5E2/obj5UB
w6 = 1E2/obj6UB
w7 = 1/obj7UB

objMinCost = (-w1*objMaxFlightNum + w2*objMinTotalDutyCost +\
      w3*objMinTotalPairingCost + w4*objMinDeadheadNum +\
     w5*objMinDutyTimeRange + w6*objMinPairingTimeRange +\
          w7*objMinBenchNum)
model.setObjective(objMinCost, grb.GRB.MINIMIZE)

# model.presolve()

model.optimize()

# Save raw results to json file
result = {}
result['objMaxFlightNum'] = objMaxFlightNum.getValue()
result['objMinDeadheadNum'] = objMinDeadheadNum.getValue()
result['objMinBenchNum'] = objMinBenchNum.getValue()
result['objMinTotalDutyCost'] = objMinTotalDutyCost.getValue()
result['objMinTotalPairingCost'] = objMinTotalPairingCost.getValue()
result['objMinDutyTimeRange'] = objMinDutyTimeRange.getValue()
result['objMinPairingTimeRange'] = objMinPairingTimeRange.getValue()

result['X'] = {}
for i in I:
    for j in VirF:
        for r in R:
            result['X'][str("{},{},{}".format(i,j,r))] = X[i,j,r].x

result['Y'] = {}
for i in I:
    for j in NotLastF:
        for jj in NextF[j]:
            result['Y'][str("{},{},{}".format(i,j,jj))] = Y[i,j,jj].x

result['DT'] = {str("{},{}".format(i,k)):DT[i,k].x for i in I for k in D}
result['DTmax'] = DTmax.x
result['DTmin'] = DTmin.x



##########################################
##### Task 3 需要记录Z变量 ################
##########################################
result['Z'] = {}
for i in I:
    for j1 in FfromB:
        for j2 in FtoBafterJ[j1]:
```

```
            result['Z'][str("{},{},{}".format(i,j1,j2))] = Z[i,j1,j2].x

result['PT'] = {str("{}".format(i)):PT[i].x for i in I}
result['PTmax'] = PTmax.x
result['PTmin'] = PTmin.x

result['OnDuty'] = {}
for i in I:
    for k in D:
        result['OnDuty'][str("{},{}".format(i,k))] = OnDuty[i,k].x

with open("./raw_results_task3.json", 'w') as f:
    json.dump(result, f)
```

```
########################################
##### Task 1 近似算法      ###############
########################################
import numpy as np
import pandas as pd
import os
import time

start = time.time()

DATA = "B"

OUTPUT_DIR = "./data/"
df_crew_A = pd.read_csv(OUTPUT_DIR+"prepared_crew_A.csv", sep=",")
df_flight_A = pd.read_csv(OUTPUT_DIR+"prepared_flight_A.csv", sep=",")
df_crew_B = pd.read_csv(OUTPUT_DIR+"prepared_crew_B.csv", sep=",")
df_flight_B = pd.read_csv(OUTPUT_DIR+"prepared_flight_B.csv", sep=",")


def generateCrewPairs(dfCrew):
    baseList = list(pd.unique(dfCrew["BaseInt"]))
    crewPairs = {base: [] for base in baseList}
    maxFeasiblePairingByBase = {base: 0 for base in baseList}
    for base in baseList:
        baseFlag = dfCrew["BaseInt"] == base
        captainCount = dfCrew.loc[baseFlag, "Captain"].sum()
        officerCount = dfCrew.loc[(
            dfCrew["Captain"] == 0) & baseFlag, "FirstOfficer"].sum()
        benchCount = dfCrew.loc[(dfCrew["FirstOfficer"] == 1)
                                & baseFlag, "Captain"].sum()
        if captainCount <= officerCount:
            maxFeasiblePairingByBase[base] = captainCount
        else:
            maxFeasiblePairingByBase[base] = min(
                (captainCount - officerCount)//2, benchCount) + officerCount
        print("Base Int:", base)
        print("Captain count(include bench):", captainCount)
        print("First Officer(only) count:", officerCount)
        print("Bench count:", benchCount)
        print("maxFeasiblePairingCount:", maxFeasiblePairingByBase[base])

        captainOnlyFlag = (dfCrew["Captain"] == 1) & (
            dfCrew["FirstOfficer"] == 0) & baseFlag
        officerOnlyFlag = (dfCrew["Captain"] == 0) & (
            dfCrew["FirstOfficer"] == 1) & baseFlag
        benchOnlyFlag = (dfCrew["Captain"] == 1) & (
            dfCrew["FirstOfficer"] == 1) & baseFlag

        captainLeftFlag = captainOnlyFlag & (~dfCrew["Pre-Assigned"])
```

```
        officerLeftFlag = officerOnlyFlag & (~dfCrew["Pre-Assigned"])
        # Exist captain(only) or officer(only)
        while np.any(captainLeftFlag) and np.any(officerLeftFlag):
            captainChosen = dfCrew.loc[captainLeftFlag, "EmpNoInt"].idxmin()
            dfCrew.loc[captainChosen, "Pre-Assigned"] = True
            officerChosen = dfCrew.loc[officerLeftFlag, "EmpNoInt"].idxmin()
            dfCrew.loc[officerChosen, "Pre-Assigned"] = True
            crewPairs[base].append((captainChosen, officerChosen))
            captainLeftFlag = captainOnlyFlag & (~dfCrew["Pre-Assigned"])
            officerLeftFlag = officerOnlyFlag & (~dfCrew["Pre-Assigned"])

        benchLeftFlag = benchOnlyFlag & (~dfCrew["Pre-Assigned"])
        # Exist captain(Only) and bench
        while np.any(captainLeftFlag) and np.any(benchLeftFlag):
            captainChosen = dfCrew.loc[captainLeftFlag, "EmpNoInt"].idxmin()
            dfCrew.loc[captainChosen, "Pre-Assigned"] = True
            officerChosen = dfCrew.loc[benchLeftFlag, "EmpNoInt"].idxmin()
            dfCrew.loc[officerChosen, "Pre-Assigned"] = True
            crewPairs[base].append((captainChosen, officerChosen))
            captainLeftFlag = captainOnlyFlag & (~dfCrew["Pre-Assigned"])
            benchLeftFlag = benchOnlyFlag & (~dfCrew["Pre-Assigned"])

        # Exist officer(only) and bench
        while np.any(officerLeftFlag) and np.any(benchLeftFlag):
            captainChosen = dfCrew.loc[benchLeftFlag, "EmpNoInt"].idxmin()
            dfCrew.loc[captainChosen, "Pre-Assigned"] = True
            officerChosen = dfCrew.loc[officerLeftFlag, "EmpNoInt"].idxmin()
            dfCrew.loc[officerChosen, "Pre-Assigned"] = True
            crewPairs[base].append((captainChosen, officerChosen))
            benchLeftFlag = benchOnlyFlag & (~dfCrew["Pre-Assigned"])
            officerLeftFlag = officerOnlyFlag & (~dfCrew["Pre-Assigned"])

        # if Still exist bench
        while np.any(benchLeftFlag):
            if(benchLeftFlag.sum() == 1):
                break
            captainChosen = dfCrew.loc[benchLeftFlag, "EmpNoInt"].idxmin()
            dfCrew.loc[captainChosen, "Pre-Assigned"] = True
            benchLeftFlag = benchOnlyFlag & (~dfCrew["Pre-Assigned"])
            officerChosen = dfCrew.loc[benchLeftFlag, "EmpNoInt"].idxmin()
            dfCrew.loc[officerChosen, "Pre-Assigned"] = True
            crewPairs[base].append((captainChosen, officerChosen))
            benchLeftFlag = benchOnlyFlag & (~dfCrew["Pre-Assigned"])

    return crewPairs


MinCT = 40
```

```
def finishOnePairing(dfCrew, dfFlight, pairing, base):
    deadheadCount = 0
    if not pairing:
        return [], deadheadCount
    lastFlight = pairing[-1]
    lastArrvTimestamp = dfFlight.loc[lastFlight, "ArrvTimestamp"]
    lastArrvStnInt = dfFlight.loc[lastFlight, "ArrvStnInt"]
    nextFlag = (~dfFlight["Assigned"]) & (dfFlight["DptrStnInt"] == \
        lastArrvStnInt) & (
        dfFlight["DptrTimestamp"]-lastArrvTimestamp >= MinCT)

    while np.any(nextFlag):
        nextFlight = dfFlight.loc[nextFlag, "DptrTimestamp"].idxmin()
        dfFlight.loc[nextFlight, "Assigned"] = True
        lastArrvTimestamp = dfFlight.loc[nextFlight, "ArrvTimestamp"]
        lastArrvStnInt = dfFlight.loc[nextFlight, "ArrvStnInt"]
        nextFlag = (~dfFlight["Assigned"]) & (dfFlight["DptrStnInt"] ==\
             lastArrvStnInt) & (
            dfFlight["DptrTimestamp"]-lastArrvTimestamp >= MinCT)
        pairing.append(nextFlight)

    if not np.any(nextFlag) and pairing:
        lastFlight = pairing[-1]
        terminateAtBaseFlag = dfFlight.loc[lastFlight, "ArrvStnInt"] == base
        # consider deadhead first
        if not terminateAtBaseFlag:
            existFlightEndAtBaseFlag = (dfFlight["ArrvStnInt"] == base) & (
                dfFlight["DptrStnInt"] == lastArrvStnInt) &\
                    (dfFlight["DptrTimestamp"]-lastArrvTimestamp >= MinCT)
            if np.any(existFlightEndAtBaseFlag):
                deadheadCount += 2
                terminateFlight = dfFlight.loc[existFlightEndAtBaseFlag,\
                    "DptrTimestamp"].idxmin(
                )
                dfFlight.loc[terminateFlight, "Assigned"] = True
                pairing.append(terminateFlight)
                terminateAtBaseFlag = True
        # Remove the last flights that do not end at Base
        while not terminateAtBaseFlag and pairing:
            dfFlight.loc[lastFlight, "Assigned"] == False
            pairing.pop()
            if pairing:
                lastFlight = pairing[-1]
                terminateAtBaseFlag = dfFlight.loc[lastFlight,
                                                   "ArrvStnInt"] == base
    return pairing, deadheadCount
```

```
def greedyPairing(dfCrew, dfFlight):

    dfCrew["Pre-Assigned"] = False
    dfCrew["Assigned"] = False
    dfFlight["Assigned"] = False
    dfFlight["CaptainEmpNoIndex"] = -1
    dfFlight["FirstOfficerEmpNoIndex"] = -1
    dfFlight["Feasible"] = True

    baseList = list(pd.unique(dfCrew["BaseInt"]))
    pairings = {base: [] for base in baseList}
    crewPairs = generateCrewPairs(dfCrew)
    for base, crewPairsByBase in crewPairs.items():
        print("Base Int:{}, crew pairs count:{}".format(
            base, len(crewPairsByBase)))
    maxFeasiblePairingByBase = {
        base: len(crewPairs[base]) for base in baseList}

    pairingCountByBase = {base: 0 for base in baseList}
    deadheadCount = 0
    for base in baseList:
        print("Prepare pairings for baseInt:", base)
        while pairingCountByBase[base] < maxFeasiblePairingByBase[base]:
            pairing = []
            startFlag = (~dfFlight["Assigned"]) & (
                dfFlight["DptrStnInt"] == base)
            if not np.any(startFlag):
                break
            startFlight = dfFlight.loc[startFlag, "DptrTimestamp"].idxmin()
            dfFlight.loc[startFlight, "Assigned"] = True
            pairing.append(startFlight)
            pairing, deadhead_count_delta = finishOnePairing(
                dfCrew, dfFlight, pairing, base)
            deadheadCount += deadhead_count_delta
            # print(pairing)
            pairingCountByBase[base] += 1
            pairings[base].append(pairing)

    for base in baseList:
        # deal with flights left
        # 存在改进空间：根据剩余pairing的长度进行二次选择
        while pairingCountByBase[base] < maxFeasiblePairingByBase[base]:
            pairing = []
            leftFlag = (~dfFlight["Assigned"]) & dfFlight["Feasible"]
            if not np.any(leftFlag):
                break
            nextStartFlight = dfFlight.loc[leftFlag, "DptrTimestamp"].idxmin()
            nextStartFlightDptrStn = dfFlight.loc[nextStartFlight, "DptrStnInt"]
            nextStartFlightDptrTimestamp = dfFlight.loc[nextStartFlight,
```

```
                                                    "DptrTimestamp"]
        # deadhead search for startFlight
        startFlag = (dfFlight["Assigned"]) & (dfFlight["DptrStnInt"] == base) & (
            dfFlight["ArrvStnInt"] == nextStartFlightDptrStn) & \
                (nextStartFlightDptrTimestamp-dfFlight["ArrvTimestamp"] >= MinCT)
        if np.any(startFlag):
            startFlight = dfFlight.loc[startFlag, "ArrvTimestamp"].idxmax()
            pairing.append(startFlight)
            pairing.append(nextStartFlight)
            deadheadCount += 2
            dfFlight.loc[nextStartFlight, "Assigned"] = True
            pairing, deadhead_count_delta = finishOnePairing(
                dfCrew, dfFlight, pairing, base)
            deadheadCount += deadhead_count_delta
            pairingCountByBase[base] += 1
            pairings[base].append(pairing)

        else:
            dfFlight.loc[nextStartFlight, "Feasible"] = False

X = {}  # X = {(fltIndex, crewIndex): roleIndex
# Assign pairing to pre-assigned crew pair
for base in baseList:
    pairingsByBase = pairings[base]
    # 第一对员工负责第一个 pairing
    crewPairsByBase = crewPairs[base][:len(pairingsByBase)]

    # 第一遍倒序循环中 deadhead的员工也被认为是机长或副机长，但可以确定航班真实机长
    for pairing, crewPair in zip(reversed(pairingsByBase),\
         reversed(crewPairsByBase)):
        # 由于deadhead 最后添加
        dfFlight.loc[pairing, "CaptainEmpNoIndex"] = crewPair[0]
        dfFlight.loc[pairing, "FirstOfficerEmpNoIndex"] = crewPair[1]
        dfCrew.loc[crewPair, "Assigned"] = True
        # captain:0 firstOfficer:1 deadhead:2
        for fltIndex in pairing:
            # 将在第二遍循环中修改
            X[(fltIndex, crewPair[0])] = 0
            X[(fltIndex, crewPair[1])] = 1

    # 寻找每一个pairing中真实机长不是对应负责员工的，说明这个员工是deadhead
    for pairing, crewPair in zip(reversed(pairingsByBase), \
        reversed(crewPairsByBase)):
        deadheadFlightFlag = dfFlight.loc[pairing,
                                          "CaptainEmpNoIndex"] != crewPair[0]
        dhFlights = dfFlight.loc[pairing,
                                 :].loc[deadheadFlightFlag, :].index
        if np.any(dhFlights):
            for fltIndex in list(dhFlights):
```

```
                    X[(fltIndex, crewPair[0])] = 2
                    X[(fltIndex, crewPair[1])] = 2

    print("GreedyPairing finished.")
    return pairings, deadheadCount, dfCrew, dfFlight, X


_dfFlight = df_flight_A if DATA == "A" else df_flight_B
_dfCrew = df_crew_A if DATA == "A" else df_crew_B

dfCrew = _dfCrew.loc[:, ["EmpNoInt", "Captain",
                         "FirstOfficer", "Deadhead", "BaseInt"]]
dfCrew["Pre-Assigned"] = False
dfCrew["Assigned"] = False

dfFlight = _dfFlight.loc[:, ["FltIndex", "DptrTimestamp",
                             "DptrStnInt", "ArrvTimestamp", "ArrvStnInt"]]
dfFlight["Assigned"] = False
dfFlight["CaptainEmpNoIndex"] = -1
dfFlight["FirstOfficerEmpNoIndex"] = -1
dfFlight["Feasible"] = True

# X = {(fltIndex, crewIndex): roleIndex}
pairings, deadheadCount, dfCrew, dfFlight, X = greedyPairing(dfCrew, dfFlight)


print("Assigned crew pair count:",
      dfCrew.loc[dfCrew["Assigned"], "EmpNoInt"].count()//2)
print("pairing count:", np.sum([len(pairingsByBase)
      for pairingsByBase in pairings.values()]))
flightArrangedCount = np.sum([np.sum(
    [len(pairing) for pairing in pairingsByBase]) for pairingsByBase\
        in pairings.values()])
print("Flight count:", dfFlight["FltIndex"].count())
deadheadCount = np.sum([1 for role in X.values() if role == 2])
print("Flight left count:", dfFlight["FltIndex"].count(
)-np.sum(dfFlight["Assigned"] == True))
print("Deadhead Count:", deadheadCount)


OUTPUT_DIR = "./task1/" + DATA + "_heuristic/"

dfCrewFull = pd.merge(_dfCrew, dfCrew, how="inner", on=[
                      "EmpNoInt", "Captain", "FirstOfficer", "Deadhead",
                       "BaseInt"])
dfFlightFull = pd.merge(_dfFlight, dfFlight, how="inner", on=[
                        "FltIndex", "DptrTimestamp", "DptrStnInt",
                        "ArrvTimestamp", "ArrvStnInt"])
dfFlightFull = dfFlightFull.drop(columns=["Feasible"])
```

```
def generateOutputs(outputDir, dfCrewFull, dfFlightFull, X):
    if not os.path.exists(outputDir):
        os.makedirs(outputDir)
    fileUncoveredFlights = outputDir+"UncoveredFlights.csv"
    fileCrewRosters = outputDir+"CrewRosters.csv"

    uncoveredFlightFlag = ~dfFlightFull["Assigned"]
    dfUncoveredFlight = None
    if np.any(uncoveredFlightFlag):
        dfUncoveredFlight = dfFlightFull.loc[uncoveredFlightFlag, [
            "FltNum", "DptrDate", "DptrTime", "DptrStn", "ArrvDate", "ArrvTime",
             "ArrvStn", "Comp"]]
        dfUncoveredFlight = dfUncoveredFlight.sort_values(
            by=["DptrDate", "DptrTime", "DptrStn", "ArrvStn"])
    else:
        dfUncoveredFlight = pd.DataFrame(columns=[
                                         "FltNum", "DptrDate", "DptrTime",
                                         "DptrStn", "ArrvDate", "ArrvTime",
                                         "ArrvStn", "Comp"])
    # print(dfUncoveredFlight)
    dfUncoveredFlight.to_csv(fileUncoveredFlights, sep=",", index=False)

    # write crew roster
    print(dfCrewFull.head())

    crewFlightRoles = np.array([(FltIndexEmpIndex[1], FltIndexEmpIndex[0], role)
                                for FltIndexEmpIndex, role in X.items()])
    dfCrewRosters = pd.DataFrame(crewFlightRoles, columns=[
                                 "EmpNoInt", "FltIndex", "Role"])
    dfCrewRosters = pd.merge(
        dfCrewRosters, dfFlightFull, how="left", on="FltIndex")
    dfCrewRosters = pd.merge(dfCrewRosters, dfCrewFull,
                             how="left", on="EmpNoInt")
    dfResult = dfCrewRosters.sort_values(
        by=["EmpNoInt", "DptrDateInt", "FltIndex"], ignore_index=True)

    dfResult["Task"] = ""
    for i in range(dfResult["Role"].count()):
        task = ""
        if (dfResult.loc[i, "Role"] == 0) and dfResult.loc[i, "Captain"]:
            task = "Captain"
        elif (dfResult.loc[i, "Role"] == 1) and (dfResult.loc[i, "Captain"] == 1):
            task = "AlternateFirstOfficer"
        elif (dfResult.loc[i, "Role"] == 1) and (dfResult.loc[i, "FirstOfficer"] == 1)\
             and (dfResult.loc[i, "Captain"] == 0):
            task = "FirstOfficer"
        elif (dfResult.loc[i, "Role"] == 2):
```

```
            task = "Deadhead"
        else:
            print("Error")
            assert False
        dfResult.loc[i, "Task"] = task

    dfCrewRosters = dfResult[["EmpNo", "Captain", "FirstOfficer", "Deadhead",
     "Task", "Base","FltNum", "DptrDate", "DptrTime", "DptrStn",
      "ArrvDate", "ArrvTime", "ArrvStn", "Comp"]]

    print(dfCrewRosters.head())
    print("Alternate count:", np.sum(
        dfCrewRosters["Task"] == "AlternateFirstOfficer"))
    print("Deadhead count:", np.sum(dfCrewRosters["Task"] == "Deadhead"))
    dfCrewRosters.to_csv(fileCrewRosters, index=False)

    return dfUncoveredFlight, dfCrewRosters


dfUncoveredFlight, dfCrewRosters = generateOutputs(
    OUTPUT_DIR, dfCrewFull, dfFlightFull, X)


print("Uncovered flight count:", dfUncoveredFlight["FltNum"].count())

endTime = time.time()-start
print("执行时间： {:.4f} s", endTime)
```

```
##########################################
##### Task 2 近似算法      ################
##########################################

import numpy as np
import pandas as pd
import os
import time
from copy import copy



DATA = "B"
TASK = 2
DATA_DIR = "./data/"
MinCT = 40  # 最小连接时间
MaxBlk = 600  # 一次执勤飞行时长最多不超过
MaxDP = 720  # •          执勤时长最多不超过
MinRest = 660  # 相邻执勤之间的休息时间不少于
MaxTAFB = 14400  # 排班周期单个机组人员任务环总时长
MaxSuccOn = 4  # 连续执勤天数不超过4天
MinVacDay = 2  # 相邻两个任务环之间至少有2天休息



startTime = time.time()
df_crew_A = pd.read_csv(DATA_DIR+"prepared_crew_A.csv", sep=",")
df_flight_A = pd.read_csv(DATA_DIR+"prepared_flight_A.csv", sep=",")
df_crew_B = pd.read_csv(DATA_DIR+"prepared_crew_B.csv", sep=",")
df_flight_B = pd.read_csv(DATA_DIR+"prepared_flight_B.csv", sep=",")

dfFlight = df_flight_A if DATA == "A" else df_flight_B
dfCrew = df_crew_A if DATA == "A" else df_crew_B
# 员工集合
I = list(np.sort(pd.unique(dfCrew["EmpNoInt"])))
# 航班集合
F = list(np.sort(pd.unique(dfFlight["FltIndex"])))
# 角色集合 0:主机长 1: 副机长 2: 搭乘
R = [0, 1, 2]
# 日期集合
D = list(np.sort(pd.unique(dfFlight["DptrDateInt"])))
# 机场集合
A = np.sort(
    np.unique(list(dfFlight["DptrStnInt"])+list(dfFlight["ArrvStnInt"])))
print("Airport set:", A)
# 员工基地map # B[i]
B = {i: dfCrew.loc[i, "BaseInt"] for i in I}

# 同一天出发的航班集合
FbyD = {
    k: list(dfFlight.loc[dfFlight["DptrDateInt"] == k, "FltIndex"]) for k in D}
```

```
# 航班j同一天出发的航班里接下来可能的下一班航班(不考虑虚拟航班)
NextFSameD = {j: [] for j in F}
for j in F:
    dptrDateInt_j = dfFlight.loc[j, "DptrDateInt"]
    arrvTimestamp_j = dfFlight.loc[j, "ArrvTimestamp"]
    arrvStnInt_j = dfFlight.loc[j, "ArrvStnInt"]
    possibleNextFlightFlag = (dfFlight["DptrStnInt"] == arrvStnInt_j) & (
        dfFlight["DptrTimestamp"] > arrvTimestamp_j)
    possibleNextFSameDayFlag = (
        dfFlight["DptrDateInt"] == dptrDateInt_j) & possibleNextFlightFlag
    if np.any(possibleNextFSameDayFlag):
        NextFSameD[j] = list(
            dfFlight.loc[possibleNextFSameDayFlag, "FltIndex"])

LastFSameD = {k: [j for j in FbyD[k] if not NextFSameD[j]]
              for k in D}  # 同一天里不存在接续航班的航班集合
NotLastFSameD = {k: list(set(FbyD[k])-set(LastFSameD[k]))
                 for k in D}  # 同一天里存在接续航班的航班集合

for k in D:
    assert set(LastFSameD[k]) | set(NotLastFSameD[k]) == set(FbyD[k])


def generateCrewPairsWithRole(dfCrew):
    dfCrew["Pre-Assigned"] = False
    dfCrew["Assigned"] = False
    baseList = list(pd.unique(dfCrew["BaseInt"]))
    crewPairs = {base: [] for base in baseList}
    maxFeasiblePairingByBase = {base: 0 for base in baseList}
    for base in baseList:
        baseFlag = dfCrew["BaseInt"] == base
        captainCount = dfCrew.loc[baseFlag, "Captain"].sum()
        officerCount = dfCrew.loc[(
            dfCrew["Captain"] == 0) & baseFlag, "FirstOfficer"].sum()
        benchCount = dfCrew.loc[(dfCrew["FirstOfficer"] == 1)
                                & baseFlag, "Captain"].sum()
        if captainCount <= officerCount:
            maxFeasiblePairingByBase[base] = captainCount
        else:
            maxFeasiblePairingByBase[base] = min(
                (captainCount - officerCount)//2, benchCount) + officerCount
        print("Base Int:", base)
        print("Captain count(include bench):", captainCount)
        print("First Officer(only) count:", officerCount)
        print("Bench count:", benchCount)
        print("maxFeasiblePairingCount:", maxFeasiblePairingByBase[base])

        captainOnlyFlag = (dfCrew["Captain"] == 1) & (
            dfCrew["FirstOfficer"] == 0) & baseFlag
```

```
officerOnlyFlag = (dfCrew["Captain"] == 0) & (
    dfCrew["FirstOfficer"] == 1) & baseFlag
benchOnlyFlag = (dfCrew["Captain"] == 1) & (
    dfCrew["FirstOfficer"] == 1) & baseFlag

captainLeftFlag = captainOnlyFlag & (~dfCrew["Pre-Assigned"])
officerLeftFlag = officerOnlyFlag & (~dfCrew["Pre-Assigned"])
# Exist captain(only) or officer(only)
while np.any(captainLeftFlag) and np.any(officerLeftFlag):
    captainChosen = dfCrew.loc[captainLeftFlag, "EmpNoInt"].idxmin()
    dfCrew.loc[captainChosen, "Pre-Assigned"] = True
    officerChosen = dfCrew.loc[officerLeftFlag, "EmpNoInt"].idxmin()
    dfCrew.loc[officerChosen, "Pre-Assigned"] = True
    crewPairs[base].append((captainChosen, officerChosen, 0, 1))
    captainLeftFlag = captainOnlyFlag & (~dfCrew["Pre-Assigned"])
    officerLeftFlag = officerOnlyFlag & (~dfCrew["Pre-Assigned"])

benchLeftFlag = benchOnlyFlag & (~dfCrew["Pre-Assigned"])
# Exist captain(Only) and bench
while np.any(captainLeftFlag) and np.any(benchLeftFlag):
    captainChosen = dfCrew.loc[captainLeftFlag, "EmpNoInt"].idxmin()
    dfCrew.loc[captainChosen, "Pre-Assigned"] = True
    officerChosen = dfCrew.loc[benchLeftFlag, "EmpNoInt"].idxmin()
    dfCrew.loc[officerChosen, "Pre-Assigned"] = True
    # exist alternate FO
    crewPairs[base].append((captainChosen, officerChosen, 0, 2))
    captainLeftFlag = captainOnlyFlag & (~dfCrew["Pre-Assigned"])
    benchLeftFlag = benchOnlyFlag & (~dfCrew["Pre-Assigned"])

# Exist officer(only) and bench
while np.any(officerLeftFlag) and np.any(benchLeftFlag):
    captainChosen = dfCrew.loc[benchLeftFlag, "EmpNoInt"].idxmin()
    dfCrew.loc[captainChosen, "Pre-Assigned"] = True
    officerChosen = dfCrew.loc[officerLeftFlag, "EmpNoInt"].idxmin()
    dfCrew.loc[officerChosen, "Pre-Assigned"] = True
    crewPairs[base].append((captainChosen, officerChosen, 0, 1))
    benchLeftFlag = benchOnlyFlag & (~dfCrew["Pre-Assigned"])
    officerLeftFlag = officerOnlyFlag & (~dfCrew["Pre-Assigned"])

# if Still exist bench
while np.any(benchLeftFlag):
    if(benchLeftFlag.sum() == 1):
        break
    captainChosen = dfCrew.loc[benchLeftFlag, "EmpNoInt"].idxmin()
    dfCrew.loc[captainChosen, "Pre-Assigned"] = True
    benchLeftFlag = benchOnlyFlag & (~dfCrew["Pre-Assigned"])
    officerChosen = dfCrew.loc[benchLeftFlag, "EmpNoInt"].idxmin()
    dfCrew.loc[officerChosen, "Pre-Assigned"] = True
    # exist alternate FO
```

```
            crewPairs[base].append((captainChosen, officerChosen, 0, 2))
            benchLeftFlag = benchOnlyFlag & (~dfCrew["Pre-Assigned"])

    return crewPairs


def createOneDuty(dayK, dfFlight, lastDutyEndTimestamp, startAirport,\
     base, lastDutyBackToBase=False):
    duty = []
    deadhead = []

    lastArrvTimestamp = copy(lastDutyEndTimestamp)
    lastArrvStnInt = copy(startAirport)
    connectionFlag = (dfFlight["DptrDateInt"] == dayK) & (dfFlight["DptrStnInt"]\
         == lastArrvStnInt) & (
        dfFlight["DptrTimestamp"]-lastArrvTimestamp >= MinCT)
    startFlag = (~dfFlight["Assigned"]) & connectionFlag
    if lastDutyEndTimestamp > 0:
        # Not first Duty; consider MinRest constrict
        startFlag = startFlag & (
            dfFlight["DptrTimestamp"]-lastArrvTimestamp >= MinRest)

    dutyTime = 0
    flightTime = 0
    # Normal case
    startFlight = None
    firstDeadheadOn = False
    if not np.any(startFlag):
        # No possible flight from this startPort -> deadhead
        # print("No possible flight from this start airport")
        connectionFlag = (dfFlight["DptrDateInt"] == dayK) & (dfFlight["DptrStnInt"]\
             == lastArrvStnInt) & (
            dfFlight["DptrTimestamp"]-lastArrvTimestamp >= MinCT)
        startFlag = (dfFlight["Assigned"]) & connectionFlag
        if lastDutyEndTimestamp > 0:
            startFlag = startFlag & (
                dfFlight["DptrTimestamp"]-lastArrvTimestamp >= MinRest)
        firstDeadheadOn = True
        if not np.any(startFlag):
            # No possible deadhead
            # print("No possible deadhead !!!")
            return [], [], lastDutyEndTimestamp, startAirport, dfFlight

    # 暂时不确定 nextFlight 是不是deadhead
    startFlight = dfFlight.loc[startFlag, "DptrTimestamp"].idxmin()
    # print(nextFlight)
    nextFlag = startFlag
    nextFlight = startFlight
    nextFlightTime = dfFlight.loc[nextFlight, "ArrvTimestamp"] - \
```

```
    dfFlight.loc[nextFlight, "DptrTimestamp"]
nextDutyTime = (dfFlight.loc[nextFlight, "ArrvTimestamp"] -
                dfFlight.loc[nextFlight, "DptrTimestamp"])
flightTime += nextFlightTime
dutyTime += nextDutyTime

while np.any(nextFlag) and dutyTime <= MaxDP and flightTime < MaxBlk:
    # print("DT:{},FT:{}".format(dutyTime,flightTime))
    if dfFlight.loc[nextFlight, "Assigned"]:
        deadhead.append(nextFlight)
    dfFlight.loc[nextFlight, "Assigned"] = True
    duty.append(nextFlight)

    lastArrvTimestamp = dfFlight.loc[nextFlight, "ArrvTimestamp"]
    lastArrvStnInt = dfFlight.loc[nextFlight, "ArrvStnInt"]

    connectionFlag = (dfFlight["DptrDateInt"] == dayK) &\
         (dfFlight["DptrStnInt"] == lastArrvStnInt) & (
        dfFlight["DptrTimestamp"]-lastArrvTimestamp >= MinCT)
    nextFlag = (~dfFlight["Assigned"]) & connectionFlag
    if np.any(nextFlag):
        nextFlight = dfFlight.loc[nextFlag, "DptrTimestamp"].idxmin()
        nextFlightTime = dfFlight.loc[nextFlight, "ArrvTimestamp"] - \
            dfFlight.loc[nextFlight, "DptrTimestamp"]
        flightTime += nextFlightTime
        nextDutyTime = (
            dfFlight.loc[nextFlight, "ArrvTimestamp"]-lastArrvTimestamp)
        dutyTime += nextDutyTime
# remove time caused by last not possible flight
if dutyTime > MaxDP or flightTime > MaxBlk:
    flightTime -= nextFlightTime
    dutyTime -= nextDutyTime

if len(duty) == 0:
    print("empty duty before check end deadhead")
    return [], [], lastDutyEndTimestamp, startAirport, dfFlight

# duty not on the last day
if not lastDutyBackToBase:
    # print("Avoid Surpassing flightTime or duty time ")
    return duty, deadhead, lastArrvTimestamp, lastArrvStnInt, dfFlight

# duty on the last day
lastFlight = duty[-1]
terminateAtBaseFlag = dfFlight.loc[lastFlight, "ArrvStnInt"] == base
if terminateAtBaseFlag:
    lastArrvTimestamp = dfFlight.loc[lastFlight, "ArrvTimestamp"]
    return duty, deadhead, lastArrvTimestamp, base, dfFlight
```

```
# consider deadhead
lastArrvStnInt = dfFlight.loc[lastFlight, "ArrvStnInt"]
lastArrvTimestamp = dfFlight.loc[lastFlight, "ArrvTimestamp"]
connectionFlag = (dfFlight["DptrDateInt"] == dayK) &\
     (dfFlight["DptrStnInt"] == lastArrvStnInt) & (
    dfFlight["DptrTimestamp"]-lastArrvTimestamp >= MinCT)
existDeadheadFlightEndAtBaseFlag = (dfFlight["Assigned"]) & (
    dfFlight["ArrvStnInt"] == base) & connectionFlag & \
        (dfFlight["ArrvTimestamp"]-lastDutyEndTimestamp <= MaxDP)
if np.any(existDeadheadFlightEndAtBaseFlag):
    terminateFlight = dfFlight.loc[existDeadheadFlightEndAtBaseFlag,\
         "DptrTimestamp"].idxmin(
    )
    duty.append(terminateFlight)
    terminateAtBaseFlag = True
    deadhead.append(terminateFlight)
    # print("find extra end deadhead")
    return duty, deadhead, dfFlight.loc[terminateFlight,\
         "ArrvTimestamp"], base, dfFlight

while not terminateAtBaseFlag and duty:
    # Remove the last flights that do not end at Base and search for deadhead
    if not(firstDeadheadOn and len(duty) == 1):
        dfFlight.loc[lastFlight, "Assigned"] == False
    duty.pop()
    if duty:
        lastFlight = duty[-1]
        terminateAtBaseFlag = dfFlight.loc[lastFlight,
                                           "ArrvStnInt"] == base
        lastArrvTimestamp = dfFlight.loc[lastFlight, "ArrvTimestamp"]
        lastArrvStnInt = dfFlight.loc[lastFlight, "ArrvStnInt"]
        if terminateAtBaseFlag:
            # print("Delete some flights and back to base")
            return duty, deadhead, lastArrvTimestamp, base, dfFlight
        else:
            connectionFlag = (dfFlight["DptrDateInt"] == dayK) & \
                (dfFlight["DptrStnInt"] == lastArrvStnInt) & (
                dfFlight["DptrTimestamp"]-lastArrvTimestamp >= MinCT)
            existFlightEndAtBaseFlag = (dfFlight["ArrvStnInt"] == base) \
                & connectionFlag & (
                dfFlight["ArrvTimestamp"]-lastDutyEndTimestamp <= MaxDP)
            if np.any(existFlightEndAtBaseFlag):
                terminateFlight = dfFlight.loc[existFlightEndAtBaseFlag,\
                     "DptrTimestamp"].idxmin(
                )
                if dfFlight.loc[terminateFlight, "Assigned"]:
                    deadhead.append(terminateFlight)
                duty.append(terminateFlight)
                dfFlight.loc[terminateFlight, "Assigned"] = True
```

```
                    terminateAtBaseFlag = True
                    # print("Delete some flights and find deadhead")
                    return duty, deadhead, dfFlight.loc[terminateFlight, \
                        "ArrvTimestamp"], base, dfFlight
                else:
                    continue

    connectionFlag = (dfFlight["DptrDateInt"] == dayK) &\
         (dfFlight["DptrStnInt"] == startAirport) & (
        dfFlight["DptrTimestamp"]-lastDutyEndTimestamp >= MinCT)
    existFlightEndAtBaseFlag = (dfFlight["ArrvStnInt"] == base) \
        & connectionFlag & (
        dfFlight["ArrvTimestamp"]-lastDutyEndTimestamp <= MaxDP) \
            & (dfFlight["DptrTimestamp"]-lastDutyEndTimestamp >= MinRest)
    if np.any(existFlightEndAtBaseFlag):
        terminateFlight = dfFlight.loc[existFlightEndAtBaseFlag, "DptrTimestamp"].idxmin(
        )
        if dfFlight.loc[terminateFlight, "Assigned"]:
            deadhead.append(terminateFlight)
        duty.append(terminateFlight)
        dfFlight.loc[terminateFlight, "Assigned"] = True
        terminateAtBaseFlag = True
        # print("Delete some flights and find deadhead")
        return duty, deadhead, dfFlight.loc[terminateFlight,\
             "ArrvTimestamp"], base, dfFlight
    # empty duty
    # print("No duty until the end function call")
    return [], [], lastDutyEndTimestamp, startAirport, dfFlight


# greedy by duty
crewPairsWithRoles = generateCrewPairsWithRole(
    dfCrew)  # base_i:[(capId, officerID, role1, role2)]
crewPairPrevDutyEndTimestamp = {}
crewPairCurLocation = {}
for base, crewPairRoleList in crewPairsWithRoles.items():
    for crewPair in crewPairRoleList:
        crewPairPrevDutyEndTimestamp[(crewPair[0], crewPair[1])] = 0
        crewPairCurLocation[(crewPair[0], crewPair[1])] = base

X = []
dfFlight["Assigned"] = False


##################
#### BFS ########
##################

# for dayK in D:
```

```
#     isLastDuty = False
#     if dayK == max(D):
#         isLastDuty = True
#     for base, crewPairRoleList in crewPairsWithRoles.items():
#         for crewPair in crewPairRoleList:
#             crew1, crew2, role1, role2 = crewPair
#             duty, deadheadList, dutyEndTimestamp, dutyEndAirport,\
#  dfFlight = createOneDuty(dayK, dfFlight,\
#                 crewPairPrevDutyEndTimestamp[(crew1,crew2)], \
# crewPairCurLocation[(crew1,crew2)], base, isLastDuty)
#             print(duty)
#             # dfFlight.loc[duty,"Assigned"] = True
#             crewPairPrevDutyEndTimestamp[(crewPair[0],crewPair[1])] = dutyEndTimestamp
#             crewPairCurLocation[(crewPair[0],crewPair[1])] = dutyEndAirport
#             for flight in list(set(duty)-set(deadheadList)):
#                 X.append((crew1,flight,role1))
#                 X.append((crew2,flight,role2))
#             for flight_ in deadheadList:
#                 X.append((crew1,flight_,-1))
#                 X.append((crew2,flight_,-1))


##################
#### DFS ########
##################

print(D)


for base, crewPairRoleList in crewPairsWithRoles.items():
    for crewPair in crewPairRoleList:
        # print(crewPair)
        crew1, crew2, role1, role2 = crewPair
        for dayK in D:
            isLastDuty = False
            if dayK == max(D):
                isLastDuty = True
            duty, deadheadList, dutyEndTimestamp, dutyEndAirport, \
                dfFlight = createOneDuty(dayK, dfFlight,\
                    crewPairPrevDutyEndTimestamp[(crew1, crew2)], \
                        crewPairCurLocation[(crew1, crew2)], base, isLastDuty)
            # print(duty)
            # print(deadheadList)
            assert set(deadheadList).issubset(set(duty))
            if not duty:
                continue

            crewPairPrevDutyEndTimestamp[(crew1, crew2)] = dutyEndTimestamp
            crewPairCurLocation[(crew1, crew2)] = dutyEndAirport
```

```
            for flight in list(set(duty)-set(deadheadList)):
                X.append((crew1, flight, role1))
                X.append((crew2, flight, role2))
            for _flight_ in deadheadList:
                X.append((crew1, _flight_, -1))
                X.append((crew2, _flight_, -1))
        print(crewPairCurLocation[(crew1, crew2)])

X = np.array(X)
fltCovered = np.unique(X[:, 1])
print("flight count:", dfFlight["Assigned"].sum())
print("flight count:", len(fltCovered))
print("captain count:", np.sum(X[:, 2] == 0))
print("officer count:", np.sum(X[:, 2] == 1))
print("alternate count:", np.sum(X[:, 2] == 2))
print("deadhead count:", np.sum(X[:, 2] == -1))

lastTime = time.time() - startTime
print("执行时间：{:.4f} s".format(lastTime))



##################################
###### 生成对应的数据 ############
##################################

notAssignedFlightIndexList = list(set(F)-set(np.unique(X[:, 1])))
outputDir = "./task{}/".format(TASK) + DATA + "_heuristic/" + "/"
if not os.path.exists(outputDir):
    os.makedirs(outputDir)
fileUncoveredFlights = outputDir + "UncoveredFlights.csv"
fileCrewRosters = outputDir + "CrewRosters.csv"

# Generate UncoveredFlights.csv
dfUncoveredFlight = dfFlight.loc[notAssignedFlightIndexList, :]
if dfUncoveredFlight.empty:
    print("All flights covered")
    dfUncoveredFlight = pd.DataFrame(columns=[
                                    "FltNum", "DptrDate", "DptrTime",
                                    "DptrStn", "ArrvDate", "ArrvTime",
                                     "ArrvStn", "Comp"])
else:
    # 存在未被执行的航班
    print("Find flights uncovered")
    dfUncoveredFlight = dfUncoveredFlight[[
        "FltNum", "DptrDate", "DptrTime", "DptrStn", "ArrvDate",
        "ArrvTime", "ArrvStn", "Comp"]]
    dfUncoveredFlight = dfUncoveredFlight.sort_values(
        by=["DptrDate", "DptrTime", "DptrStn", "ArrvStn"])
```

```
print("Uncovered Flights:")
print(dfUncoveredFlight)
dfUncoveredFlight.to_csv(fileUncoveredFlights, sep=",", index=False)



# Generate CrewRosters.csv
dfResult = pd.DataFrame(X, columns=["EmpNoInt", "FltIndex", "Role"])  # 包含-1,-2
dfResult = dfResult.loc[~((dfResult["FltIndex"] == -1)
                          | (dfResult["FltIndex"] == -2)), :]  # 不再包含-1,-2
dfResult = pd.merge(dfResult, dfCrew, how="left", on="EmpNoInt")
dfResult = pd.merge(dfResult, dfFlight, how="left", on="FltIndex")
dfResult = dfResult.sort_values(
    by=["EmpNoInt", "DptrDateInt", "FltIndex"], ignore_index=True)

dfResult["Task"] = ""
for i in range(dfResult["Role"].count()):
    task = ""
    if (dfResult.loc[i, "Role"] == 0):
        task = "Captain"
    elif (dfResult.loc[i, "Role"] == 2):
        task = "AlternateFirstOfficer"
    elif (dfResult.loc[i, "Role"] == 1):
        task = "FirstOfficer"
    elif (dfResult.loc[i, "Role"] == -1):
        task = "Deadhead"
    else:
        print("Error")
        assert False
    dfResult.loc[i, "Task"] = task

dfCrewRosters = dfResult[["EmpNo", "Captain", "FirstOfficer",
"Deadhead", "Task", "Base","FltNum", "DptrDate", "DptrTime",
 "DptrStn", "ArrvDate", "ArrvTime", "ArrvStn", "Comp"]]

print(dfCrewRosters.head())
dfCrewRosters.to_csv(fileCrewRosters, index=False)
```

```
########################################
##### Task 3 近似算法      ###############
########################################

import numpy as np
import pandas as pd
import os
import time
from copy import copy



DATA = "B"
TASK = 3
DATA_DIR = "./data/"
MinCT = 40  # 最小连接时间
MaxBlk = 600  # 一次执勤飞行时长最多不超过
MaxDP = 720  # •          执勤时长最多不超过
MinRest = 660  # 相邻执勤之间的休息时间不少于
MaxTAFB = 14400  # 排班周期单个机组人员任务环总时长
MaxSuccOn = 4  # 连续执勤天数不超过4天
MinVacDay = 2  # 相邻两个任务环之间至少有2天休息



startTime = time.time()
df_crew_A = pd.read_csv(DATA_DIR+"prepared_crew_A.csv", sep=",")
df_flight_A = pd.read_csv(DATA_DIR+"prepared_flight_A.csv", sep=",")
df_crew_B = pd.read_csv(DATA_DIR+"prepared_crew_B.csv", sep=",")
df_flight_B = pd.read_csv(DATA_DIR+"prepared_flight_B.csv", sep=",")

dfFlight = df_flight_A if DATA == "A" else df_flight_B
dfCrew = df_crew_A if DATA == "A" else df_crew_B
# 员工集合
I = list(np.sort(pd.unique(dfCrew["EmpNoInt"])))
# 航班集合
F = list(np.sort(pd.unique(dfFlight["FltIndex"])))
# 角色集合 0:主机长 1: 副机长 2: 搭乘
R = [0, 1, 2]
# 日期集合
D = list(np.sort(pd.unique(dfFlight["DptrDateInt"])))
# 机场集合
A = np.sort(
    np.unique(list(dfFlight["DptrStnInt"])+list(dfFlight["ArrvStnInt"])))
print("Airport set:", A)
# 员工基地map # B[i]
B = {i: dfCrew.loc[i, "BaseInt"] for i in I}

# 同一天出发的航班集合
FbyD = {
    k: list(dfFlight.loc[dfFlight["DptrDateInt"] == k, "FltIndex"]) for k in D}
```

```
# 航班j同一天出发的航班里接下来可能的下一班航班(不考虑虚拟航班)
NextFSameD = {j: [] for j in F}
for j in F:
    dptrDateInt_j = dfFlight.loc[j, "DptrDateInt"]
    arrvTimestamp_j = dfFlight.loc[j, "ArrvTimestamp"]
    arrvStnInt_j = dfFlight.loc[j, "ArrvStnInt"]
    possibleNextFlightFlag = (dfFlight["DptrStnInt"] == arrvStnInt_j) & (
        dfFlight["DptrTimestamp"] > arrvTimestamp_j)
    possibleNextFSameDayFlag = (
        dfFlight["DptrDateInt"] == dptrDateInt_j) & possibleNextFlightFlag
    if np.any(possibleNextFSameDayFlag):
        NextFSameD[j] = list(
            dfFlight.loc[possibleNextFSameDayFlag, "FltIndex"])

LastFSameD = {k: [j for j in FbyD[k] if not NextFSameD[j]]
              for k in D}  # 同一天里不存在接续航班的航班集合
NotLastFSameD = {k: list(set(FbyD[k])-set(LastFSameD[k]))
                 for k in D}  # 同一天里存在接续航班的航班集合

for k in D:
    assert set(LastFSameD[k]) | set(NotLastFSameD[k]) == set(FbyD[k])


def generateCrewPairsWithRole(dfCrew):
    dfCrew["Pre-Assigned"] = False
    dfCrew["Assigned"] = False
    baseList = list(pd.unique(dfCrew["BaseInt"]))
    crewPairs = {base: [] for base in baseList}
    maxFeasiblePairingByBase = {base: 0 for base in baseList}
    for base in baseList:
        baseFlag = dfCrew["BaseInt"] == base
        captainCount = dfCrew.loc[baseFlag, "Captain"].sum()
        officerCount = dfCrew.loc[(
            dfCrew["Captain"] == 0) & baseFlag, "FirstOfficer"].sum()
        benchCount = dfCrew.loc[(dfCrew["FirstOfficer"] == 1)
                                & baseFlag, "Captain"].sum()
        if captainCount <= officerCount:
            maxFeasiblePairingByBase[base] = captainCount
        else:
            maxFeasiblePairingByBase[base] = min(
                (captainCount - officerCount)//2, benchCount) + officerCount
        print("Base Int:", base)
        print("Captain count(include bench):", captainCount)
        print("First Officer(only) count:", officerCount)
        print("Bench count:", benchCount)
        print("maxFeasiblePairingCount:", maxFeasiblePairingByBase[base])

        captainOnlyFlag = (dfCrew["Captain"] == 1) & (
            dfCrew["FirstOfficer"] == 0) & baseFlag
```

```
officerOnlyFlag = (dfCrew["Captain"] == 0) & (
    dfCrew["FirstOfficer"] == 1) & baseFlag
benchOnlyFlag = (dfCrew["Captain"] == 1) & (
    dfCrew["FirstOfficer"] == 1) & baseFlag

captainLeftFlag = captainOnlyFlag & (~dfCrew["Pre-Assigned"])
officerLeftFlag = officerOnlyFlag & (~dfCrew["Pre-Assigned"])
# Exist captain(only) or officer(only)
while np.any(captainLeftFlag) and np.any(officerLeftFlag):
    captainChosen = dfCrew.loc[captainLeftFlag, "EmpNoInt"].idxmin()
    dfCrew.loc[captainChosen, "Pre-Assigned"] = True
    officerChosen = dfCrew.loc[officerLeftFlag, "EmpNoInt"].idxmin()
    dfCrew.loc[officerChosen, "Pre-Assigned"] = True
    crewPairs[base].append((captainChosen, officerChosen, 0, 1))
    captainLeftFlag = captainOnlyFlag & (~dfCrew["Pre-Assigned"])
    officerLeftFlag = officerOnlyFlag & (~dfCrew["Pre-Assigned"])

benchLeftFlag = benchOnlyFlag & (~dfCrew["Pre-Assigned"])
# Exist captain(Only) and bench
while np.any(captainLeftFlag) and np.any(benchLeftFlag):
    captainChosen = dfCrew.loc[captainLeftFlag, "EmpNoInt"].idxmin()
    dfCrew.loc[captainChosen, "Pre-Assigned"] = True
    officerChosen = dfCrew.loc[benchLeftFlag, "EmpNoInt"].idxmin()
    dfCrew.loc[officerChosen, "Pre-Assigned"] = True
    # exist alternate FO
    crewPairs[base].append((captainChosen, officerChosen, 0, 2))
    captainLeftFlag = captainOnlyFlag & (~dfCrew["Pre-Assigned"])
    benchLeftFlag = benchOnlyFlag & (~dfCrew["Pre-Assigned"])

# Exist officer(only) and bench
while np.any(officerLeftFlag) and np.any(benchLeftFlag):
    captainChosen = dfCrew.loc[benchLeftFlag, "EmpNoInt"].idxmin()
    dfCrew.loc[captainChosen, "Pre-Assigned"] = True
    officerChosen = dfCrew.loc[officerLeftFlag, "EmpNoInt"].idxmin()
    dfCrew.loc[officerChosen, "Pre-Assigned"] = True
    crewPairs[base].append((captainChosen, officerChosen, 0, 1))
    benchLeftFlag = benchOnlyFlag & (~dfCrew["Pre-Assigned"])
    officerLeftFlag = officerOnlyFlag & (~dfCrew["Pre-Assigned"])

# if Still exist bench
while np.any(benchLeftFlag):
    if(benchLeftFlag.sum() == 1):
        break
    captainChosen = dfCrew.loc[benchLeftFlag, "EmpNoInt"].idxmin()
    dfCrew.loc[captainChosen, "Pre-Assigned"] = True
    benchLeftFlag = benchOnlyFlag & (~dfCrew["Pre-Assigned"])
    officerChosen = dfCrew.loc[benchLeftFlag, "EmpNoInt"].idxmin()
    dfCrew.loc[officerChosen, "Pre-Assigned"] = True
    # exist alternate FO
```

```
            crewPairs[base].append((captainChosen, officerChosen, 0, 2))
            benchLeftFlag = benchOnlyFlag & (~dfCrew["Pre-Assigned"])

    return crewPairs


def createOneDuty(dayK, dfFlight, lastDutyEndTimestamp, startAirport, \
    base, lastDutyBackToBase=False):
    duty = []
    deadhead = []

    lastArrvTimestamp = copy(lastDutyEndTimestamp)
    lastArrvStnInt = copy(startAirport)
    connectionFlag = (dfFlight["DptrDateInt"] == dayK) & \
        (dfFlight["DptrStnInt"] == lastArrvStnInt) & (
        dfFlight["DptrTimestamp"]-lastArrvTimestamp >= MinCT)
    startFlag = (~dfFlight["Assigned"]) & connectionFlag
    if lastDutyEndTimestamp > 0:
        # Not first Duty; consider MinRest constrict
        startFlag = startFlag & (
            dfFlight["DptrTimestamp"]-lastArrvTimestamp >= MinRest)

    dutyTime = 0
    flightTime = 0
    # Normal case
    startFlight = None
    firstDeadheadOn = False
    if not np.any(startFlag):
        # No possible flight from this startPort -> deadhead
        # print("No possible flight from this start airport")
        connectionFlag = (dfFlight["DptrDateInt"] == dayK) & \
            (dfFlight["DptrStnInt"] == lastArrvStnInt) & (
            dfFlight["DptrTimestamp"]-lastArrvTimestamp >= MinCT)
        startFlag = (dfFlight["Assigned"]) & connectionFlag
        if lastDutyEndTimestamp > 0:
            startFlag = startFlag & (
                dfFlight["DptrTimestamp"]-lastArrvTimestamp >= MinRest)
        firstDeadheadOn = True
        if not np.any(startFlag):
            # No possible deadhead
            # print("No possible deadhead !!!")
            return [], [], lastDutyEndTimestamp, startAirport, dfFlight

    # 暂时不确定 nextFlight 是不是deadhead
    startFlight = dfFlight.loc[startFlag, "DptrTimestamp"].idxmin()
    # print(nextFlight)
    nextFlag = startFlag
    nextFlight = startFlight
    nextFlightTime = dfFlight.loc[nextFlight, "ArrvTimestamp"] - \
```

```
    dfFlight.loc[nextFlight, "DptrTimestamp"]
nextDutyTime = (dfFlight.loc[nextFlight, "ArrvTimestamp"] -
                dfFlight.loc[nextFlight, "DptrTimestamp"])
flightTime += nextFlightTime
dutyTime += nextDutyTime

while np.any(nextFlag) and dutyTime <= MaxDP and flightTime < MaxBlk:
    # print("DT:{},FT:{}".format(dutyTime,flightTime))
    if dfFlight.loc[nextFlight, "Assigned"]:
        deadhead.append(nextFlight)
    dfFlight.loc[nextFlight, "Assigned"] = True
    duty.append(nextFlight)

    lastArrvTimestamp = dfFlight.loc[nextFlight, "ArrvTimestamp"]
    lastArrvStnInt = dfFlight.loc[nextFlight, "ArrvStnInt"]

    connectionFlag = (dfFlight["DptrDateInt"] == dayK) &\
         (dfFlight["DptrStnInt"] == lastArrvStnInt) & (
        dfFlight["DptrTimestamp"]-lastArrvTimestamp >= MinCT)
    nextFlag = (~dfFlight["Assigned"]) & connectionFlag
    if np.any(nextFlag):
        nextFlight = dfFlight.loc[nextFlag, "DptrTimestamp"].idxmin()
        nextFlightTime = dfFlight.loc[nextFlight, "ArrvTimestamp"] - \
            dfFlight.loc[nextFlight, "DptrTimestamp"]
        flightTime += nextFlightTime
        nextDutyTime = (
            dfFlight.loc[nextFlight, "ArrvTimestamp"]-lastArrvTimestamp)
        dutyTime += nextDutyTime
# remove time caused by last not possible flight
if dutyTime > MaxDP or flightTime > MaxBlk:
    flightTime -= nextFlightTime
    dutyTime -= nextDutyTime

if len(duty) == 0:
    print("empty duty before check end deadhead")
    return [], [], lastDutyEndTimestamp, startAirport, dfFlight

# duty not on the last day
if not lastDutyBackToBase:
    # print("Avoid Surpassing flightTime or duty time ")
    return duty, deadhead, lastArrvTimestamp, lastArrvStnInt, dfFlight

# duty on the last day
lastFlight = duty[-1]
terminateAtBaseFlag = dfFlight.loc[lastFlight, "ArrvStnInt"] == base
if terminateAtBaseFlag:
    lastArrvTimestamp = dfFlight.loc[lastFlight, "ArrvTimestamp"]
    return duty, deadhead, lastArrvTimestamp, base, dfFlight
```

```
# consider deadhead
lastArrvStnInt = dfFlight.loc[lastFlight, "ArrvStnInt"]
lastArrvTimestamp = dfFlight.loc[lastFlight, "ArrvTimestamp"]
connectionFlag = (dfFlight["DptrDateInt"] == dayK) \
    & (dfFlight["DptrStnInt"] == lastArrvStnInt) & (
    dfFlight["DptrTimestamp"]-lastArrvTimestamp >= MinCT)
existDeadheadFlightEndAtBaseFlag = (dfFlight["Assigned"]) & (
    dfFlight["ArrvStnInt"] == base) & connectionFlag &\
        (dfFlight["ArrvTimestamp"]-lastDutyEndTimestamp <= MaxDP)
if np.any(existDeadheadFlightEndAtBaseFlag):
    terminateFlight = dfFlight.loc[existDeadheadFlightEndAtBaseFlag,\
        "DptrTimestamp"].idxmin(
    )
    duty.append(terminateFlight)
    terminateAtBaseFlag = True
    deadhead.append(terminateFlight)
    # print("find extra end deadhead")
    return duty, deadhead, dfFlight.loc[terminateFlight,\
        "ArrvTimestamp"], base, dfFlight

while not terminateAtBaseFlag and duty:
    # Remove the last flights that do not end at Base and search for deadhead
    if not(firstDeadheadOn and len(duty) == 1):
        dfFlight.loc[lastFlight, "Assigned"] == False
    duty.pop()
    if duty:
        lastFlight = duty[-1]
        terminateAtBaseFlag = dfFlight.loc[lastFlight,
                                           "ArrvStnInt"] == base
        lastArrvTimestamp = dfFlight.loc[lastFlight, "ArrvTimestamp"]
        lastArrvStnInt = dfFlight.loc[lastFlight, "ArrvStnInt"]
        if terminateAtBaseFlag:
            # print("Delete some flights and back to base")
            return duty, deadhead, lastArrvTimestamp, base, dfFlight
        else:
            connectionFlag = (dfFlight["DptrDateInt"] == dayK) \
                & (dfFlight["DptrStnInt"] == lastArrvStnInt) & (
                dfFlight["DptrTimestamp"]-lastArrvTimestamp >= MinCT)
            existFlightEndAtBaseFlag = (dfFlight["ArrvStnInt"] == base)\
                 & connectionFlag & (
                dfFlight["ArrvTimestamp"]-lastDutyEndTimestamp <= MaxDP)
            if np.any(existFlightEndAtBaseFlag):
                terminateFlight = dfFlight.loc[existFlightEndAtBaseFlag,\
                    "DptrTimestamp"].idxmin(
                )
                if dfFlight.loc[terminateFlight, "Assigned"]:
                    deadhead.append(terminateFlight)
                duty.append(terminateFlight)
                dfFlight.loc[terminateFlight, "Assigned"] = True
```

```
                    terminateAtBaseFlag = True
                    # print("Delete some flights and find deadhead")
                    return duty, deadhead, dfFlight.loc[terminateFlight,\
                        "ArrvTimestamp"], base, dfFlight
                else:
                    continue

    connectionFlag = (dfFlight["DptrDateInt"] == dayK) & (dfFlight["DptrStnInt"]\
        == startAirport) & (
        dfFlight["DptrTimestamp"]-lastDutyEndTimestamp >= MinCT)
    existFlightEndAtBaseFlag = (dfFlight["ArrvStnInt"] == base) &\
         connectionFlag & (
        dfFlight["ArrvTimestamp"]-lastDutyEndTimestamp <= MaxDP) &\
             (dfFlight["DptrTimestamp"]-lastDutyEndTimestamp >= MinRest)
    if np.any(existFlightEndAtBaseFlag):
        terminateFlight = dfFlight.loc[existFlightEndAtBaseFlag, "DptrTimestamp"].idxmin(
        )
        if dfFlight.loc[terminateFlight, "Assigned"]:
            deadhead.append(terminateFlight)
        duty.append(terminateFlight)
        dfFlight.loc[terminateFlight, "Assigned"] = True
        terminateAtBaseFlag = True
        # print("Delete some flights and find deadhead")
        return duty, deadhead, dfFlight.loc[terminateFlight, "ArrvTimestamp"], base, dfFligh
    # empty duty
    # print("No duty until the end function call")
    return [], [], lastDutyEndTimestamp, startAirport, dfFlight


dutyPattern = []
if DATA == "A":
    pat1 = [1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1]
    pat2 = [0, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 0, 1, 1]
    dutyPattern = [pat1, pat2]
elif DATA == "B":
    pat1 = [1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1,
            0, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1]
    pat2 = [0, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 0, 1, 1,
            1, 1, 0, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 1, 1]


# greedy by duty
crewPairsWithRoles = generateCrewPairsWithRole(
    dfCrew)  # base_i:[(capId, officerID, role1, role2)]
crewPairPrevDutyEndTimestamp = {}
crewPairCurLocation = {}
for base, crewPairRoleList in crewPairsWithRoles.items():
    for crewPair in crewPairRoleList:
        crewPairPrevDutyEndTimestamp[(crewPair[0], crewPair[1])] = 0
```

```
        crewPairCurLocation[(crewPair[0], crewPair[1])] = base

X = []
dfFlight["Assigned"] = False

#################
#### DFS ########
#################

print(D)

for base, crewPairRoleList in crewPairsWithRoles.items():
    cnt = 0
    for crewPair in crewPairRoleList:
        print(crewPair)
        crew1, crew2, role1, role2 = crewPair
        for dayK in D:
            if cnt % 2 == 0 and pat1[dayK] == 0:
                continue
            elif cnt % 2 == 1 and pat2[dayK] == 0:
                continue
            else:
                isLastDuty = False
                if dayK == max(D):
                    isLastDuty = True
                duty, deadheadList, dutyEndTimestamp, dutyEndAirport, dfFlight\
                     = createOneDuty(dayK, dfFlight,\
                          crewPairPrevDutyEndTimestamp[(crew1, crew2)], \
                              crewPairCurLocation[(crew1, crew2)], base, isLastDuty)
                # print(duty)
                # print(deadheadList)
                assert set(deadheadList).issubset(set(duty))
                if not duty:
                    continue

                crewPairPrevDutyEndTimestamp[(crew1, crew2)] = dutyEndTimestamp
                crewPairCurLocation[(crew1, crew2)] = dutyEndAirport

                for flight in list(set(duty)-set(deadheadList)):
                    X.append((crew1, flight, role1))
                    X.append((crew2, flight, role2))
                for _flight_ in deadheadList:
                    X.append((crew1, _flight_, -1))
                    X.append((crew2, _flight_, -1))
        print(crewPairCurLocation[(crew1, crew2)])
        cnt += 1

X = np.array(X)
fltCovered = np.unique(X[:, 1])
```

```
print("flight count:", dfFlight["Assigned"].sum())
print("flight count:", len(fltCovered))
print("captain count:", np.sum(X[:, 2] == 0))
print("officer count:", np.sum(X[:, 2] == 1))
print("alternate count:", np.sum(X[:, 2] == 2))
print("deadhead count:", np.sum(X[:, 2] == -1))

lastTime = time.time() - startTime
print("执行时间：{:.4f} s".format(lastTime))

##################################
###### 生成对应的数据 ############
##################################

notAssignedFlightIndexList = list(set(F)-set(np.unique(X[:, 1])))
outputDir = "./task{}/".format(TASK) + DATA + "_heuristic/" + "/"
if not os.path.exists(outputDir):
    os.makedirs(outputDir)
fileUncoveredFlights = outputDir + "UncoveredFlights.csv"
fileCrewRosters = outputDir + "CrewRosters.csv"

# Generate UncoveredFlights.csv
dfUncoveredFlight = dfFlight.loc[notAssignedFlightIndexList, :]
if dfUncoveredFlight.empty:
    print("All flights covered")
    dfUncoveredFlight = pd.DataFrame(columns=[
                                    "FltNum", "DptrDate", "DptrTime",
                                     "DptrStn", "ArrvDate", "ArrvTime",
                                      "ArrvStn", "Comp"])
else:
    # 存在未被执行的航班
    print("Find flights uncovered")
    dfUncoveredFlight = dfUncoveredFlight[[
        "FltNum", "DptrDate", "DptrTime", "DptrStn", "ArrvDate",
         "ArrvTime", "ArrvStn", "Comp"]]
    dfUncoveredFlight = dfUncoveredFlight.sort_values(
        by=["DptrDate", "DptrTime", "DptrStn", "ArrvStn"])
print("Uncovered Flights:")
print(dfUncoveredFlight)
dfUncoveredFlight.to_csv(fileUncoveredFlights, sep=",", index=False)



# Generate CrewRosters.csv
dfResult = pd.DataFrame(X, columns=["EmpNoInt", "FltIndex", "Role"])  # 包含-1,-2
dfResult = dfResult.loc[~((dfResult["FltIndex"] == -1)
                          | (dfResult["FltIndex"] == -2)), :]  # 不再包含-1,-2
dfResult = pd.merge(dfResult, dfCrew, how="left", on="EmpNoInt")
dfResult = pd.merge(dfResult, dfFlight, how="left", on="FltIndex")
dfResult = dfResult.sort_values(
```

```
    by=["EmpNoInt", "DptrDateInt", "FltIndex"], ignore_index=True)

dfResult["Task"] = ""
for i in range(dfResult["Role"].count()):
    task = ""
    if (dfResult.loc[i, "Role"] == 0):
        task = "Captain"
    elif (dfResult.loc[i, "Role"] == 2):
        task = "AlternateFirstOfficer"
    elif (dfResult.loc[i, "Role"] == 1):
        task = "FirstOfficer"
    elif (dfResult.loc[i, "Role"] == -1):
        task = "Deadhead"
    else:
        print("Error")
        assert False
    dfResult.loc[i, "Task"] = task

dfCrewRosters = dfResult[["EmpNo", "Captain", "FirstOfficer",
"Deadhead", "Task", "Base","FltNum", "DptrDate", "DptrTime",
"DptrStn", "ArrvDate", "ArrvTime", "ArrvStn", "Comp"]]

print(dfCrewRosters.head())
dfCrewRosters.to_csv(fileCrewRosters, index=False)
```

```
########################################
##### 打印统计量          ###############
########################################

import numpy as np
import pandas as pd
import os
import json
from ast import literal_eval as make_tuple

DATA = "A"
TASK = 2  # 2 or 3
FILENAME = "____________________"


DATA_PATH = "./data/"
PREPARED_CREW_FILE = os.path.join(
    DATA_PATH, "prepared_crew_{}.csv".format(DATA))
PREPARED_FLIGHT_FILE = os.path.join(
    DATA_PATH, "prepared_flight_{}.csv".format(DATA))
OUTPUT_DIR = "./task{}/".format(TASK) + DATA + "/"


def statistics(rawOutputFile):
    dfCrew = pd.read_csv(PREPARED_CREW_FILE, sep=",")
    dfFlight = pd.read_csv(PREPARED_FLIGHT_FILE, sep=",")
    # 员工集合
    I = list(np.sort(pd.unique(dfCrew["EmpNoInt"])))
    # 航班集合
    F = list(np.sort(pd.unique(dfFlight["FltIndex"])))
    # 日期集合
    D = list(np.sort(pd.unique(dfFlight["DptrDateInt"])))

    with open(rawOutputFile, "r") as f:
        result = json.load(f)
    Y = result["Y"]
    crewConsecFlights = np.array(
        [make_tuple(key) for key, value in Y.items() if np.round(value) == 1])
    dfResultY = pd.DataFrame(crewConsecFlights, columns=[
                             "EmpNoInt", "FltIndex1", "FltIndex2"])

    X = result["X"]
    crewFlightRoles = np.array([make_tuple(
        key) for key, value in X.items() if np.round(value) == 1])
        # FltIndex包含-1, -2
    notAssignedFlightIndexList = list(set(list(
        dfFlight["FltIndex"])) - set(np.unique(crewFlightRoles[:, 1])))
        # 集合差集除去-1, -2
    assert -1 not in notAssignedFlightIndexList
```

```
assert -2 not in notAssignedFlightIndexList

print("不可执行的航班数量：", dfFlight["FltIndex"].count(
)-int(result['objMaxFlightNum']))
print("可执行的航班数量：", int(result['objMaxFlightNum']))
print("总体乘机次数：", result['objMinDeadheadNum'])
print("替补资格使用次数：", result['objMinBenchNum'])

crewFlight = crewFlightRoles[:, :2]
crewFlight = crewFlight[crewFlight[:, 1] >= 0, :]
flightTime = {(i, k): 0 for i in I for k in D}
for cnt in range(crewFlight.shape[0]):
    i = crewFlight[cnt, 0]
    fltIndex = crewFlight[cnt, 1]
    k = dfFlight.loc[fltIndex, "DptrDateInt"]
    flightTime[(i, k)] += (dfFlight.loc[fltIndex, "ArrvTimestamp"] -
                           dfFlight.loc[fltIndex, "DptrTimestamp"])
flightTimeList = [value for value in flightTime.values()]
print("最小一次执勤飞行时长：", np.min(flightTimeList))
print("平均一次执勤飞行时长：", np.mean(flightTimeList))
print("最大一次执勤飞行时长：", np.max(flightTimeList))

DTdict = {make_tuple(key): value for key, value in result['DT'].items()}
DT = np.zeros((len(I), len(D)), dtype=np.int64)
for i in I:
    for k in D:
        DT[i, k] = DTdict[(i, k)]
print("最小一次执勤执勤时长：", np.min(DT))
print("平均一次执勤执勤时长：", np.mean(DT))
print("最大一次执勤执勤时长：", np.max(DT))

print("机组总体利用率：{:.4f}%".format(100*np.sum(flightTimeList)/np.sum(DT)))

dutyDayCount = [np.sum(DT[i, :] > 0) for i in I]
print("最小一次机组人员执勤执勤天数：", np.min(dutyDayCount))
print("平均一次机组人员执勤执勤天数：", np.mean(dutyDayCount))
print("最大一次机组人员执勤执勤天数：", np.max(dutyDayCount))

print("总体执勤成本", int(result['objMinTotalDutyCost']))

if TASK == 3:
    OnDuty = {make_tuple(key): value for key,
              value in result['OnDuty'].items()}  # OnDuty[(i,k)]
    pairingTimeCount = {d: 0 for d in range(1, 11)}  # 任务环不超过10天
    Z = result["Z"]
    arrZ = np.array([make_tuple(key) for key, value in Z.items()
                     if np.round(value) == 1])  # i,j1,j2
    for row in range(arrZ.shape[0]):
        j1 = arrZ[row, 1]
```

```
            j2 = arrZ[row, 2]
            dptrDateIntJ1 = dfFlight.loc[j1, "DptrDateInt"]
            dptrDateIntJ2 = dfFlight.loc[j2, "DptrDateInt"]
            d = dptrDateIntJ2 - dptrDateIntJ1 + 1
            pairingTimeCount[d] += 1
        for dd, cnt in pairingTimeCount.items():
            print("时长为{}天任务环数量：{}".format(dd, cnt))


if __name__ == "__main__":
    statistics(FILENAME)
```

```
########################################
##### 生成要求的输出          ############
########################################


import numpy as np
import pandas as pd
import os
import json
from ast import literal_eval as make_tuple

DATA = "A"
TASK = 1


DATA_PATH = "./data/"
PREPARED_CREW_FILE = os.path.join(
    DATA_PATH, "prepared_crew_{}.csv".format(DATA))
PREPARED_FLIGHT_FILE = os.path.join(
    DATA_PATH, "prepared_flight_{}.csv".format(DATA))
OUTPUT_DIR = "./task{}/".format(TASK) + DATA + "/"


def generate_output(rawOutputFile, outputDir):
    dfCrew = pd.read_csv(PREPARED_CREW_FILE, sep=",")
    dfFlight = pd.read_csv(PREPARED_FLIGHT_FILE, sep=",")

    with open(rawOutputFile, "r") as f:
        result = json.load(f)
    Y = result["Y"]
    crewConsecFlights = np.array(
        [make_tuple(key) for key, value in Y.items() if np.round(value) == 1])
    dfResultY = pd.DataFrame(crewConsecFlights, columns=[
                             "EmpNoInt", "FltIndex1", "FltIndex2"])

    X = result["X"]
    crewFlightRoles = np.array([make_tuple(
        key) for key, value in X.items() if np.round(value) == 1])
         # FltIndex包含-1, -2
    notAssignedFlightIndexList = list(set(list(
        dfFlight["FltIndex"])) - set(np.unique(crewFlightRoles[:, 1])))
         # 集合差集除去-1, -2
    assert -1 not in notAssignedFlightIndexList
    assert -2 not in notAssignedFlightIndexList

    if not os.path.exists(outputDir):
        os.makedirs(outputDir)
    fileUncoveredFlights = outputDir + "UncoveredFlights.csv"
    fileCrewRosters = outputDir + "CrewRosters.csv"
```

```
# Generate UncoveredFlights.csv
dfUncoveredFlight = dfFlight.loc[notAssignedFlightIndexList, :]
if dfUncoveredFlight.empty:
    print("All flights covered")
    dfUncoveredFlight = pd.DataFrame(columns=[
                                        "FltNum", "DptrDate",
                                        "DptrTime", "DptrStn",
                                         "ArrvDate", "ArrvTime",
                                          "ArrvStn", "Comp"])
else:
    # 存在未被执行的航班
    print("Find flights uncovered")
    dfUncoveredFlight = dfUncoveredFlight[[
        "FltNum", "DptrDate", "DptrTime", "DptrStn", "ArrvDate",
         "ArrvTime", "ArrvStn", "Comp"]]
    dfUncoveredFlight = dfUncoveredFlight.sort_values(
        by=["DptrDate", "DptrTime", "DptrStn", "ArrvStn"])
print("Uncovered Flights:")
print(dfUncoveredFlight)
dfUncoveredFlight.to_csv(fileUncoveredFlights, sep=",", index=False)

# Generate CrewRosters.csv
dfResult = pd.DataFrame(crewFlightRoles, columns=[
                        "EmpNoInt", "FltIndex", "Role"])  # 包含-1,-2
dfResult = dfResult.loc[~(
    (dfResult["FltIndex"] == -1) | (dfResult["FltIndex"] == -2)), :]  # 不再包含-1,-2
dfResult = pd.merge(dfResult, dfCrew, how="left", on="EmpNoInt")
dfResult = pd.merge(dfResult, dfFlight, how="left", on="FltIndex")
dfResult = dfResult.sort_values(
    by=["EmpNoInt", "DptrDateInt", "FltIndex"], ignore_index=True)

dfResult["Task"] = ""
for i in range(dfResult["Role"].count()):
    task = ""
    if (dfResult.loc[i, "Role"] == 0) and dfResult.loc[i, "Captain"]:
        task = "Captain"
    elif (dfResult.loc[i, "Role"] == 1) and (dfResult.loc[i,\
         "Captain"] == 1):
        task = "AlternateFirstOfficer"
    elif (dfResult.loc[i, "Role"] == 1) and (dfResult.loc[i,\
        "FirstOfficer"] == 1) and (dfResult.loc[i, "Captain"] == 0):
        task = "FirstOfficer"
    elif (dfResult.loc[i, "Role"] == 2):
        task = "Deadhead"
    else:
        print("Error")
        assert False
    dfResult.loc[i, "Task"] = task
```

```
    dfCrewRosters = dfResult[["EmpNo", "Captain", "FirstOfficer",
    "Deadhead", "Task", "Base","FltNum", "DptrDate", "DptrTime",
    "DptrStn", "ArrvDate", "ArrvTime", "ArrvStn", "Comp"]]

    ##########################################
    ##### Task 3 需要增加 休假日标注输出 #######
    ##########################################

    print(dfCrewRosters.head())
    dfCrewRosters.to_csv(fileCrewRosters, index=False)

    return dfUncoveredFlight, dfCrewRosters


if __name__ == "__main__":
    rawOutputFile = "./raw_results_task1_3600s.json"
    dfUncoveredFlight, dfCrewRosters = generate_output(
        rawOutputFile, OUTPUT_DIR)
```